JON BONSO AND CARLO ACEBEDO

AWS CERTIFIED DEVELOPER ASSOCIATE DVA-C02 EXAM

Tutorials Dojo Study Guide

TABLE OF CONTENTS

INTRODUCTION

The trend of emerging technologies that have a large impact on industries is a common theme in today's headlines. More and more companies are adopting newer technologies that will allow them to grow and increase returns. The Amazon Web Services (AWS) Cloud is one example of it. There are more than a million active users of Amazon Web Services. These users are a mix of small businesses, independent contractors, large enterprises, developers, students, and many more. There is no doubt that AWS already has a community of users, and the number of newcomers is increasing rapidly each day.

With AWS, you don't have to purchase and manage your infrastructure. It is a considerable cost saving for your company, and it is a sigh of relief as well for your sysadmin team. Although AWS offers a huge collection of services and products that require little configuration, they also offer traditional ones like VMs and storage devices that work similarly to their physical counterparts. This means that transitioning from an on-premises type of environment to an AWS virtualized environment should be straightforward.

We think that developers benefit the most from the cloud. You can access your applications from anywhere if you have a browser and an Internet connection (and maybe an SSH/RDP client too). You can spin up machines in a matter of minutes from a catalog of OS and versions that suit your needs. You also have control over your storage devices, which you can also provision and de-provision easily. Aside from traditional VMs, AWS also has other options for you, such as container instances, batch instances, and serverless models that can easily be integrated with APIs. You also do not need to worry about databases. You can host them on your own in VMs with your own licenses or use AWS-provided licenses, or you can even use a managed database, so you do not need to worry about infrastructure. If these are too much for a simple app that you just want to test, AWS has a platform-as-a-service solution wherein you can simply deploy your code, and the service handles the infrastructure for you. In essence, AWS has already provided the tools that its users need to quickly take advantage of the cloud.

With all of the benefits presented in using AWS, the true value of being an AWS Certified Developer has your knowledge and skills in developing in AWS recognized by the industry. With this recognition, you'll gain more opportunities for career growth, financial growth, and personal growth as well. Certified individuals can negotiate for higher salaries, aim for promotions, or land higher job positions. Becoming an AWS Certified Developer Associate is also a great way to start on your journey in DevOps, which is one of the most in-demand and well-compensated roles in the IT industry today. This certificate is a must-have in your portfolio if you wish to progress your career in AWS.

Note: We took extra care to come up with these study guides and cheat sheets. However, this is meant to be just a supplementary resource when preparing for the exam. We highly recommend working on **hands-on sessions** and **practice exams** to further expand your knowledge and improve your test-taking skills.

AWS CERTIFIED DEVELOPER ASSOCIATE EXAM OVERVIEW

Amazon Web Services (AWS) began the Global Certification Program with the primary purpose of validating the technical skills and knowledge for building secure and reliable cloud-based applications using the AWS platform. By successfully passing the AWS exam, individuals can prove their AWS expertise to their current and future employers.

The AWS Certified Solutions Architect - Associate exam was the first AWS certification that was launched in 2013, followed by two other role-based certifications: Systems Operations (SysOps) Administrator and Developer Associate later that year. AWS is always releasing new development services and features on its cloud platform; thus, there is a continuous stream of updates on its exam content. Some changes are minor ones, while others are major updates that overhaul the entire certification test.

The very first version of the AWS Certified Developer - Associate exam (DVA-C00) and was released in January 2014. Four years later, in June 2018, AWS released the updated version of this test with an exam code of DVA-C01. AWS unveiled the third iteration of this test with **DVA-C02** as its latest exam code. You might notice that for every major exam update, the last digit of the exam code is incremented, so expect that the next one will be called DVA-C03 after several years. It's quite important to know the relevant details of the AWS exam version to ensure that you are using up-to-date review materials.

Exam Details

The AWS Certified Developer - Associate examination is intended for individuals who perform a development role and have one or more years of hands-on experience developing and maintaining an AWS-based application.

Exam Code:	DVA-C02
No. of Questions:	65
Score Range:	100/1000
Cost:	150 USD
Passing Score:	720/1000
Time Limit:	2 hours 10 minutes (130 minutes)
Format:	Multiple choice/multiple answers
Delivery Method:	Testing center or online proctored exam

Exam Domains

The AWS Certified Developer - Associate exam has four (4) different domains, each with corresponding weight and topic coverage. The exam domains are Development with AWS Services (32%), Security (26%), Deployment (24%), and Troubleshooting and Optimization (18%)

Domain 1: Development with AWS Services (32%)

1.1 Develop code for applications hosted on AWS
1.2 Develop code for AWS Lambda
1.3 Use data stores in application development

Domain 2: Security (26%)

