**Advanced IAM**

**Web identity federation:**

Let you give your users access to AS resources after they have successfully authenticated with a web-based identity provider like Amazon, Facebook and Google.

Following successful authentication, the user receives as authentication code from Web ID provider, which they can trade for temporary AWS security credentials.

**Amazon Cognito:**

AWS Cognito provides web identity federation with the following feature:

* Sign-up and sign-in to your apps.
* Access for guest users.
* Acts as *identity broker between your application and web ID providers*, so you don’t need to write any additional code.
* *Synchronizes user data for multiple devices*. Seamless experiences.
* Recommended for all mobile application AWS services.

**Amazon Cognito Use cases:**

Cognito broker between the app and Facebook or Google to provide temporary credential which map to an IAM role allowing access to the required resources.

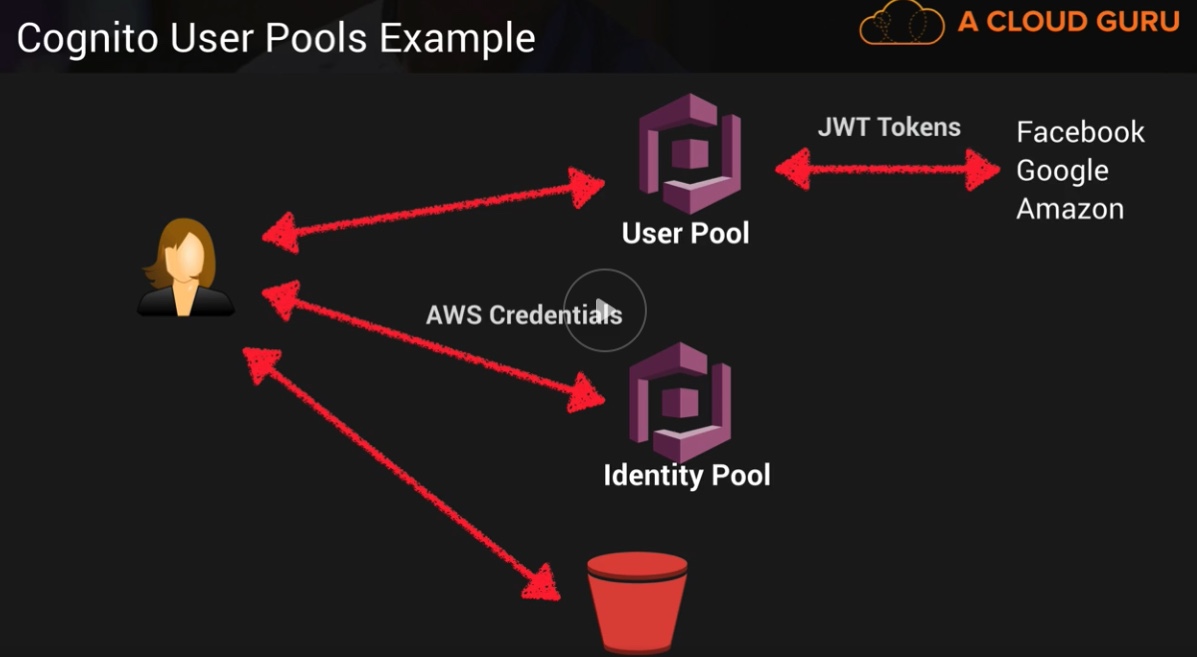
No need for the application to embed or store AWS credential locally on the device and it gives user a seamless experience across all mobile devices.

**Cognito User pools**

**User Pools** are user directories used to manage sign-up and sign-in functionalities for mobile and web application.

User can sign-in directly to user Pool, or indirectly via identity provider like Facebook or Google. Cognito acts as an Identity Broker between the ID provider and AWS. Successful authentication generates a number of JSON web token (JWTs).

**Identity Pools** *enable you to create unique identities for your user and authenticate them with identity providers*. With an identity, you can obtain temporary, limited privilege AWS credentials to access other AWS services.

****

**Push Synchronization:**

Cognito tracks the association between user identity and the various different devices they sign-in-from.

In order to provide the seamless user experience for your application, Cognito uses **Push**

**Synchronization** to push updates and Synchronize user date across multiple devices.

Amazon SNS is used to send the silent push notification to all the devices associated with a given identity whenever data stored in the cloud changes.

**Inline Policies vs Managed Polices vs Custom Policies**

**Manages Policies.**

AManaged Policy is an IAM policy which is created and administered by AWS.

AWS provide Managed Policies for common use cases bases on job function. E.g. AmazonDynamoDBFullAccess, AWSCodeCommitPowerUser, AmazonEC2ReadOnlyAccess etc.

These AWS-provided policies allow you to assign appropriate permission to your users, group and role without having to write the policy yourself.

A single Managed Policy be *attached to multiple user, group or roles* within the same AWS account and across different accounts.

You cannot change the permission defined in an AWS Manages Policy.

AWS *occasionally update the permission* defined in an AWS managed policy.

**Inline Policy:**

An inline policy is an IAM policy which is actually *embedded within the user, group or role* to which it applies, there is strict 1:1 relationship between the entity and the policy.

*When you delete the user, group or role in which the inline policy is embedded, the policy is also be deleted*.

In most cases, AWS recommends using Managed Policies or inline Policies.

Inline Policies are useful when you want to be sure that the permission in a policy are not inadvertently assigned to any other user, group or role than the one for which they’re intended. (i.e. You are creating a policy that must only ever be attached to a single user, group or role.)

**Customer Managed Policy:** Managed by you.

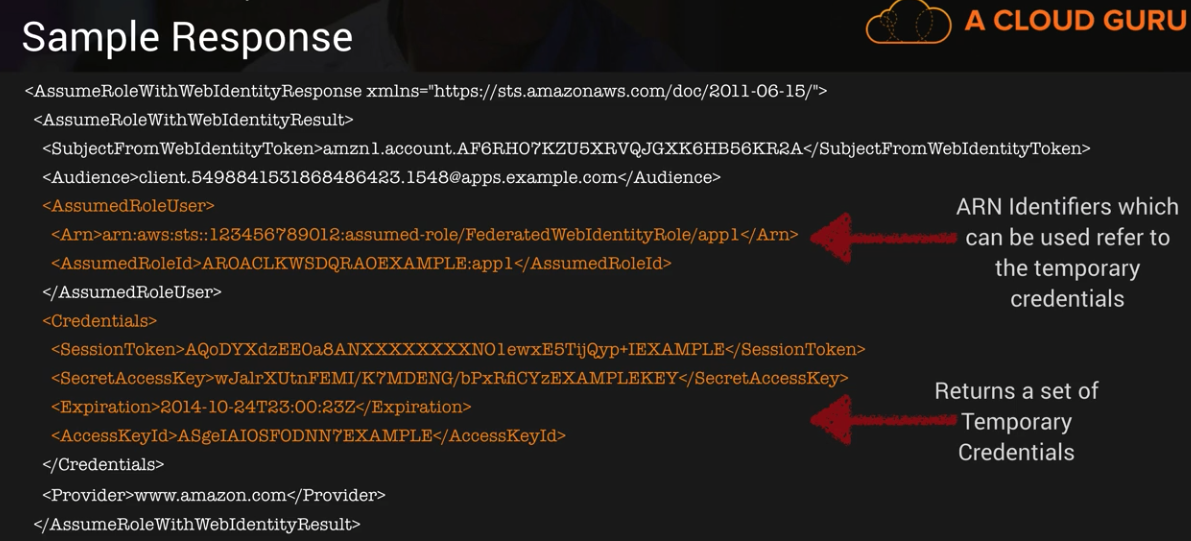
It can assigned to multiple user, group or role in your account.

* In most cases, AS recommends using Managed policies over inline Policies.

**STS AssumeRoleWithWebIdentity**

* *assume-role-with-web-identity* is an API provided by STS (Security Token service).
* Returns temporary security credentials for users authenticated by mobile or web application or using a Web ID provider like Facebook, google or Amazon ect.
* For mobile application, Cognito is recommended.
* Regular web ap5plication can use the STS *assume-role-with-web-identity API.*





If successful, STS will return temporary credential enabling access to AWS recourses.

* AssumedRoleUser ARN and AssumeRoleID- are used to programmatically reference the temporary credential- not an IAM role or user.

**Cross Account Access**

Many AWS customers use separate AWS accounts for their development and production resources. This separation allows them to cleanly separate different types of resources and can also provide some security benefits.

Cross account access makes it easier to you to work productively within a multi-account (or multi role) AWS environment by making it easy for you to switch roles within the AWS Management console. You can now sign in to the console using IAM username then switch the console to manage another account without having to enter (or remember) another user name and password.

~Steps (High level in Exam):

* Identify our account numbers.
* Create a group in IAM-dev.
* Create a user in IAM-Dev.
* Log into production
* Create a “read-write-app-bucket” policy.
* Create the “UpdateApp” Cross Account ~role.
* Apply the newly created policy to the role.
* Log into Developer account.
* Create a new inline policy.
* Apply a new inline policy.
* Apply it to the Developer group.
* Login as john
* The RDS instances unlike the AWS the AWS EBS backed instances cannot be stopped or paused. The user needs to take the final snapshot, terminate the instances and launch a new instance in future from that snapshot.

**DynamoDB Provisioned Throughput**

* DynamoDB provisioned throughput is measured in Capacity units.
* When you create your table, you specify your requirement in terms of Read Capacity Units and write Capacity Units.
* 1 x Write Capacity Unit = 1 x 1KB write/second.
* 1 x Read Capacity Units = 1 x Strongly Consistent Read of 4KB/Second.

OR

2 x Eventually Consistent Reads of 4KB/second.

* Billing commences when Amazon EC2 initiate the boot sequences of an AMI instance. Billing ends when instances shut down. Which could occur through a web services command, by running “shutdown -h” or through instances failure.
* You can configure ELB to use SSL certificate in order to improve your system security. The load balancer uses the certificate to terminate and decrypt request before sending them to back-end instances. ELB uses AWS IAM to upload your certificate to your load balancer.
* User can add additional EC2 instances to load balancer on the fly. You can add or remove EC2 instances from your load balancer as your needs change, without disrupting the overall flow of information.
* An IAM group is a collection of IAM users. Groups let the user specify permission for a collection of users. Which can make it easier to manage the permission for those users.
* SQS message retention period => *1 minute to 14 days*. The *default is 4 days* and once the message retention limit reached your messages will be automatically deleted.
* In DynamoDB

Database – Collection of tables.

Table - Collection of items.

Item – Collection of Attribute.

* DynamoDB SSDs help achieves design goal of predictable ***low-latency response*** times for storing and accessing data at any scale. The high I/O performance of SSDs also enables to serve high-scale request workloads cost efficiently, and pass the efficiently along in low request pricing.
* A few of the security measure are listed below.

~ Always keep updated OS update.

~ Not allow IAM user to connect with EC2 instances

~ Disable the password-based login for all the users.

~ Create a procedure to revoke the access rights of the individual user when they are not required to connect to EC2 instance anymore for the purpose of allocation configuration.

~ Lock down unnecessary ports.

~ Audit any proprietary application that the user may be running on the EC2 instances provide temporary escalation privileges, such as sudo for users who need to perform occasional privileged tasks.

~ It is not useful to connect RDP/SSH with an instance.

* In DynamoDB, DeleteItem deletes a single item in table by primary key, but BatchDeleteItem does not exist.
* Valid Operation: BatchWriteItem, DescribeTable, BatchGetItem, DeleteItem.
* Scan operation are always eventually consistent.
* In Amazon SNS, when you want to subscribe to a topic and receive notification to your email. *Select Email in protocol drop down box*. Enter an email address you can use to receive the notification in endpoint field.
* *Basic is the type of monitoring data* (EBS) which is available automatically in *5 minutes at no charge called.*
* **DescribeTable** return information about table. Including the current status of the table, when it was created, The **primary key schema, and indexes on the table**. This is used to **get detailed listing of secondary indexes on the table**.
* EC2 once terminated by user, the user can find details about termination from the description tab under the label State transition reason. If the user has explicitly stopped or terminated the instance, the reason will be “User Initiated shutdown”.
* AWS (Amazon Simple workflow) is a task coordination and state management service for cloud application.
* When you create a table with hash-and-range key, it is optional to defined one or more secondary indexes on the table.
* RDS DB instance storage comes in two type

**Standard Storage**: Standard storage is allocated on the EBS volume and connected to the user’s DB instance.

**Provisioned IOPS**: Provisioned IOPS uses optimized EBS volumes and optimized configuration stack. It provides, dedicated capacity for the EBS I/O.

* When a user is trying to detach an EBS volume, the user can either terminate the instance or explicitly remove the volume, it is recommended practice to unmount the volume first to avoid any file system damage.
* DynamoDB support following data types:

**Scaler data Types**: Number, String, Binary

**Multi Valued Type**: String set, Number set and Binary set.

* Like SNS other send notification messages services are:

1. Google Cloud Messaging for Android. (GCM).
2. Apple Push Notification service. (APNS)
3. Amazon Device Messaging. (ADM)

* RDS provides a managed DB platform, which offers feature, such as automated backup, patch management, automated failure detection and recovery. The scaling is not automated the user needs to plan it with few clicks.
* S3 Objects stored in the bucket before the user has set the versioning state have a version ID of null. When the user enables versioning, the objects in the bucket do not change and their ID remain null.
* SQS allows the user to move data between distributed components so they can perform different task without losing messages or requiring each component to be always available. The ***user can delete a queue at any time***. **whether it is empty or not**.it is impotent to note that queue retain messages for a set period of time, **by default a queue retain messages for 4 days.**
* Speaking about DynamoDB, **if your application performs more reads/second or writes/second than your table’s provisioned throughput capacity allows,** requests above your provisioned capacity will be **throttled and you will receive 400 error code**.
* **User can** define **multiple private IP address** for his instance. The number of network interfaces and private IP address that a user can specify for an instance **depends on the instance type**. This scenario helps when user wants to host multiple website on a single EC2 instance. After the user has assigned a secondary private address for his instance, he needs to configure the OS on that instance to recognize the secondary private IP address. AWS Linux, the ***ec2-net-utils* package can take care for these steps**. It configures additional network instances that user can attached while the instance is running, refreshes secondary IP address during DHCP lease renewal, and updates the related routing rules.
* **SQS** supports an unlimited number of queue and unlimited number of messages per queue for each user. SQL automatically delete messages that have been in the queue for more than 4 days.
* In Amazon VPC, you can assign any private address to your instance as long as it is:

1. Part of associated subnet IP address range.
2. Not reserved by amazon IP networking purpose.
3. Not currently assigned to another interface.

* Queue name restriction.

1. 80 char
2. Alphanumeric plus hyphen (-) and underscore (\_) allowed.
3. Name must be unique.
4. After delete the queue, you can reuse the queue name.

* To send push notification to mobile devices using SNS and A~DM, you need to obtain 1) Registration ID

2) Client secret.

* To begin SNS mobile push notification, you first need an app for the mobile endpoint that uses one of the supported push notification services: APNS, GCM or ADM. After you ‘have registered and configured the app to use one of these services. You configure SNS to send push notification to mobile endpoint.
* The user can get notification using SNS if he has configured the notification while creating the auto scaling group.
* Before Auto scaling selects an instance to terminate, it first identifies the AZ that has more instances than the other AZ used by the group. If all the AZ have the same number of instances, it identifies a random AZ.
* In DynamoDB, an index cannot be modified once it is created.
* AWS Elastic Beanstalk provides an environment to easily develop and run application in the cloud. It is integrated with developer tools and provides a one-stop experiences for you to manage the lifecycle of your applications.
* The instances that reside in the **private subnets** of your VPC are not reachable from the internet, meaning that is **not possible to SSH into them**. To interact with them you can use a **bastion server, located in a public subnet**, that will act as proxy of them. **You can also connect if you have direct connected or VPN.**
* AWS Cloud formation supports Amazon EC2 tagging.
* When creating an RDS instance, the user needs to specify whether it is Multi AZ or not. If the user does not provide the value for the zone. The maintenance window or automated backup window, RDS will automatically select the value.

If the user is launching RDS with Multi AZ the user cannot provision the Availability Zone. RDS is launched automatically instead.

* If the user does not specify a preferred backup window while enabling an automated backup, Amazon RDS assign a default 30-minute backup window which is selected at random from an 8-hour block of time per region.
* If you need to host multiple websites (with different IPs) on a single EC2 instance, the following is the suggested method from AWS.

Launch a VPC instance with two network interfaces.

Assign elastic IPs from VPC EIP pool to those interfaces (Because, when the user has attached more one network interface with an instance, AWS can not assign public IPS to them.). Assign separate security groups if separate security groups are needed This scenario also helps for operating network appliances. Such as firewalls or load balancers that have multiple private IP addresses for each network interface.

* DynamoDB automatically replicates your data synchronously across multiple Availability zones within an AWS Region to ensure high-availabilities and data durability.
* In Amazon SWF, the **coordination logic** in a workflow is contained in a software program called a **decider**. The decider schedules activity task, provides input data to the activity workers, processes events that arrive while the workflow is in progress, and ultimately ends (or closes) the workflow when the objective has been completed.
* When the user makes any changes to the RDS security group the rule *status will be authorizing* for some time until the changes are applied to all intendances that the group connected with. *Once the changes are propagated the rule status will change to authorized.*
* When a user creates an EBS volume and attached it as a device, it is required to mount the device. It the device/volume not mounted it will not be available in the listing.
* If the user wants to temporarily stop the access to S3 the best solution is to disable the keys.
* If an EBS volumes stays in the detaching state, the user can force the detachment by clicking Force Detach. Forcing the detachment can lead to either data loss or a corrupted file system. The user should use this option only as a last resort to detach a volume from a failed instance or if he is detaching a volume with the intention of deleting it.
* To configure the Auto scaling termination policy, the user can either specify any one of the policies as a standalone policy or list multiple policies in an ordered list. The policies are executed in the order that they are listed.
* The user can configure the Autoscaling group to automatically scale up and then scale down based on the specified condition. To configure this, the user must **setup policies** which will **get triggered by the CloudWatch alarms**.
* In DynamoDB, you can increase the throughput you have provisioned for your table using updateTable API or in the AWS Management console. If you wish to exceed throughput rates of 10,000 writes/sec or 10,000 read/sec, you must first contact AWS.
* IN AWS Elastic Beanstalk, you can update your deployed application, even while it is part of a running environment, for a java application, you can also use the AWS Toolkit for Eclipse to update your deployed application.
* Amazon SQS is a distributed queuing system that is optimized for horizontal scalability, not for single threaded sending or receiving speeds.
* The IAM policy is never region specific. If the user wants to configure the region-specific setting, he needs to provide conditions as part of the policy.
* DynamoDB has seamless scalability with no table size limits and unlimited storage, so you shouldn’t be worried about managing storage on the host or to provisioning more drive, as your data requirement changes.

**Elastic Beanstalk**

Elastic beanstalk is a service for deploying and scaling web application developed in many popular language Java, .Net, PHP, NodeJS, python, Go and ~Docker onto widely used application server platform like Apache Tomcat, NgInx, Passenger and IIS.

~ Developer can focus on writing code and don’t need to worry about any of the underlying infrastructure needed to run the application.

~ You upload the code and Elastic Beanstalk will handle deployment, capacity provisioning, load balancing, auto scaling and application health.

~ You retain full control of the underlying AWS recourses powering your application and you pay only for the AWS resources required to store and run your applications (e.g. EC2 instances and S3 buckets.)

~ Monitor and manage application health via a dashboard.

~ Integrated with CloudWatch and X-ray for performance data metrices.

~ AWS Elastic Beanstalk is designed to support multiple running environment, as an example you could have one for integrated testing, one for pre-production, and one for production, with each environment independently configured and running on its own separate AWS resources.

~ You can customize Elastic Beanstalk environment using Elastic Beanstalk configuration files. (e.g. you can define packages to install, create Linux user and group, run shell commands, specify services to enable or configure your load balancer etc.)

~ These are files written in YAML or JSON format. They can have a filename of your choice but must have a .config extension and saved inside a folder called. ebextensions**.**

~ The. ebextensions folder must be included into the top-level directory of your application source code bundle.

~ This means that the configurations files can be placed under source control along with the rest of your application code.

* If an Amazon EBS volume is root device of an instance, it cannot be detached unless the instance is in the stopped state.

**Lambda**

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

~ Scale out (not up) automatically.

~ Lambda function are independent, 1 event = 1 function

~ Lambda function can trigger other lambda functions, 1 event can = x function if functions trigger other function.