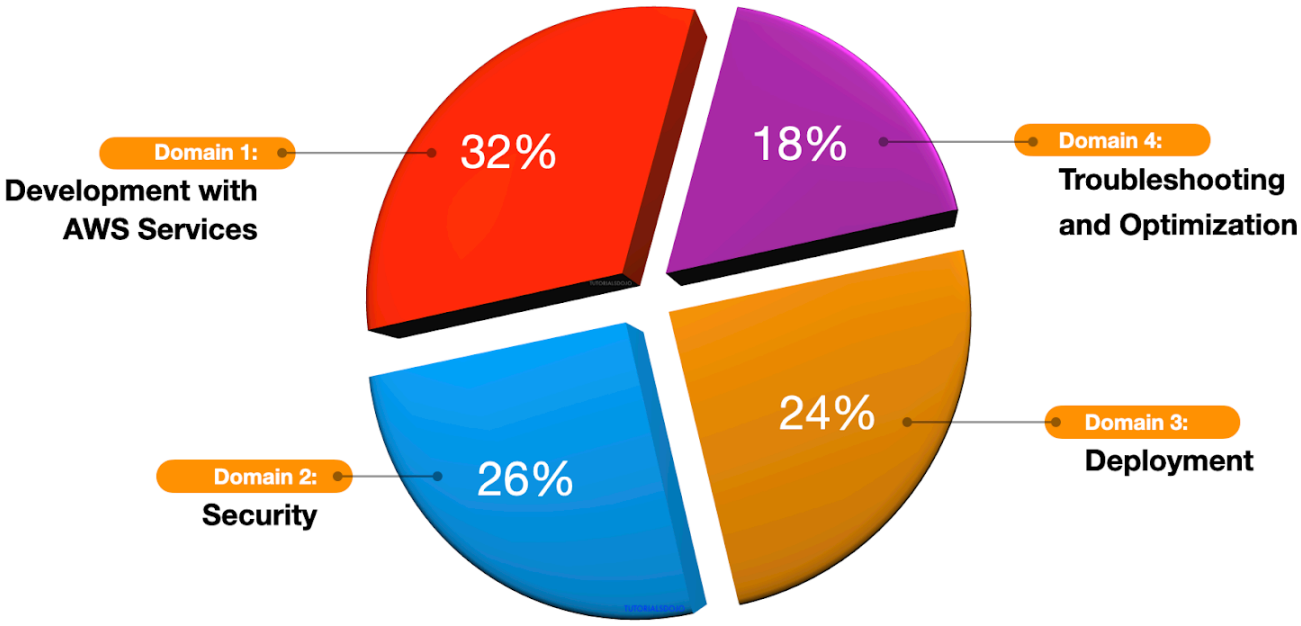
2.1 Implement authentication and/or authorization for applications and AWS services
2.2 Implement encryption by using AWS services
2.3 Manage sensitive data in application code.

Domain 3: Deployment (24%)

3.1 Prepare application artifacts to be deployed to AWS.
3.2 Test applications in development environments
3.3 Automate deployment testing
3.4 Deploy code by using AWS CI/CD services.

Domain 4: Troubleshooting and Optimization (18%)

5.1 Assist in a root cause analysis
5.2 Instrument code for observability.
5.3 Optimize applications by using AWS services and features.



Exam Scoring

You can get a score from 100 to 1,000 with a minimum passing score of **720** when you take the AWS Certified Developer Associate exam. AWS uses a scaled scoring model to associate scores across multiple exam types that may have different levels of difficulty. Your complete score report will be sent to you by email 1 - 5 business days after your exam. However, as soon as you finish your exam, you'll immediately see a pass or fail notification on the testing screen.

For individuals who unfortunately do not pass their exams, you must wait 14 days before you are allowed to retake the exam. There is no hard limit on the number of attempts you can retake an exam. Once you pass, you'll receive various benefits, such as a discount coupon which you can use for your next AWS exam.

Once you receive your score report via email, the result should also be saved in your AWS Certification account already. The score report contains a table of your performance on each domain and it will indicate whether you have met the level of competency required for these domains. Take note that you do not need to achieve competency in all domains for you to pass the exam. At the end of the report, there will be a score performance table that highlights your strengths and weaknesses, which will help you determine the areas you need to improve on.

Exam Benefits

If you successfully pass any AWS exam, you will be eligible for the following benefits:

- **Exam Discount** - You'll get a 50% discount voucher to apply for your recertification or any other exam you plan to pursue. To access your discount voucher code, go to the "Benefits" section of your AWS Certification Account, and apply the voucher when you register for your next exam.

- **AWS Certified Store** - All AWS certified professionals would be given access to exclusive AWS Certified merchandise. You can get your store access from the "Benefits" section of your AWS Certification Account.

- **Certification Digital Badges** - You can showcase your achievements to your colleagues and employers with digital badges on your email signatures, Linkedin profile, or on your social media accounts. You can also show your Digital Badge to gain exclusive access to Certification Lounges at AWS re-invent, regional Appreciation Receptions, and select AWS Summit events. To view your badges, Go to the "Digital Badges" section of your AWS Certification Account.