~ Architecture can get extremally complicated, AS X-ray allows you to debug what is happening.

~ Lambda can do things globally, you can use it to back up S3 buckets to other S3 buckets etc.

~ As an event-driven compute service where AWS Lambda runs your code in response of events. These events could be changes to data in an AWS S3 bucket or an AWS DynamoDB table.

**How is Lambda priced?**

Number of requests.

* + First 1 million requests are free. $0.20 per million requests thereafter**.**

**Duration**

* + Duration is calculated from the time your code begins executing until it returns or otherwise terminates, rounded up to the nearest 100ms. The price depends on the amount of memory you allocate to your function. You are charged $0.00001667 for every GB-second used.

**Why is lambda cool?**

* + No SERVES.
  + Continuous scaling
  + Super cheap.
* DynamoDB with no table size limit and unlimited storage.
* If the use launching RDS with multi AZ the user cannot provision the AZ. RDS is launched automatically instead.
* AWS Elastic Beanstalk is designed to support multiple running environment. As an example, you could have one for integration testing, one for pre-production, and one for production, with each environment independently configured and running on its own separate resources.
* If an Amazon EBS volume is the root device, it cannot be detached unless the instance is in the stopped state.
* SQS maximum message size is 256KB.
* The user cannot authorize an EC2 security group if it is in different AWS region than the RDS instance. The user can authorize an IP range or specify an amazon Ec2 security group in the same region that refer to an IP address in another region.
* A stack is the set of AWS resources that are created and managed as single unit when AWS CloudFormation initiate a template.
* Global secondary index: An index with hash and range key that can be different from those on the table.
* If you have predictable load changes, you can set a schedule through Auto scaling to plan your scaling activities. You can use CloudWatch to send alarms trigger scaling activities and ELB to help distribute traffic to your instance with Auto scaling groups. Auto scaling enables you to run Amazon EC2 fleet at optimal utilization.
* The webserver is running but the user is not able to access the website from the internet. The possible reason for this failure is the security group of the instance is not properly configured.
* AWS Elastic beanstalk source bundle can be:

Zip/war file.

Not exceed 512MB.

Not include top or parent folder.

* To host a static website, the user needs to configure an S3 bucket for website and then upload the website content into the bucket. The website is then available at the region-specific website endpoint of the bucket.
* Every SQS queue has visibility timeout. For the designated amount of time after a message is read from queue. It will not be visible for any other reader. As long as the amount of time that it takes to process the message is less than the visibility timeout, every message will be processes and deleted. In the event that the component processing the message fails or becomes unavailable, the message will be again become visible to any component reading the queue once the visibility timeout ends. This allows you to have many components all reading messages from the same queue, with each working to process different messages.
* DynamoDB, a secondary index is a data structure that contains a subset of attribute from a table, along with an alternate key to support Query operation.
* AWS SNS can select one of the following transports as part of the subscription request:

HTTP, HTTPS, Email, Email-JSON, SQS and SMS

* The xvd[f-p] is the recommended device name for EBS volumes that can be attached to the EC2 instance running on Window.
* It is possible to have one instance part of two separate ELBs. Through both ELBs have different configuration. ELBs are never launched in specific zones.
* Access Key ID and secret access key interact with AWS CLI/ AWS SDK or service specific APIs.
* The IAM users by default cannot change their password. The root user or IAM admin needs to set the policy in the password policy page, which should allow the user to change their password. Once it is enabled, the IAM user can always change their password from AWS console/CLI.
* SQS used for photo editing software.
* An **EBS volume** provides the persistent data storage. The **user can attach a volume to any instance provided they are both in the same AZ**. Even if they are in same region but in a different AZ. It will not able to attach the volume to the instance.
* If EBS volume is not in the same AZ of an EC2 instance, it cannot be attached to the instance. The only option is to take a snapshot of the volume and create a new volume in the instance’s AZ.
* DynamoDB integrate with IAM, you can use IAM to grant access to DynamoDB resources and API actions. To do so, you first write an IAM policy, which is a document the explicitly lists the permission you want to grant. You then attached the policy to AWS user or role.
* SNS makes it simple and cost effective to push to mobile devices, such as IPhone, iPad, Android, Kindle Fire and internet connected smart device, as well as pushing to other distributed services.
* Perform penetration testing as performed by attackers to find any vulnerabilities. He organization must take an approval from AWS before performing penetration testing perform hardening testing to find if there are any unnecessary ports open perform SQL injection to find any DB security issue.

~ The code memory checks are generally useful when the organization wants to improve the application performance.

* A user can share an AMI with another user/peer using the command.

ec2-modify-image-attribute <AMI-Id> -l -a <AWS-Account-id>

* When a user is trying to mount a blank EBS volume, it is required that user first create a file system with in volume. If the volume created from existing snapshot then user need not to create filesystem on the volume as it will wipe out the existing data.
* It is recommended rule that the root user should grant the least privileges to IAM user or the group. The higher the privileges, the more problem it can create.
* SQS provides “at least once” delivery of all the messages is its queue. You should design your system so that processing a message more than once doesn’t create any error or inconsistencies.
* With regards to IAM, when a request is made, the AWS services decides whether a given request should be allowed or denied, he evaluation logic follow these rules.

~ By default, all the request is denied (In general, request made using the account credential for resources in account are always allowed).

~ An explicit allow policy overrides this default.

~ An Explicit deny policy overrides any allows.

* The only recommendation use case for the S3 bucket ACL is to grant the write permission to the S3 log delivery group to write access log object to the user’s bucket.
* The Multi AZ feature allows the user to achieve High availability. MS SQL doesn’t support multi AZ.
* AWS RRS provides the same functionalities as S3, but at the cheaper rate. It is ideally suited for non-mission critical application. It provides less disabilities than S3, but is a cheaper option.
* An ELB performs a health check on its instances to ensure that it diverts traffic only to healthy instances. The ELB can perform a health check on HTTP, HTTPS, TCP and SSL protocols.
* It is possible to create an S3 bucket accessible only by a certain IAM user, using policies in a CloudFormation template. You can manage that AWS services and resources are available to each user or specific user.
* The IAM group policy is always aggregated, if user does not have permission for one group but has permission for another group, he will have full access to EC2. unless there is specific deny policy, the user will be able to access EC2.
* The AWS RDS DB instance is an isolated DB environment provided by AWS in which the user can create more than 1 database. The maximum size of the instance should be between 5GB to 3 TB. The size of each DB can be anything in this range.
* If the user needs to connect to RDS then he has to open 3306 in the RDS security group for his IP address.
* A user can create always a new EBS volume of higher size than the original snapshot size. The user cannot create the volume of lower size. When the new volume is created the size in the instance will be shown as the original size. The user needs to change the size of the device with resize2fs or other OS specific commands.
* Identity federation with SSO enables users from existing directory to access recourses within your AWS account, making it easier to manage your user by maintaining their identities in a single place. In this case, the federated user is only solution since AWS doesn’t allow creating more than 5000 IAM user.
* The AWS ELB allows mapping a custom domain name with ELB. The user can map ELB with DNS in 2 ways.

1. By creating CNAME with the existing domain name service provider, or
2. By Creating a record with Route 53

* AWS CloudFormation can be used to bootstrap both the Chef server and Chef Client software on EC2 instances.
* Amazon SWF an activity worker is a program that receives activities tasks, perform them, and provides result back. Which translate to a piece of software that implements tasks.
* Auto scaling attempts to distribute evenly the AZ that are enabled for the user’s auto scaling group. Auto scaling does this by attempting to launch new instances in the AZ with the fewest instances.
* Authentication mechanism are provided to ensure that messages stored in SQS queue are secured against unauthorized access. Only the AWS account owner can access the queue they create. SQS uses proven cryptographic methods to authenticate your identity. Either through the use of your access key Id and request signature, or through the use of X.509 certificate.
* Elastic beanstalk leverages AWS services such as EC2, S3, SNS, ELB and auto scaling to deliver the same reliable, scalable and cost-effective infrastructure that hundreds of business depend on today.
* In AWS Elastic Beanstalk, if the application returns response other than 200, OK or there no response within the configured inactivityTimeout period, SQS once again makes the message visible in the queue and available for another attempt at processing.
* The X-Forwarded-Port request header helps the user identify the port used by the client while sending a request to ELB.
* When designing an Amazon SWF workflow, you precisely define each of the required activities. You then register each activity with Amazon SWF as an activity type. when you register the activity, you provide information such as a **name** and **version**, and some **timeout values** based on how long you expect the activity to take.
* If a user has launched an EBS backed instance, the user will be charged for the EBS volume even through the instance is in a stopped state. The instance will be charged for the EC2 hourly cost when it is running.
* When a user trying to create a policy from AWS console, it will have options such as create policy from template or use a policy generator. The user can also define a custom policy or choose the option to have no permission. The policy simulator is not available in the console.
* AWS S3 follow the eventual consistent model. Once the object is updated it may return a new value or the old value based on whether all the content is replicated across multiple servers until it becomes consistent (eventual).
* AWS Elastic beanstalk will change the health status of web server environment tier to grey color when your application’s health status is unknown (because status is reported when the application is not in the ready state).
* If the user is going to specify an IP range in RDS security group, AWS recommends using the private IP address of the EC2 instance. This provides a more direct network route from the EC2 instance to the RDS instance, and doesn’t incur network charge for the data send outside the Amazon network.
* The EBS snapshot are a point in time backup of EBS volume. It is an incremental snapshot, but is always specific to the region and never specific to a single AZ. Hence the statement “It is stored in the same AZ as the volume” is incorrect.

~ When creating an EBS the user can not specify the subnet or VPC. However, the user must create the EBS in the same zone as the instance so that it can attached the EBS volume to the running instance.

* AWS Elastic beanstalk is best suited for those groups who want to deploy and manage their application within minutes in the AWS cloud. As a bonus, you don’t even need experience with cloud computing to get started.
* If you delete a queue, you need to wait at least 60 seconds before creating a queue with the same name. Please note that when you delete a queue. The deletion process takes up to 60 seconds, request you send to a recently deleted queue might succeed during the 60 seconds period. For example, a SendMessage request might succeed , but after 60 seconds the queue and that message you send no longer exists.
* Tags help you to categorize your load balancer in different way, for example, by purpose, owner or environment. The following basic restriction apply to tags. The **maximum tags per resource is 50**. The maximum key length size is 128 Unicode characters and value 256 Unicode characters. Tags key value are case sensitive.
* S3 bucket can be in one the three state.

~ unversioned (default).

~ versioning-enabled.

~ versioning-suspended.

The bucket owner can configure the versioning state of the bucket. The versioning state applies to all (never some) of the object in that bucket. The first-time owner enables a bucket for versioning, objects in it are thereafter always versioned and given a unique version ID.

* To configure SQS message retention period, set the MessageRetentionPeriod attribute using the SetQueueAttribute method. This attribute is used to specify the number of seconds a message will be retained by SQS. Currently the default value for the message retention period is 4 days. Using the MessageRetentionPeriod attribute, the message retention period can be set anywhere from 1 minute to 14 days.
* A user can see the number of Autoscaling resources currently allowed for the AWS account by using the as-describe-account-limits command or by calling the DescribeAccountLimits action.
* IAM is free service. You can create as many IAM users or groups as desired free of cost.
* If the user is using server-side encryption feature, Amazon S3 encrypts the object data before saving it on disks in its data centers and decrypts it when the user downloads the objects. Thus, the user is free from the tasks of managing encryption keys, and related tools.
* If a user is running two instances in separate AZs, it will provide HA with ELB since ELB will automatically stop routing the traffic to unhealthy instances and send it to healthy instances only.
* Amazon DynamoDB supports incremental and decremental atomic operations.
* Amazon S3 offers access policy options broadly categorized as resource-based policies and user policies. Access policies, such as ACL and resources policy can be attached to the bucket. With the object the user can only have ACL and not an object policy. The user can also attach access policies to the IAM users in the account. These are called user policies.
* DynamoDB, an index cannot be modified once it is created.
* When a user is trying to mount a blank EBS volume, it is required that the user first creates a file system within the volume.
* With regards to RDS, the user can manage the configuration of DB engine by using the DB parameter group. A DB group contains engine configuration values that can be applied to one or more DB instances of the same instance type.
* If a user is configuring HTTPS on the front end and TCP on the back end, ELB will not allow saving these listeners and will respond back with the message. “Load balancer protocol is an application layer protocol, but instance protocol is not. Both the load balancer protocol and the instance protocol should be at the same layer, please fix it.”
* Comparing with on-premises or EC2 based MS-SQL, RDS provides an automated backup feature, PIOPS is available with both RDS and EBS, However HA is not available with MS SQL.
* The EC2 instances are registered with the load balancer using the IP address associated with the instance. When the instance is stopped and then started. The IP address with the instance changes. This prevents the load balancer from routing traffic to the restarted instance. When the user stops and then start registered EC2 instances, it is recommended that to de-register the stopped instance from load balancer, and then register the restarted instance. Failure to do so may prevent the load balancer from performing health check and routing the traffic to the restarted instance.
* *In order to enable encryption at rest using EC2 and Elastic lock store, you must configure encryption when creating the EBS volume*.
* 504 Error is a Gateway timeout error.

**ElasticCache**

Elastic cache: In memory cache in the clod.

~ Improve performance of web application, allowing you to retrieve information from fast-in-memory cache rather than slower disk base databases.

~ Sits between your application and the database:

1. E.g. an application frequently requesting specific product information for your best-selling products.

~ Takes the load off your database.

~ Good if your database is particularly read-heavy and the data is not changing frequently.

~ Improve performance for read heavy workloads. E.g. Social networking, gaming media sharing and Q&A portals.

* Frequently -accessed adapt is stored in memory for low latency access, improving the overall performance of your application.
* Also good for compute heavy load e.g. recommendation engines.
* Can be used to store results of I/O intensive database queries or output of compute intensive calculation.

2 Type of Elastic Cache

**Memcached**:

~ Widely adopted memory object caching system.

~ Multi-Threaded

~ NO Multi AZ capabilities.

**Redis**:

~ Open source **in-memory key-value store**.

~ Supports more complex data structure: sorted sets and lists.

~ Support Master/ Slave replication and Multi-AZ for cross AZ redundancy.

Caching Strategies:

* 2 strategies available Lazy Loading and Write Through

**Lazy Loading**: Loads the data into the cache only when necessary. If requested data is in the cache, Elasticache return the data to the application. If the data is not in cache or has expired, Elasticache return null. Your application then fetches the data from database and writes the data received into the cache so that it is available next time.

**Advanctages:**

~ Only requested data cached: Avoids filling up cache with useless data.

~ Node failure are not fatal a new empty node will just have a lot cache missed initially.

**Disadvantages**:

~ Cache miss penalty: Initial request Query to database writing of data to the cache.

~ Stale Data:

If data is only updated when there is cache miss, it can become stale. **Doesn’t automatically update if the data in the database changes.**

**Lazy Loading and TTL:**

* Lazy loading treats an expired key as cache miss and causes the application to retrieve the data from the database and subsequently write the data into the cache with a new TTL.
* Does not eliminate stale data – but helps to avoids it.

**Write Through:**

* Write through strategy writes data into the cache whenever there is change to database.
* Data is never stale.
* Write penalty: Each write involves a write to cache.
* Elasticache node failure means that data is kissing until added or updated in the database.
* Wasted resources if most of the data is never used.
* HTTP 3XX code means there has been redirection.

**Lambda**

~ Lambda scales out (not up) automatically.

~ Lambda function are independent 1 event = 1 function

~ Lambda function can trigger other Lambda function, 1 event can = x function if function trigger other function.

~ Architecture can get extremely complicated, AWS X-ray allow you to debug what is happening.

~ Lambda can do things globally, you can use it to back up S3 bucket to other S3 bucket etc.

----------

**API Gateway**

**~** API Gateway is fully managed Service that makes it easy for developer to publish, maintain, monitor and secure APIs at any scale. With a few clicks in th AWS Management console, you can create an API that acts as front door for application access data, business logic or functionality from your back-end services, such as application running on Amazon EC2 code running on AWS Lambda, or any web application.

~ API Gateways has caching capabilities to increate performance.

~ You can throttle API Gateways to prevent attacks.

~ you can log result to CloudWatch.

~ if you using JavaScript/AJAX that uses multiple domains with API Gateways, ensure that you have enabled CORS on API Gateways.

~CORS is enforced by client.

**Version Control with Lambda.**

**~ When you use versioning in AWS** Lambda, you can publish one or more version of Lambda function, as a result, you can work with different variation of you lambda function in your development workflows, such as development, beta and production.

~ Each Lambda function version has unique ARN. After you publish a version, it is immutable (that is can not change)

~ AWS Lambda maintains your latest function code in $LATEST version, when you update your function code, AWS Lambda replaces the code in the $LATEST version of the Lambda function.

**Qualified/UnQulaified ARNs**

~ You can refer this function using its Amazon Resource Name (ARN). There are two ARN` accoaciated with this initial version.

**Qualified ARN:** -> This function ARN with the version suffix.

arn:aws:lamda:aws-region:acc-id:function:helloworld:$LATEST

Unqualified ARN: The function ARN without the version suffix.

arn:aws:lamda:aws-region:acc-id:function:helloworld

Alias:

~ After initially creating a Lambda function (the $LATEST version), you can publish a version 1 of it. By creating a lambda named PROD that point to version 1, you can now use the PROD alias to invoke version 1 of the Lambda function.

~ Now you can update the code (The $LATEST) with all your improvements, and then publish another stable and improved version (version 2). you can promote version 2 to production by remapping the PROD alias so that it points to version 2, if you find something wrong, you can easily roll back the production version to version1 by remapping the PROD alias so that it points to version 1.

~

**Step function**

* Step functions allows you to visualize and test your serverless application. Step functions provides a graphical console to arrange and visualize the component of your application as a series of steps. This makes it simple to build and run multistep applications. Step functions automatically trigger and track each step, and retrieve when there is error, so your application executes in order and as expected, Step functions logs the state of each step, so when things do go wrong, you can diagnose and debug problems quickly.
* Great way to visualize your serverless application.
* Step functions automatically trigger and track each step.
* Step functions log the state of each step so if something goes wrong you can track what wrong and where.

**X-Ray**

* AWX X-Ray is a service that collects data about request that your application serves, and provides tools you can view, filter, and gain insight into that data to identify issues and opportunities for optimization. For any traced request to your application, you can see detailed information not only about the request and response. But also, about calls that your application makes to downstream AWS resources, micro services, database and HTTP web APIs.

**The X-Ray SDK provides:**

* Interceptor to add to your code to trace incoming HTTP request.
* Client handles 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.
* The X-Rays integrate with the following AWS services:

1. ELB
2. Lambda
3. API Gateways
4. EC2
5. Elastic Beanstalk

* X-Ray integrate following language

1. Java 2) Go 3) Node.js 4) Python 5) Ruby 6).Net

**Advanced API Gateways**

* You can use the API Gateway import API feature to import an API from and external definition file into API Gateway. Currently, the **import API feature supports Swagger v2.0 definition files**.
* With the import API, you can either create a new API by submitting a POST request that includes a swagger definition in the payload and endpoint configuration, or you can update an existing API by using a PUT request that contains a swagger definition in the payload. You can update an API by overwriting it with a new definition, or merge a definition with an existing API. You specify the options using a mode query parameter in the request URL.
* Import API’s using Swagger 2.0 definition files
* API Gateway can be throttled; Default limits are 10000 RPS or 5000 concurrently.
* You can Configure API Gateway as a SOAP services passthrough.

**KMS**

* AWS Key Management Service (KMS) is managed service that makes it easy for you create and control the encryption keys used to encrypt your data. AWS KMS is integrated with other AWS services including
  + - * + EBS
        + S3
        + Redshift
        + Elastic Transcoder
        + AWS WorkMail
        + RDS

And others to make it simple to encrypt your data with encryption keys that you manage.

* The Customer Master Key
  + CMK
    - Alias
    - Creation date
    - Description
    - Key state
    - Key material (either customer provided or AWS provided)
  + Can NEVER be exported.
* KMS uses multi-tenant hardware but HSM dedicated to you.

**Setup a customer Master key:**

* Create alias and description.
* choose material option
* Define key administrative permission
  + IAM users/role that can be administer (but not use) the key through the KMS API.
* Define key usage permission
  + IAM user/role that can use the key to encrypt and decrypt data.
* Key material option:
* Use KMS generated key material
* Your own key material.