- **Eligibility to join AWS IQ -** With the AWS IQ program, you can monetize your AWS skills online by providing hands-on assistance to customers around the globe. AWS IQ will help you stay sharp and well-versed in various AWS technologies. You can work in the comforts of your home and decide when or where you want to work.

You can visit the official AWS Certification FAQ page to view the frequently asked questions about getting AWS Certified and other information about the AWS Certification: https://aws.amazon.com/certification/faqs/**.**

AWS CERTIFIED DEVELOPER ASSOCIATE EXAM - STUDY GUIDE AND TIPS

The AWS Certified Developer Associate certification is for those who are interested in handling cloud-based applications and services. Typically, applications developed in AWS are sold as products in the AWS Marketplace. This allows other customers to use the customized, cloud-compatible application for their own business needs. Because of this, AWS developers should be proficient in using the AWS CLI, APIs, and SDKs for application development.

The AWS Certified Developer Associate exam (or AWS CDA for short) will test your ability to:

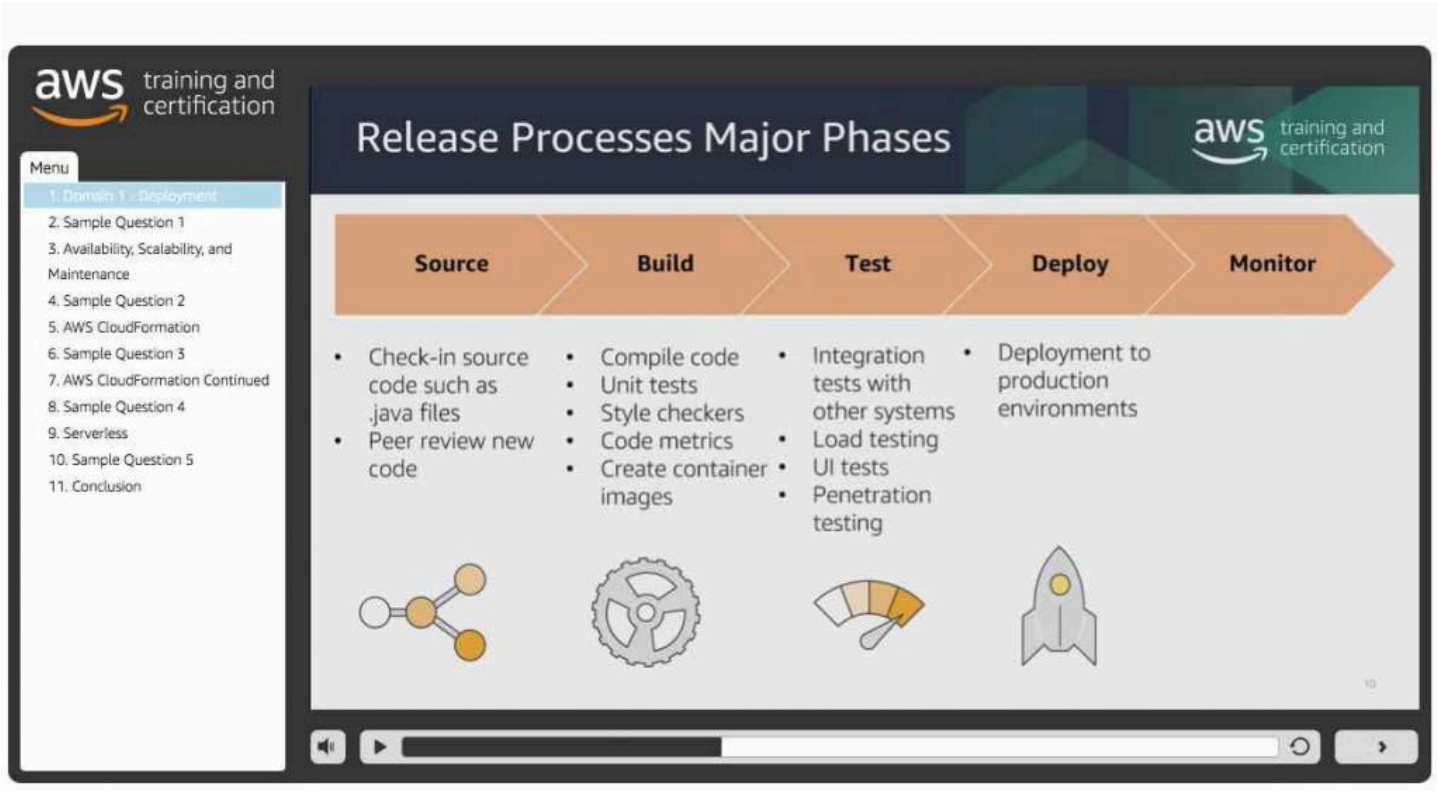
- Demonstrate an understanding of core AWS services, uses, and basic AWS architecture best practices.
- Demonstrate proficiency in developing, deploying, and debugging cloud-based applications using AWS.