**aws kms encrypt** --key-id YOURKEYIDHERE --plaintext fileb://secret.txt --output text --query CiphertextBlob | base64 --decode > encryptedsecret.txt

**aws kms decrypt** --ciphertext-blob fileb://encryptedsecret.txt --output text --query Plaintext | base64 --decode > decryptedsecret.txt

**aws kms re-encrypt** --destination-key-id YOURKEYIDHERE --ciphertext-blob fileb://encryptedsecret.txt | base64 > newencryption.txt

**aws kms enable-key-rotation** --key-id YOURKEYIDHERE

**Envelope Encryption:**

The customer master key:

* + Customer master key used to decrypt the data key (envelope key).
  + Envelope key is used to decrypt the data.

**Developer Theory**

**CI/CD**

* CI/CD are best practices for development and deployment.
* They enable frequent software changes to be applied whilst maintaining system and service stability.
* Companies like AWS, Netflix, Google and Facebook have pioneered this approach to release code, successfully applying thousands of changes per day.
* Continuous delivery is the development practice where merged changes are automatically built, and prepared for release into staging and eventually production environment.
* There is usually a manual decision process to initiate deployment of the new code.

**AWS CodeCommit** : Source control service.

**AWS CodeBuild**: compile source code, run tests and package code.

**AWS CodeDeploy**: Automated deployment to EC2, on premises systems and Lambda.

**AWS CodePipeline**: CI/CD workflow tool, fully automates the entire release process (build, test, deployment)

**CodeCommit:**

* **Fully manage** source control service that enables companies to host secure and highly scalable private git repositories.
* Git is an 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 updated from multiple sources.
  + Enables collaboration
* Your data is encrypted in transit and at rest.

**AWS Code Deploy:**

* Automatically scales with your infrastructure and integrates with various CI/CD tools e.g. Jenkins, GitHub, Atlassian, AWS CodePipeline as well as config management tools like Ansible, Puppet and Chef.

**Two deployment approaches available**

* In-Place (rolling update) and Blue-green

**In-Place Deployment:**

The application is stopped on each instance and the latest revision installed.

* The instance is out of service during this time and your capacity will be reduced.
* If the instance is behind a load balancer, you can configure the load balancer to stop sending request to the instances which are being upgraded.
* In-place is also known as rolling update.
* It can only be used for EC2 and on-premise systems- it is not supported for Lambda.
* Of you need to roll back your changes, the previous version of the application will need to be re-deployed.

**Blue-green deployment:**

* New instances are provisioned and the latest revision is installed on the new instances. Blue represents the active deployment, green is the new release.
* The new instances are registered with an Elastic Load Balancer, traffic is then routed to the new instances and original instance are eventually terminated.
* Advantages of Blue/Green deployment ate that the new instances can be created ahead of time and the code released to production by simply switching all traffic to the new servers.
* Switching back to the original environment is faster and more reliable and is just case routing the traffic back to original servers (as long as you haven’t already terminated them).

AWS Code deploy -Terminology

* **Deployment Group**: A set of EC2 instances or lambda functions to which a new revision of the software is to be deployed.
* **Deployment**: The process and components 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 action you want AWS CodeDeploy to execute.
* **Revision**: Everything needed to deploy the new version: AppSpec file, application files, executables, config files.
* **Application**: Unique identifiers for the application you want to deploy. To ensure the correct combination of revision, deployment configuration and deployment group are referenced during a deployment.

Code Deploy code :

#Install CodeDeploy agent on your EC2 instance:

sudo yum update

sudo yum install ruby

sudo yum install wget

cd /home/ec2-user

wget https://aws-codedeploy-eu-central-1.s3.amazonaws.com/latest/install

chmod +x ./install

sudo ./install auto

sudo service codedeploy-agent status

#Create your application.zip and load it into CodeDeploy:

aws deploy create-application --application-name mywebapp

aws deploy push --application-name mywebapp --s3-location s3://<MY\_BUCKET\_NAME>/webapp.zip --ignore-hidden-files

**zip -r file.zip .**

**AppSpec file -Lambda Deployment:**

The Appspec file is used to ***defined the parameters*** that will be used for a CodeDeploy deployment. The file structure depends on whether you ae deploying to Lambda or EC2 /On premises

For Lambda deployment, the Appspec file may be written in YAML or JSON and contains the following fields.

**Version**: Reserved for future use- currently the only allowed value is 0.0

os: The operating system version you are using e.g. Linux, windows.

**Resources**: The name and properties of the Lambda function to deploy.

**files**: The location of any application files the need to be copied and where they should be copied to (source and destination folder)

**Hooks**: Specifies Lambda function to run at set points in the deployment lifecycle to validate the deployment e.g. validation tests to run before allowing traffic to be send to your newly deployed instance.

Lifecycle events hooks allow you to specify scripts that need to run at set points in the deployment lifecycle (e.g. unzip application files prior to deployment, run functional tests on the newly deployed application, and to de-register and re-register instances with a load balancer.



**BeforeAllowTraffic**: Used to specify the tasks or functions you want to run before traffic is routed to the newly deployed Lambda function (e.g. test to validate that the function has been deployed correctly)

**AfterAllowTraffic**: Used to specify the tasks or functions you want to run the traffic has been routed to the newly deployed Lambda function (e.g. test to validate that the function is accepting traffic correctly and having as expected.)

AppSpec File-EC2 and On Premises:

For EC2 and on premises deployment, the appspec.yml must be placed in the root of the directory of your revision- this is directory containing your application source code, otherwise deployment will fail.

Typical setup looks like this:

Mywebapp folder:

* + - Appspec.yml
    - /Scripts
    - /Config
    - /Source

**Appspec.yml file -EC2 Example**

version: 0.0

os: linux

Files:

-source: Config/config.txt

destination: /webapps/Config

-source: Source

destination: /webapps/myApp

**appspec.yml File -EC2 Example**

BeforeInstall:

-location: Script/UnzipResourceBundle.sh

-location: Script/UnzipDataBundle.sh

AfterInstall:

-location: Script/RunResourceTests.sh

timeout: 180

ApplicationStart:

-location: Scripts/RnFuctionalTests.sh

timeout: 3600

ValidateService:

-location: Script/MoniterService.sh

timeout:3600

runas: codedeployuser

**AppSpec File: Supported Hooks for EC2 and on premises**

**BeforeBlockTraffic**: Run tasks on instance before they are deregistered from load balancer.

**BlockTraffic**: Deregister instances from load balancer

**AfterBlockTraffic**: Run tasks on instances after they are Deregistered from load balancer.

**ApplicationStop**: Gracefully stop the application in preparation for deploying the new revision.

**DownloadBundle**: The CodeDeploy agent copies the application revision files to temporary location.

**BeforeInstall**: Details of any pre-installation scripts, e.g. backing up the current version, descripting files.

**Install**: The CodeDeploy agent copes the application revision files from their temporary location to their correct location.

**AfterInstall**: Details of any post-installation scripts e.g. configuration task, change the file permission.

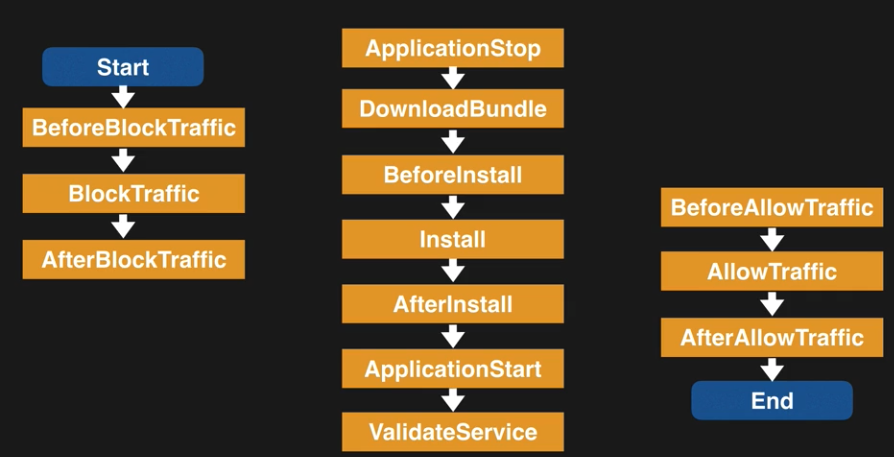
**AppicationStart**: Restart any services that were stopped during ApplicationStop.

**ValidateService**: Details of any tests to validate the service.

**BeforeAllowTraffic**: Run tasks on instances before they are register with a load balancer

**AllowTraffic**: Register instances with the load balancer

**AfterAllowTraffic**: Run tasks on instances after they are registered with the load balancer.



**Docker:**

Docker is open source technology which allow you to create application based on either Linux or Window containers.

* A container is lightweight standalone executable software packages which includes everything the software needs to run-code, runtime environment, libraries, environment settings etc.
* AWS provides **Elastic Container Service** as fully managed clustered platform which allow you to run your Docker images in the cloud.
* **AWS CodeBuild** is fully managed build service which runs a set of command that you define, e.g. complies code runs tests and produces artifacts that are ready to deploy.

##### Retrieve the login command to use to authenticate your Docker client to your registry.

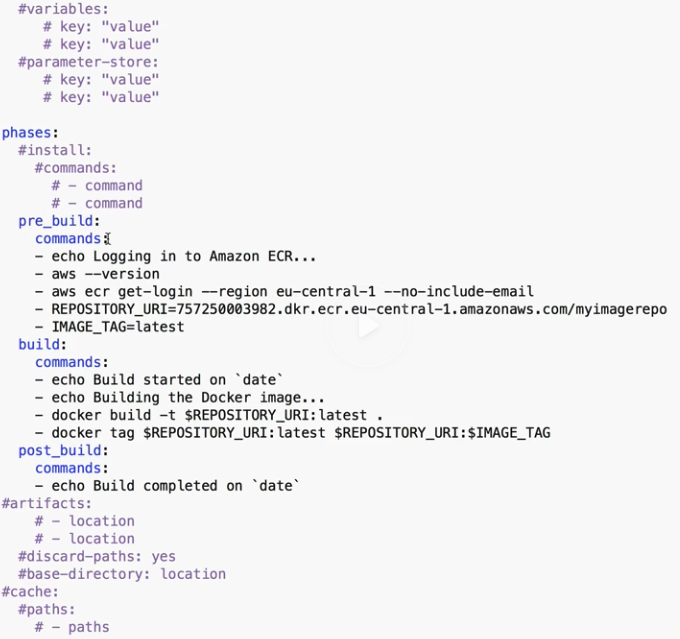
Use the AWS CLI:

* $(aws ecr get-login --no-include-email --region us-east-1)
* docker build -t myrepo .
* docker tag myrepo:latest 418185438171.dkr.ecr.us-east-1.amazonaws.com/myrepo:latest
* docker push 418185438171.dkr.ecr.us-east-1.amazonaws.com/myrepo:latest

**buildspec.yml.**

Use buildspec.yml to defined the build command and settings used by CodeBuild to run your build.