Having prior experience in programming and scripting for both standard, containerized, and/or serverless applications will greatly make your review easier. Additionally, we recommend having an AWS account available for you to play around with to better visualize parts in your review that involve code. For more details regarding your exam, you can check out this AWS exam study path.

Study Materials

If you are not well-versed in the fundamentals of AWS, we suggest that you visit our AWS Certified Cloud Practitioner review guide to get started. AWS also offers a free virtual course called AWS Cloud Practitioner Essentials that you can take in their training portal. Knowing the basic concepts and services of AWS will make your review more coherent and understandable for you.

The primary study materials you'll be using for your review are the: FREE AWS Exam Readiness video course, official AWS sample questions, AWS whitepapers, FAQs, AWS cheat sheets, AWS practice exams and AWS video course with included hands-on labs.



For whitepapers, they include the following:

1. Implementing Microservices on AWS – This paper introduces the ways you can implement a microservice system on different AWS Compute platforms. You should study how these systems are built and the reasoning behind the chosen services for that system.

2. Running Containerized Microservices on AWS – This paper talks about the best practices in deploying a containerized microservice system in AWS. Focus on the example scenarios where the best practices are applied, how they are applied, and using which services to do so.

3. Optimizing Enterprise Economics with Serverless Architectures – Read upon the use cases of serverless in different platforms. Understand when it is best to use serverless vs. maintaining your own servers. Also, familiarize yourself with the AWS services that are under the serverless toolkit.

4. Serverless Architectures with AWS Lambda – Learn about Serverless and Lambda as much as you can. Concepts, configurations, code, and architectures are all important and are most likely

to come up in the exam. Creating a Lambda function of your own will help you remember features faster.

5. Practicing Continuous Integration and Continuous Delivery on AWS Accelerating Software Delivery with DevOps – If you are a developer aiming for the DevOps track, then this whitepaper is packed with practices for you to learn. CI/CD involves many stages that allow you to deploy your applications faster. Therefore, you should study the different deployment methods and understand how each of them works. Also, familiarize yourself with the implementation of CI/CD in AWS. We recommend performing a lab of this in your AWS account.

6. Blue/Green Deployments on AWS – Blue/Green Deployments is a popular deployment method that you should learn as an AWS Developer. Study how blue/green deployments are implemented and know the set of AWS services used in application deployment. It is also crucial that you understand the scenarios where blue/green deployments are beneficial and where they are not. Do NOT mix up your blue environment from your green environment.

7. AWS Security Best Practices – Understand the security best practices and their purpose in your environment. Some services offer more than one form of security feature, such as multiple key management schemes for encryption. It is important that you can determine which form is most suitable for the given scenarios in your exam.

8. AWS Well-Architected Framework – This whitepaper is one of the most important papers that you should study for the exam. It discusses the different pillars that make up a well-architected cloud environment. Expect the scenarios in your exam to be heavily based upon these pillars. Each pillar will have a corresponding whitepaper of its own that discusses the respective pillar in more detail.

AWS Services to Focus On

AWS offers extensive documentation and well-written FAQs for all of its services. These two will be your primary source of information when studying AWS. You need to be well-versed in many AWS products and services since you will almost always be using them in your work. I recommend checking out Tutorials Dojo's AWS Cheat Sheets, which provide a summarized but highly informative set of notes and tips for your review on these services.

Services to study for:

1. Amazon EC2 / ELB / Auto Scaling – Be comfortable with integrating EC2 to ELBs and Auto Scaling. Study the commonly used AWS CLI commands, APIs and SDK code under these

services. Focus as well on security, maintaining high availability, and enabling network connectivity from your ELB to your EC2 instances.AWS Elastic Beanstalk – Know when Elastic Beanstalk is more appropriate to use than other computing or infrastructure-as-a-code solutions like CloudFormation. Experiment with the service yourself in your AWS account, and understand how you can deploy and maintain your own application in Beanstalk.

2. Amazon ECS – Study how you can manage your own cluster using ECS. Also, figure out how ECS can be integrated into a CI/CD pipeline. Be sure to read the FAQs thoroughly since the exam includes multiple questions about containers.

3. AWS Lambda – The best way to learn Lambda is to create a function yourself. Also, remember that Lambda allows custom runtimes that a customer can provide himself. Figure out what services can be integrated with Lambda and how Lambda functions can capture and manipulate incoming events. Lastly, study the Serverless Application Model (SAM).

4. Amazon RDS / Amazon Aurora – Understand how RDS integrates with your application through EC2, ECS, Elastic Beanstalk, and more. Compare RDS to DynamoDB and Elasticache and determine when RDS is best used. Also, know when it is better to use Amazon Aurora than Amazon RDS and when RDS is more useful than hosting your own database inside an EC2 instance.

5. Amazon DynamoDB – You should have a complete understanding of the DynamoDB service as this is very crucial in your exam. Read the DynamoDB documentation since it is more detailed and informative than the FAQ. As a developer, you should also know how to provision your own DynamoDB table, and you should be capable of tweaking its settings to meet application requirements.