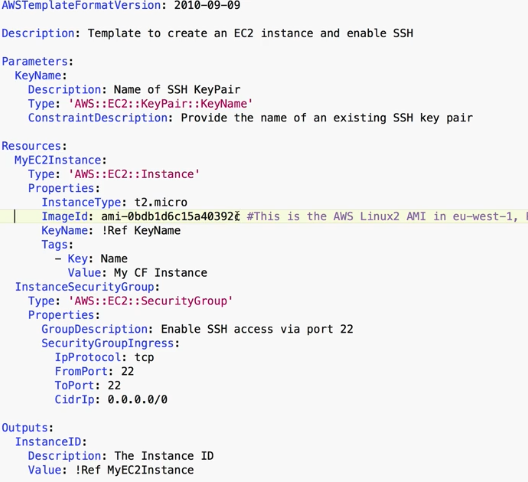
You can override the setting in buildspec.yml by adding your own command in the console when you launch the build



If your build fails, check the build logs in the CodeBuild console and you can also view the CodeBuild log in CloudWatch.

**CloudFormation**

* CloudFormation is a service that allow you to manage configure and provision your AWS infrastructure as code.
* Resources are defined using the CloudFormation template
* CloudFormation interprets the template and make appropriate API calls to create the resources you have defined.
* Support YAML and JSON.
* Infrastructure is provisioned consistently, with fewer mistakes.
* Less time and effort than configuring thing manually.
* You can version control and peer review your template
* Free to use
* Can be used manage updated & dependencies
* Can be used to rollback and delete the entire stack as well.
* Transform section (CFN) is used to reference additional code stored in S3, allowing for code re-use. E.g. for Lambda code or template snippets/ reusable pieces of CloudFormation code.



**Serverless Application Model (SAM)**

* SAM is an extension to CloudFormation used to define serverless application.
* Simplified syntax for defining serverless resources: APIs, Lambda Functions, DynamoDB tables etc.
* Use the SAM CLI to package your deployment code, upload it to S3 and deploy your serverless application

sam package \

--template-file ./lambda.yml \

--output-template-file sam-template.yml \

--s3-bucket cfsambucket

sam deploy \

--template-file sam-template.yml \

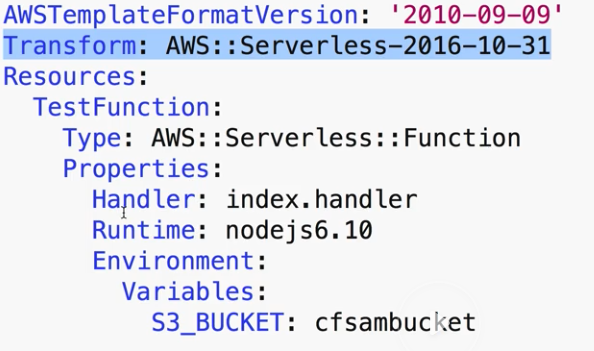
--stack-name mystack \

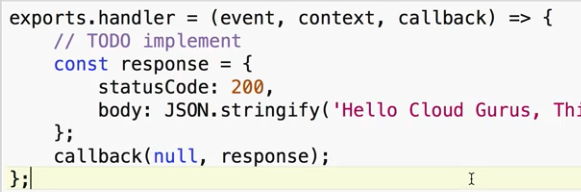
--capabilities CAPABILITY\_IAM

User via CLI : aws iam get-user

Create S3 bucket: aws s3 mb s3://cfsambucket –region eu-west-1

lambda.yml





**CloudFormation Nested Stacks:**

Nested Stacks allow re-use of CloudFormation code for common use cases.

* E.g. standard configuration for a load balancer, web server, application server etc.
* Instead of copying out the code each time, create a standard template for each command use case and reference from within your CloudFormation template.
* Simply create a CloudFormation template, store in S3 and you can reference it in the Resources section of any CloudFormation template using the Stack resource type.

**DynamoDB**

* Amazon DynamoDB is fast and flexible NoSQL database service for all application that need consistent, single-digit millisecond latency at any scale. It is a fully managed database and support both document and key-value data models. Its flexible data model and reliable performance make it a great fir for mobile, web, gaming, ad-tech, IoT, and many other applications.
* This is serverless.
* Stored in SSD storage
* Spread across 3 geographical distinct data centers.
* Choice of 2 consistency models.
  + Eventual Consistency Read (Default)
  + Strongly Consistent Read

**Eventual Consistency Read**:

* Consistency across all copies of data is usually reached within a second. Repeating a read after a short time should return the updated data (best Read Performance)

**Strongly Consistent Read**:

* A Strongly consistent read returns a result that reflects all writes that received a successful response prior to the read.

**Primary key:**

* DynamoDB stored and retrieves data based on the primary key
* 2 types of primary
* **Partition Key**: unique attribute (e.g. user ID)
* Value of partition key is input to an internal hash function which determines the partitioned of physical location on which the data is stored.
* If you are using partition key as your primary key, then no two items can have the same partition key.
* **Composite key (Partition Key + Sort Key)** in combination
  + e.g. Same user posting multiple time to forum
  + primary key would be a composite key consisting of
  + Partition key – User ID
  + Sort key – timestamp of the post
  + 2 items may have the same Partition key, but they must have a different sort Key.
  + All items with the same Partition key are stored together, then sorted according to the sort key value.
  + Allow you to store multiple items with the same Partition key.

**DynamoDB access Control:**

* Authentication and access control are managed using AWS IAM.
* You can create an IAM user within your AWS account which has specific permission to access and create DynamoDB tables.
* You can create an IAM role which enables you to obtain temporary access keys which can be used to access DynamoDB.
* You can also use a special IAM Condition to restrict user access to only their own records.

**DynamoDB- IAM Condition example:**

* Imagine a mobile gaming application with millions of users.
* User need to access the high scores for each game they are playing.
* Access must be restricted to ensure they cannot view anyone else data

|  |  |  |  |
| --- | --- | --- | --- |
|  | User ID | Game Title | Top score |
| Only this access | Ram | Cricket | 101 |
| X | Mohan | - | - |

* This can do by adding a condition to an IAM Policy to allow access only to item where the partition key value matches their User ID.



**Query DynamoDB from CLI:**

* aws dynamodb get-item –table-name ProductCatalog --key ‘{“Id”: {“N”:”100”}}’
* You need IAM access to allow dynamodb table access to user.

**What is an Index?**

In SQL databases, an index is a data structure which allows to you perform fast queries on specific columns in a table. You select the column that you want included in the index and run your searches on the index – rather than on the entire dataset.

In DynamoDB, 2 types of index are supported to help speed-up your DynamoDB queries:

* Local Secondary index.
* Global Secondary Index.

**Local Secondary Index:**

* Can only be created when you are creating your table.
* You cannot add, remove, or modify it later.
* It has same partition key as your original table.
* But a different sort key
* Gives you a different view of your data, organized according to an alternative Soft Key
* Any queries based on this Sort key are much faster using the index than the main table.
* E.g. Partition key: user ID

Sort Key: account creation date

**Global Secondary index:**

* You can create when you create your table, or add it later
* Different Partition key as well as different sort key
* So, gives a completely different view of the data
* Speeds up any queries relating to this alternative partition and sort key
* E.g. Partition key: email address

Sort key: last log-in data

**DynamoDB indexes: Exam Tips**:

* Indexes enables fast queries on specific data columns
* Gives you different view of your data, based on alternative Partition/Sort Keys.
* Important to understand the differences.

|  |  |
| --- | --- |
| **Local Secondary Index** | **Global Secondary Index** |
| Must be created at when you create your table | Can be created any time – at table creation or After |
| Same partition keys as your table | Different Partition key |
| Different Soft key | Different Sort key |

**Scan vs Query API call**

**Query:**

* A Query operation finds items in a table based on the Primary key attribute and a distinct value to search for.
* E.g. select an item where the user ID is equal to 212, will select all the attribute for that item, e.g. first name, surname, email etc.
* Use an optional sort key name and value to refine the result.
* E.g. if your sort key is a timestamp, you can refine the query to only select items with a timestamp of the last 7 days.
* By default, a query returns all the attributes for the items but you can use the ProjectionExpression parameters if you want the query to only return the specific attributes you want.
* E.g. if you only want to see the email address rather than all the attributes.
* Result are always sorted by sort key.
* Numeric Order – by default in ascending order (1,2,3,4)
* ASCII character code values
* You can reserve the order by setting the SacnIndexForward parameter to false.
* By default, Queries are eventually consistent
* You need to explicitly set the query to be Strongly consistent.

What is a scan?

* A Scan operation examines every item in the table.
* By default, return all data attributes
* Use the **ProjectionExpression** parameter to refine the scan to only return the attribute you want.
* Applying filter.

Query vs Scan?

* Query is more efficient than a Scan.
* Scan dumps the entire table, then filters out the values to provide the desired result – removing the unwanted data
* This adds an extra step of removing the data you don’t want
* As the table grows, the scan operation takes longer.
* Scan operation on a large table can use up the provisioned throughput for a large table in just a single operation.

**How to improve Performance:**

* You can reduce the impact of query or scan by setting a smaller page size which uses fewer read operation.
* E.g. set the page size to return 40 items
* Larger number of smaller operations will allow other requests to succeed without throttling
* Avoid using scan operations if you can: design tables in a way that can use the query, get, or BatchGetItem APIs.

**How to Improve Scan Performance:**

* By default, a scan operation process data sequentially in returning 1MB increment before moving on to retrieve the next 1MB of data, it can be only one partition at a time.
* You can configure DynamoDB to use parallel scans instead by logically dividing a table or index into segments and scanning each segment in parallel.
* Bet to avoid parallel scans if your table or index is already incurring heavy read / write activity from other application.

**DynamoDB Provisioned Throughput**

* DynamoDB provisioned Throughput is measured in Capacity Unit.
* When you create your table, you specify your requirement in terms of read Capacity Unit and Write capacity Unit.
* 1 x write Capacity Unit = 1 x 1KB write per second
* 1 x Read Capacity Unit = 1 x Strongly consistent Read of 4KB per second.

Or

2 x Eventual Consistency read of 4 KB per second.

Example:

* Table with 5 x Read Capacity unit and 5 x write capacity units.
* This configuration will be able to perform:
* 5 x 4KB Strongly consistent reads = 20KB per second
* Twice as many Eventual Consistent = 40KB
* 55 x 1 KB writes = 5 KB per second
* If your application reads or writes larger items it will consume more capacity units and will cost you more as well.

**Strongly consistent reads Calculation:**

* Your application needs to read 80 items (table rows) per second.
* Each item 3KB in size
* You need strongly consistent reads.
  + First calculate how many read capacity units needed for read:

size of each item / 4KB

3KB/4KB = 0.75

* + Rounded-up the nearest whole number, each read will need 1x read capacity unit per read operation.
  + Multiply by the number of read per second = 80 Read Capacity Unit required.
* **Eventual Consistent read:**
  + What if you need Eventual Consistent Reads?
  + You do the same calculation. however, as this is for Eventual consistent reads and you get 2x 4KM reads per second-

Or double the throughput of strongly consistent read.

* + Size of each item /4KB

-3KB/4KB = 0.75 round-up to the nearest whole number, = 1

- multiply by the number of reads per second = 80

* + Divide 80 by 2, so you need only 40 Read capacity unit

**Write Capacity Unit Calculation:**

* You want to write 100 items per second.
* Each item 512 bytes in size.
* First, Calculation how many Capacity Unites for each write:

Size of each item / 1 KB (for write capacity unit)

512 bytes / 1 KB = 0.5

* Rounded-up to the nearest whole number, each write will need 1 x write capacity unit per write operation.
* Multiplied by the number of writes per second = 100 write capacity unit required.

**DynamoDB on-demand capacity Option**

* Charges apply for. Reading, writing and strong data
* With On-demand, you don’t need to specify your requirements.
* DynamoDB instantly scales up and down based on the activity your application.
* Great for unpredictable workloads.
* You want to pay for only what you use. (pay per request).

|  |  |
| --- | --- |
| **On-demand** | **Provisioned Capacity** |
| Unknown workloads | You can forecast read and write capacity requirement. |
| Unpredictable application traffic | Predictable application traffic. |
| You want a pay-per-use modal | Application traffic is consistent or increased gradually. |
| Spiky, short lived peaks |  |

**DynamoDB Accelerator (DAX)**

* **Provides in-memory caching for DynamoDB tables.**
* DynamoDB Accelerator (DAX) is fully managed, clustered in-memory cache for DynamoDB
* Deliver up to a 10-x read performance improvement.
* Microsecond performance for millions of requests per second
* Ideal for read-heavy and brushy workloads
* E.g. auction application, gaming and retail sites during black Friday promotion.

**How Does it work?**

* DAX is write-through caching service- this means data is written in the cache as well as back end store at the same time.
* Allow you point your DynamoDB API call at the DAX cluster.
* If the item you are querying is in the cache (cache hit), DAX return the result to the application.
* If the item is not available (cache miss) then DAX performs an eventual consistent GetItem operation against DynamoDB.
* Retrieval of data from DAX reduces the read load on the DynamoDB tables.
* May be able to reduce

Provisioned Read Capacity

**What is NOT suitable for?**

* Caters for eventual consistent read only – so not suitable for application that require strongly consistent read.
* Write intensive application
* Application that do not perform many read operations.
* Application that do not require microsecond response times

**ElasticCache DynamoDB**

* In-memory cache sits between your application and dataset.
* 2 different caching strategies: Lazy Loading and write Through
* Lazy loading only caches the data when it is required
* ElasticCache nodes failure not fatal, Just lots of cache misses.
* Cache miss penalty: initial request, query database, writing to cache.
* Avoid stale data by implementing a TTL.
* Data is never stale
* Write Penalty: Each write involves a write to the cache.
* ElasticCache node failure means that data is missing until added or updated in the database.
* Wasted resources if most of the data is never used.
* Write Though strategy writes data into the cache whenever there is a change to
* Good if your database is particularly read heavy and the data is not changing frequently.
* Also good for compute heavy workloads e.g. recommendation engine

2 type:

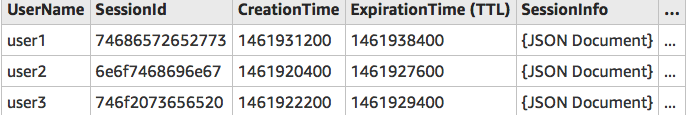
* + Memcached
  + Redis

**DynamoDB Transaction(2018)**

* ACID transaction (Atomic, Consistent, isolated, durable).
* Read or Write multiple item across multiple table as an all or nothing operation.
* Check for a pre-requisite condition before writing to a table.

**DynamoDB TTL**

* Time to Live attribute defines an expiry time of your data
* Expired item marked for deletion
* Great for removing irrelevant or old data:
  + - Session data
    - Event logs
    - Temporary data
* Reduces cost by automatically removing data which is no longer relevant.
* When the current time is greater than TTL, the item will be expired and maked for deletion
* You can filter out expired items from your query.
* TTL expressed as epoch time.



1) Check your IAM user permissions:

aws iam get-user

2) Create SessionData table:

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

3) Populate SessionData Table:

aws dynamodb batch-write-item --request-items file://items.json

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"}

}

}

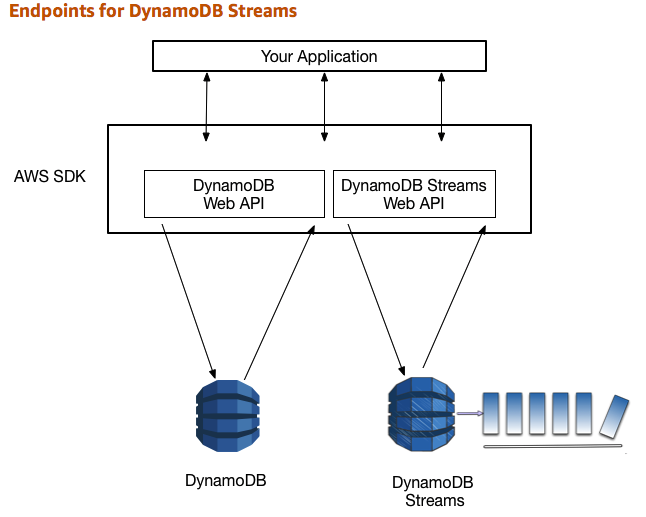
}

]

}

**DynamoDB Streams**

* Time-order sequence of item level modification (insert, update, delete)
* Logs are encrypted at rest and stored for 24 hours.
* Accessed using a dedicated endpoint
* By default, the primary key is recorded
* Before and after images can be captured.
* Can be used as an event source for Lambda so you can create application which take actions based on evets in your DynamoDB table.



**Processing DynamoDB streams:**

* Events are recorded in near real-time.
* Application can take actions based on contents.
* Event source for Lambda.
* Lambda polls the DynamoDB stream.
* Execute Lambda code based on a DynamoDB streams event.

**Provisioned ThroughPut Exceeded Exception**

* ProvisionedThroughputExceededException
* Your request rate is too high for the read / write capacity provisioned on your DynamoDB. table
* SDK will automatically retry the requests until successful.
* If you are not using the SDK you can:
  + Reduce request frequency
  + Use **Exponential Backoff**.

**What is Exponential Backoff?**

* Many components in a network can generate error due to being overloaded.
* In addition to simple retries all AWS SDKs use Exponential Backoff.
* Progressively longer waits between consecutive retries e.g. 50ms, 100 ms .. for improved flow control.
* If after 1 minute this doesn’t work, you request size may be exceeding the throughput for your read/write capacity.

Exam Tips:

* If you see a ProvisionedThroughputExceededException error, this means the number of requests is too high.
* **Exponential Backoff improves flow by retrying requests using progressively longer waits.**
* This is not just true for DynamoDB, **Exponential Backoff** is a feature for every SDK and applies to many services within AWS. E.g. S3 bucket, CloudFormation, SES.

**CloudWatch**

* Amazon CloudWatch is a **monitoring service** to monitor your AWS resources, as well as the applications that you run on AWS.
* What can CloudWatch do?
  + Compute
    - Autoscaling Group
    - Elastic Load Balancer
    - Route53 health check
  + Storage & Content Delivery
    - EBS Volume
    - Storage Gateways
    - CloudFront
* CloudWatch can monter thinks like
  + Database & Analytics
  + DynamoDB
  + Elasticache Nodes
  + RDS instances
  + Elastic MapReduce Job Flows
  + Redshift
* Other

SNS Topics

SQS Queue

Opsworks

CloudWatch log

Estimated Charge on your AWS bill

**Host Level Metrics consist of EC2:**

* CPU
* Network
* Disk
* Status Check

Exam Tip: **RAM utilization is a custom metric**! by default EC2 monitoring is 5-minute interval, unless you enable detailed monitoring which will then make it 1-minute intervals.

**How long are CloudWatch Metrics stored?**

* You can retrieve data using the GetMetricStatistics API or by using third party tools offered by AWS Partners
* You can store your log data in CloudWatch logs for as long as you want. By default, CloudWatch logs will store your data indefinitely. You can change the retention for each Log Group at any time
* You Can retrieve data from terminated EC2 or ELB instance after its termination.

**Metric Granularity?**

It depends on the AWS service. Many default metrics for many default services are 1 minute. But it can be 3 or 5 minutes depending on the service.

* For custom metrics the minimum granularities that you can have is 1 minute.

**CloudWatch Alarms:**

* You can create an alarm to monitor any Amazon CloudWatch metric in your account. This can include EC2 Utilization, Elastic Load Balancer Latency or even the charges on your AWS bill. You can set the Appropriate threshold in which to trigger the alarms and also set what action should be taken if an alarm state is reached. This will be covered in a subsequent lecture.

CloudWatch can be used on premise: Not restricted to just AWS resources. Can be on premise too. Just need to download and install the SSM agent and CloudWatch agent.