6. Amazon Elasticache – Elasticache is a caching service you'll often encounter in the exam. Compare and contrast Redis from Memcached. Determine when Elasticache is more suitable than DynamoDB or RDS.

7. Amazon S3 – S3 is usually your go-to storage for objects. Study how you can secure your objects through KMS encryption, ACLs, and bucket policies. Know how S3 stores your objects to keep them highly durable and available. Also, learn about lifecycle policies. Compare S3 to EBS and EFS to know when S3 is more preferred than the other two.

8. Amazon EFS – EFS is used to set up file systems for multiple EC2 instances. Compare and contrast S3 to EFS and EBS. Study file encryption and EFS performance optimization as well.

9. Amazon Kinesis– There are usually tricky questions on Kinesis, so you should read its documentation too. Focus on Kinesis Data Streams and explore the other Kinesis services.

Familiarize yourself with Kinesis APIs, Kinesis Sharding, and integration with storage services such as S3 or compute services like AWS Lambda.

10. Amazon API Gateway – API gateway is usually used together with AWS Lambda as part of the serverless application model. Understand API Gateway's structure, such as resources, stages, and methods. Learn how you can combine API Gateway with other AWS services, such as Lambda or CloudFront. Determine how you can secure your APIs so that only a select number of people can execute them.

11. Amazon Cognito – Cognito is used for mobile and web authentication. You usually encounter Cognito questions in the exam along with Lambda, API Gateway, and DynamoDB. This usually involves some mobile application requiring an easy sign-up/sign-in feature from AWS. It is highly suggested that you try using Cognito to better understand its features.

12. Amazon SQS – Study the purpose of different SQS queues, timeouts, and how your messages are handled inside queues. Messages in an SQS queue are not deleted when polled, so be sure to read on that as well. There are different polling mechanisms in SQS, so you should compare and contrast each.

13. Amazon CloudWatch – CloudWatch is a primary monitoring tool for all your AWS services. Verify that you know what metrics can be found under CloudWatch monitoring and what metrics require a CloudWatch agent installed. Also, study CloudWatch Logs, CloudWatch Alarms, and Billing monitoring. Differentiate the kinds of logs stored in CloudWatch vs. logs stored in CloudTrail.

14. AWS IAM – IAM is the security center of your cloud. Therefore, you should familiarize yourself with the different IAM features. Study how IAM policies are written and what each section in the policy means. Understand the usage of IAM user roles and service roles. You should have read up on the best practices whitepaper in securing your AWS account through IAM.

15. AWS KMS – KMS contains keys that you use to encrypt EBS, S3, and other services. Know what these services are. Learn the different types of KMS keys and in which situations each type of key is used.

16. AWS CodeBuild / AWS CodeDeploy / AWS CodePipeline – These are your tools for implementing CI/CD in AWS. Study how you can build applications in CodeBuild (`buildspec`), and how you'll prepare configuration files (`appspec`) for CodeDeploy. I suggest that you build a simple pipeline of your own in CodePipeline to see how you should manage your code deployments. It is also important to learn how you can roll back to your previous application version after a failed deployment. The whitepapers above should have explained in-place deployments and blue/green deployments and how to perform automation.

17. AWS CloudFormation – Study the structure of CloudFormation scripts and how you can use them to build your infrastructure. Be comfortable with both JSON and YAML formats. Read a bit about StackSets. List down the services that use CloudFormation in the backend for provisioning AWS resources, such as AWS SAM, and processes, such as in CI/CD.

Aside from the concepts and services, you should study the AWS CLI, the different commonly used APIs (for services such as EC2, EBS, or Lambda), and the AWS SDKs. Read up on the AWS Serverless Application Model (AWS SAM) and AWS Application Migration Service, as well as these that may come up in the exam. It will also be very helpful to have experience interacting with AWS APIs and SDKs and troubleshooting any errors that you encounter while using them.

Common Exam Scenarios

Scenario	Solution
AWS Lambda	
An application running in a local server is converted to a Lambda function. When the function was tested, an *Unable to import module* error showed.	Install the missing modules in your application's folder and package them into a ZIP file before uploading to AWS Lambda.
A Developer is writing a Lambda function that will be used to send a request to an API in different environments (Prod, Dev, Test). The function needs to automatically invoke the correct API call based on the environment.	Use Environment Variables
A Lambda function needs temporary storage to store files while executing.	Store the files in the `/tmp` directory
Lambda function is writing data into an RDS database. The function needs to reuse the database connection to reduce execution time.	Use execution context by placing the database connection logic outside of the event handler.
A Developer needs to increase the CPU available to a Lambda function to process data more efficiently.	Increase the allocated memory of the function.
Amazon API Gateway	
A Developer has an application that uses a RESTful API hosted in API Gateway. The API requests are failing with a `"No 'Access-Control-Allow-Origin'`	Enable CORS in the API Gateway Console.

`header is present on the requested resource"` error message.	
A website integrated with API Gateway requires user requests to reach the backend server **without intervention** from the API Gateway. Which integration type should be used?	`HTTP_PROXY`
A serverless application is composed of AWS Lambda, DynamoDB, and API Gateway. Users are complaining about getting HTTP 504 errors.	The API requests are reaching the maximum integration timeout for API Gateway (29 seconds).
How to invalidate API Gateway cache?	1. Send a request with a `Cache-Control: max-age` header. 2. Enable the `Require Authorization` option on your API cache settings.
A developer needs to deploy different API versions in API Gateway	Use stage variables
Amazon DynamoDB	
A Developer needs a cost-effective solution to delete session data in a DynamoDB table.	Expire session data with DynamoDB TTL
New changes to a DynamoDB table should be recorded in another DynamoDB table.	Use DynamoDB Streams
Reduce the DynamoDB database response time.	Use DynamoDB Accelerator (DAX)
Choosing the best partition key for the DynamDB table.	Use the partition key with the highest cardinality (e.g. student ID, employee ID)
An application uses a DynamoDB database with Global Secondary Index. DynamoDB requests are returning a `ProvisionedThroughputExceededException` error. Why is this happening?	The write capacity of the GSI is less than the base table.
CloudFormation and AWS SAM	
What section must be added to a CloudFormation template to include resources defined by AWS SAM?	Transform
A developer needs a reliable framework for building serverless applications in AWS	AWS SAM

A CloudFormation stack creation process failed unexpectedly.	CloudFormation will roll back by deleting resources that it has already created.
A CloudFormation template will be used across multiple AWS accounts	Use CloudFormation StackSets
Deployment and Security	
It is required that incoming traffic is shifted in two increments. 10% of the traffic must be shifted in the first increment, and the remaining 90% should be deployed after some minutes.	Canary
You need to authenticate users of a website using social media identity profiles.	Amazon Cognito Identity Pools
A company has two accounts. The developers from Account A need to access resources on Account B.	Use cross-account access role
Multiple developers need to make incremental code updates to a single project and then deploy the new changes.	Use GitHub/GitLab/Bitbucket as the code repository and directly deploy the new package using AWS CodeDeploy.
A development team is using CodePipeline to automate their deployment process. The code changes must be reviewed by a person before releasing to production	Add a manual approval action stage
Relevant API/CLI commands	
A Developer needs to decode an encoded authorization failure message.	Use the `aws sts decode-authorization-message` command.
How can a Developer verify permission to call a CLI command without actually making a request?	Use the `--dry-run` parameter along with the CLI command.
A Developer needs to deploy a CloudFormation template from a local computer.	Use the `aws cloudformation package` and `aws cloudformation deploy` command
A Developer has to ensure that no applications can fetch a message from an SQS queue that's being processed or has already been processed.	Increase the `VisibilityTimeout` value using the `ChangeMessageVisibility` API and delete the message using the `DeleteMessage` API.
A Developer has created an IAM Role for an application that uploads files to an S3 bucket. Which API call should the Developer use to allow the application to make upload requests?	Use the `AssumeRole` API