**#!/bin/bash**

**yum update -y**

**sudo yum install -y perl-Switch perl-DateTime perl-Sys-Syslog perl-LWP-Protocol-https perl-Digest-SHA.x86\_64**

**cd /home/ec2-user/**

**curl https://aws-cloudwatch.s3.amazonaws.com/downloads/CloudWatchMonitoringScripts-1.2.2.zip -O**

**unzip CloudWatchMonitoringScripts-1.2.2.zip**

**rm -rf CloudWatchMonitoringScripts-1.2.2.zip**

**# Use the commands below for the lab.**

**# /home/ec2-user/aws-scripts-mon/mon-put-instance-data.pl --mem-util --verify --verbose**

**# /home/ec2-user/aws-scripts-mon/mon-put-instance-data.pl --mem-util --mem-used --mem-avail**

**# \*/1 \* \* \* \* root /home/ec2-user/aws-scripts-mon/mon-put-instance-data.pl --mem-util --mem-used --mem-avail**

**CloudWatch vs CloudTrail vs Config**

* Cloud Monitor performance
* CloudTrail monitor API calls in the AWS Platform
* AWS Config records the state of your AWS environment and can notify you of changes.

**Lambda**

**What is Lambda?**

* Data Centers
* Hardware
* Assembly code /Protocol
* High Level Language
* Operating System
* Application Layer/AWS API
* AWS Lambda
* AWS Lambda is a compute service where you can upload your code and create a Lambda function. AWS Lambda takes care of provisioning and managing the servers that you use to run the code. You don’t have to worry about operating systems, patching, scaling etc. You can use Lambda in the following way.
  + As an **event driven compute service** where AWS Lambda runs your code in response of events. These events could be **changes to data in an S3 bucket or an Amazon DynamoDB table.**
  + As a compute service to run your code in response to HTTP request using Amazon API gateways or API calls made using AWS SDKs. This is what we use at A Cloud Guru.

**API Gateways**

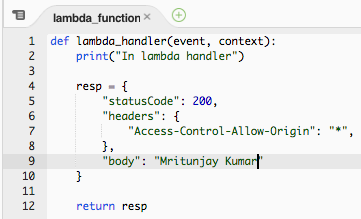
* An API is an Application Programming interface

Types of API:

* + REST API
  + SOAP API

What Can API gateway Do?

* Expose HTTPS endpoints to defined a Restful API.
* Serverless connect to service like Lambda and DynamoDB.
* Send each API endpoint to a different target.
* Scale effortlessly.
* Track and control use by API key.
* Throttle request to prevent attacks.
* Now supports AWS Certificate Manage: free SSL/TLS certs.
* When you enable caching API caches Reponses from your endpoint for a specified time-to-live (TTL), in second.
* You need to enable CROS.



Using Polly to help you pass your exam: A Serverless Approach

**Advanced API Gateways**

**Import APIs.**

You can import **API Gateway import** API feature to import an API from an external definition file into API Gateway. Currently, the import supports **Swagger v2.0** definition files.

With the import API, you can either create a new API by submitting a POST request that includes a Swagger definition in the payload and endpoint configuration, or you can update an existing API by using a PUT request that contains a Swagger definition in the payload. You can update an API by overwriting it with a new definition, or merge a definition with an existing API. You specify that options using a mode query parameter in the request URL.

API Throttling

* By default, API gateway limits the steady-state request rate to 10000 request per second (rps).
* This maximum concurrent request is 5000 requests across all APIs within an AWS account.
* If you go over 10000 request per second or 5000 concurrent requests you will receive a

**429 Too Many Request** error response.

* You can configure API gateways as a SOAP web service passthrough.

**EBS:**

[root@ip-172-31-37-6 ec2-user]# **lsblk**

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

xvda 202:0 0 8G 0 disk

`-xvda1 202:1 0 8G 0 part /

xvdf 202:80 0 100G 0 disk

[root@ip-172-31-37-6 ec2-user]# **file -s /dev/xvdf**

/dev/xvdf: data

[root@ip-172-31-37-6 ec2-user]# **mkfs -t ext4 /dev/xvdf**

mke2fs 1.42.9 (28-Dec-2013)

Filesystem label=

OS type: Linux

Block size=4096 (log=2)

Fragment size=4096 (log=2)

Stride=0 blocks, Stripe width=0 blocks

6553600 inodes, 26214400 blocks

1310720 blocks (5.00%) reserved for the super user

First data block=0

Maximum filesystem blocks=2174746624

800 block groups

32768 blocks per group, 32768 fragments per group

8192 inodes per group

Superblock backups stored on blocks:

32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,

4096000, 7962624, 11239424, 20480000, 23887872

Allocating group tables: done

Writing inode tables: done

Creating journal (32768 blocks): done

Writing superblocks and filesystem accounting information: done

[root@ip-172-31-37-6 ec2-user]# **file -s /dev/xvdf**

/dev/xvdf: Linux rev 1.0 ext4 filesystem data, UUID=2c181cb5-7635-48da-bc02-4949e5b0aa95 (extents) (64bit) (large files) (huge files)

[root@ip-172-31-37-6 ec2-user]# lsblk

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

xvda 202:0 0 8G 0 disk

`-xvda1 202:1 0 8G 0 part /

xvdf 202:80 0 100G 0 disk

[root@ip-172-31-37-6 ec2-user]# cd /

[root@ip-172-31-37-6 /]# mkdir filesystem

[root@ip-172-31-37-6 /]# mount /dev/xvdf /filesystem

[root@ip-172-31-37-6 /]# lsblk

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

xvda 202:0 0 8G 0 disk

`-xvda1 202:1 0 8G 0 part /

xvdf 202:80 0 100G 0 disk /filesystem

Unmount: [root@ip-172-31-37-6 /]# umount -d /dev/xvdf

* You can encrypt the root device volume (the volume the OS is installed on) using operating system level encryption.
* You can encrypt the root device by taking a snapshot of that volume, and then creating a copy of that snap with encryption. You can then make an AMI of this snap and deploy the encrypted root device volume.
* You can encrypt additional attached volume using the console, CLI or API.

**Elastic Beanstalk**

Elastic Beanstalk is service for deploying and scaling web application developed in many popular language: Java, .Net, PHP, Node.js, Python, Ruby, Go and Docker into widely used application server platform like Apache Tomcat, Nginx, Passenger and IIS.

* Developer can focus on writing code and don’t need to worry about any of underlying infrastructure needed to run the application.
* You upload the code and Elastic Beanstalk will handle deployment, capacity provisioning, load balancing, auto scaling and application health.
* You retain full control of underlying AWS resources and you pay only for the AWS resources required to store and run your applications. (e.g. EC2 instance and S3 bucket)
* Fastest and simple way to deploy.
* Automatically scale and up and down.
* You can select the Ec2 instance that are optimal for your Application
* You can retain full administrative control over the resources powering your application, or have Elastic Beanstalk for it for you.
* Monitor and manage application health via a dashboard.
* Integrated with CloudWatch and X-ray for performance data and the metrics.

Elastic Beanstalk supports server option for processing deployments.

1. **All at once**
   * Deploy the new version to all the instance.
   * All of our instance ae out of service while the deployment takes place.
   * You will experience an outage while the deployment is taking place-not ideal for mission critical production system.
   * If the update fails, you need to roll back the changes by re-deploying the original version to all the instances.
2. **Rolling**
   * Deploy the new version in batches.
   * Each batch of instances is taken out of service while the deployment takes place.
   * Your environment capacity will be reduced by number of instances in the batch while deployment takes place.
   * Not ideal for performance sensitive system
   * If the upgrade fails, you need to perform and additional rolling update to roll back the changes.
3. **Rolling with additional batch**
   * Launches and additional batch of instances.
   * Deploy the new version in batches.
   * Maintains full capacity during the deployment process.
   * If the update fails, you need to perform and additional rolling update to roll back the changes.
4. **Immutable**
   * deploy the new version of fresh group of instances in their own new autoscaling group.
   * When the new instances pass their checks, they are moved to your existing auto scaling: and finally, the old instances are terminated.
   * Maintain full capacity during the deployment process
   * The impact of failed update is far less and rollback process requires only terminating the new auto scaling group.
   * Prefer option for mission critical production system.

**Advanced Elastic Beanstalk**

You can customize your Elastic Beanstalk environment using Elastic Beanstalk configuration files (e.g. you can define packages to install, create Linux user and group, run shell commands, specify services to enable or configure your load balancer etc.)

* There are files written in YAML or JSON format, they can have a filename and your choice but must have .**config** extension and be saved a folder called .**ebextentions**.
* .**ebextention** folder must be the top level directory of your application source code bundle.
* This means that the configuration files can be placed under source control along with the rest of your application code.

**RDS & Elastic Beanstalk**

* Elastic Beanstalk supports Two way of integration an RDS database with your Elastic Beanstalk environment.
* You can launch the RDS instance from within the Elastic Beanstalk console, which means the RDS instance is created within console. Which means the RDS instance is created within your Elastic Beanstalk environment- a good option for Dev and test deployment.
* However, this may not be ideal for production environments because it means the lifecycle of your database is tied to the lifecycle of your application environment. If you terminate the environment, the database instance will be terminated too.
* For production environment, the proffered option is to d**ecouple the RDS instance from your EBS environment**: i.e. launch it outside of Elastic Beanstalk, directly from the RDS section of the console.
* This option gives you a lot more flexibility, allows you to connect multiple environment to the same database, provides a wider choice of database types, and allow you to tear down your application environment without affecting the database instance.

**Access to RDS from Elastic Beanstalk:**

* + To allow the EC2 instances in your Elastic Beanstalk environment to connect to an outside database, there are two additional configuration steps required.
    - An additional security Group must be added to your enlivenment’s auto scaling group.
    - You will need to provide connection string using configuration information to your application server (endpoint, password using Elastic Beanstalk environments properties).

**AWS Systems Manager Parameter Store:**

* You work for Bank as system admin. You need to store users, password, license key. This information needed to be passed to EC2 as bootstrap script. while maintain the confidentialities of the information.
* You can store values as plain text or you can encrypt the data.
* You can then reference these values by using their names.
* You can use this service with EC2, CloudFormation, lambda, EC2 run commands etc.

**SNS**:

* A variety of message formats are supported: SMS text message, email, SQS queue, any HTTP endpoint.
* Pub/Sub Model – consumers must subscribe to a topic.
* You can fan out messages to large number of recipients, (e.g. multiple client each with their own SQS queue)