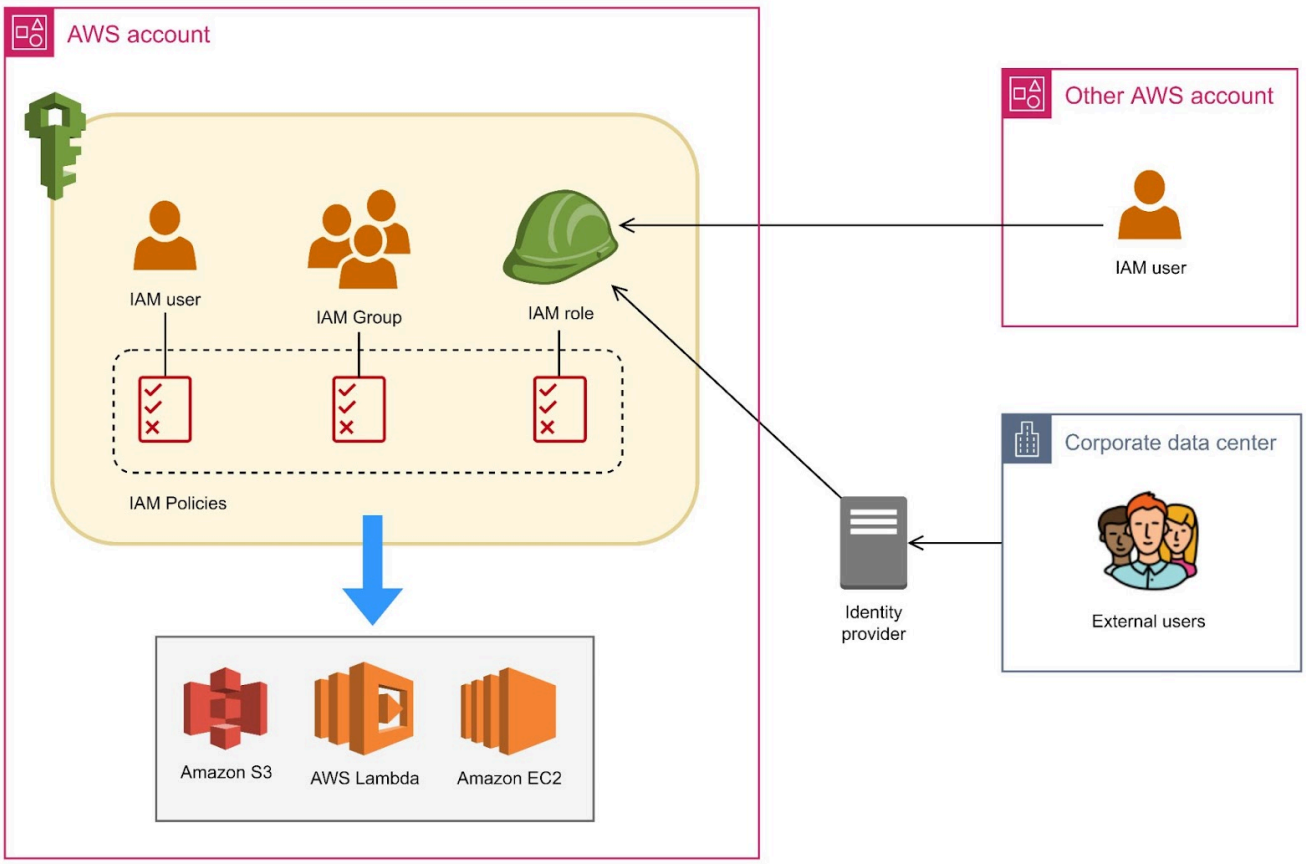
AWS Deep Dives

AWS Identity Access Management (IAM)

IAM is the primary tool for controlling and managing access to an AWS account. It sits at the core of AWS security; everything you do with AWS, whether it's creating a Lambda function, uploading a file to an S3 bucket, or something mundane as viewing EC2 instances on the Console, is governed by IAM. It allows you to specify who, which AWS resources, as well as what actions they can and cannot do. These are also known as *authentication* and *authorization*.

IAM Identities

IAM identities handle the authentication aspect of your AWS account. It pertains to any user, application, or group that belongs to your organization.

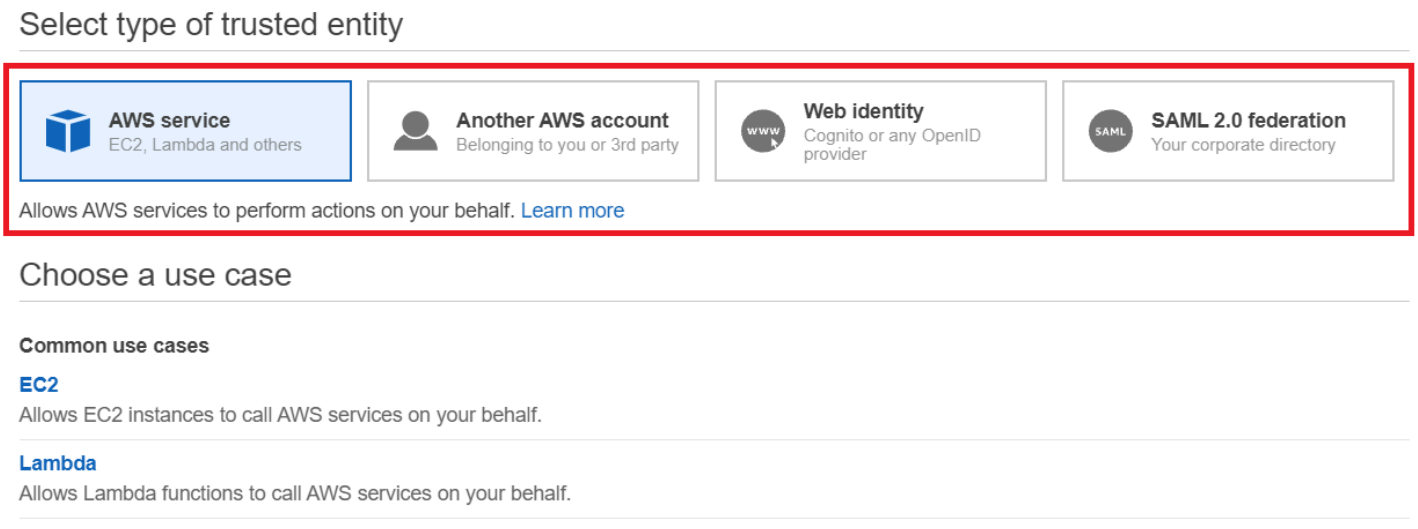


An IAM identity can be an **IAM User**, **IAM role**, or **IAM Group**.

An **IAM User** represents the user or application who interacts with AWS services. Take note that an IAM user is different from the root user. Although full admin permissions can be given to an IAM user, there are certain actions that only a root user can carry out, such as deleting an AWS account. IAM users are given long-term credentials for accessing AWS services. These credentials can be in the form of a username and password (for console access) or an access key and secret key (for programmatic access). The former is primarily used when logging into the AWS Console, whereas the latter is used when interacting with AWS Services via CLI/API commands.

An **IAM role** isn't intended to be associated with a unique user, rather, it is meant to be assumed by certain AWS Services or a group of external users with common business roles. Unlike IAM users, roles use time-limited security credentials for accessing AWS. When creating a role, you choose a trusted entity. The trusted entity determines who is authorized to assume the IAM role.

An IAM role can be assumed by AWS Services to perform actions on your behalf, an IAM user within your account or from an external one, and users federated by identity providers that AWS trusts. Examples of these identity providers are Microsoft Active Directory, Facebook, Google, Amazon, or any IdP that is compatible with OpenID Connect (OIDC) and SAML 2.0.



IAM Group is simply a collection of IAM users. The policies attached to an IAM Group are inherited by the IAM users under it. It's possible to assign an IAM user to multiple groups, but groups cannot be nested (a group within a group). IAM group offers an easy way of managing users in your account. For example, if you have a team of developers that needs access to AWS Lambda, instead of attaching the necessary permission for them individually, you can put the developers together in an IAM Group called 'Developers' and associate the

Lambda permissions to that group. If a new member joins the team, simply add him/her to the 'Developers' IAM group.

IAM Policy

An IAM identity cannot perform AWS actions without an IAM Policy attached to it unless the resource being accessed allows the IAM Identity to do so. An IAM Policy is what authorizes an IAM user or role to control and access your AWS resources. There are three types of IAM Policies to choose from:

- AWS Managed Policies - these are built-in policies that have been constructed to conform to common use cases and job roles. For example, let's say you're working as a system administrator, and you have to give your new database administrator the necessary permissions to do his/her job. Rather than figuring out the right permissions, you may simply use the `DatabaseAdministrator` managed policy, which grants complete access to AWS services necessary to set up and configure AWS database services. AWS Managed Policies cannot be deleted or modified.

- Customer-managed Policy - this refers to the policies that you manually create in your account.

- Inline Policy - a policy that is embedded in an IAM Identity. Unlike AWS Managed Policies and Customer-managed Policies, an inline policy does not have its own ARN; thus, it can't be referenced by other IAM identities. It is scoped to a specific IAM user or role.

References:
https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html
https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html
https://docs.aws.amazon.com/IAM/latest/UserGuide/id_groups.html

IAM Policy Structure

An IAM Policy is expressed as a JSON document. It is made up of two components: policy-wide information and one or more statements. The policy-wide information is an optional element that is placed on top of the document. It's usually just a Version element pertaining to the AWS policy language version being used. This element usually has a static value of `2012-10-17`, referring to the release date of the current AWS policy access language.

The Statement block is where you add the permissions you need for accessing various AWS services. A policy can have single or multiple statements where each statement is enclosed within a bracket.

The following is an example of an IAM Policy.

```
{
    "Version": "2012-10-17",
    "Statement": [{
                "Sid": "AllowFullEC2AccessFromMyNetwork",
                 "Effect": "Allow",
                 "Action": ["ec2:DescribeInstance*"],
                 "Resource": "*",
                 "Condition": {"IpAddress": {"aws:SourceIp": "180.0.111.0/24"}}
            }]
}
```

An individual statement has the following elements:

- `Statement ID (Sid)` - this is just a label that you give to a statement. It's helpful for quickly identifying what a statement is for rather than reviewing what's in the `Action`, `Effect`, or `Resource`. This element is optional; however, it might be useful when modifying or debugging policies with multiple statements.

- `Effect` - explicitly dictates whether the policy allows or denies access to a given resource. It only accepts an `Allow` or a `Deny`.

- `Action` - this element contains the list of actions that the policy allows or denies. You can use the (*) wildcard to grant access to all or a subset of AWS operations. In the example above, the `ec2:DescribeInstance*` refers to all actions that start with the string "DescribeInstance". So this can be referring to `DescribeInstanceAttribute`, `DescribeInstances`, `DescribeInstancesStatus`, and so on.

- `Resource` - the list of AWS resources to which the `Action` element is applied. The example policy above uses a (*) wildcard alone to match all characters. This implies that Action is applicable to all resources. You can be more restrictive to the resources that you want to work with by specifying their Amazon Resource Names (ARN).

- `Condition` - an optional element that you can use to apply logic to your policy when it's in effect. For instance, the sample policy above only allows requests originating from the `180.0.111.0/24` network. Some conditions that you should be aware of are:

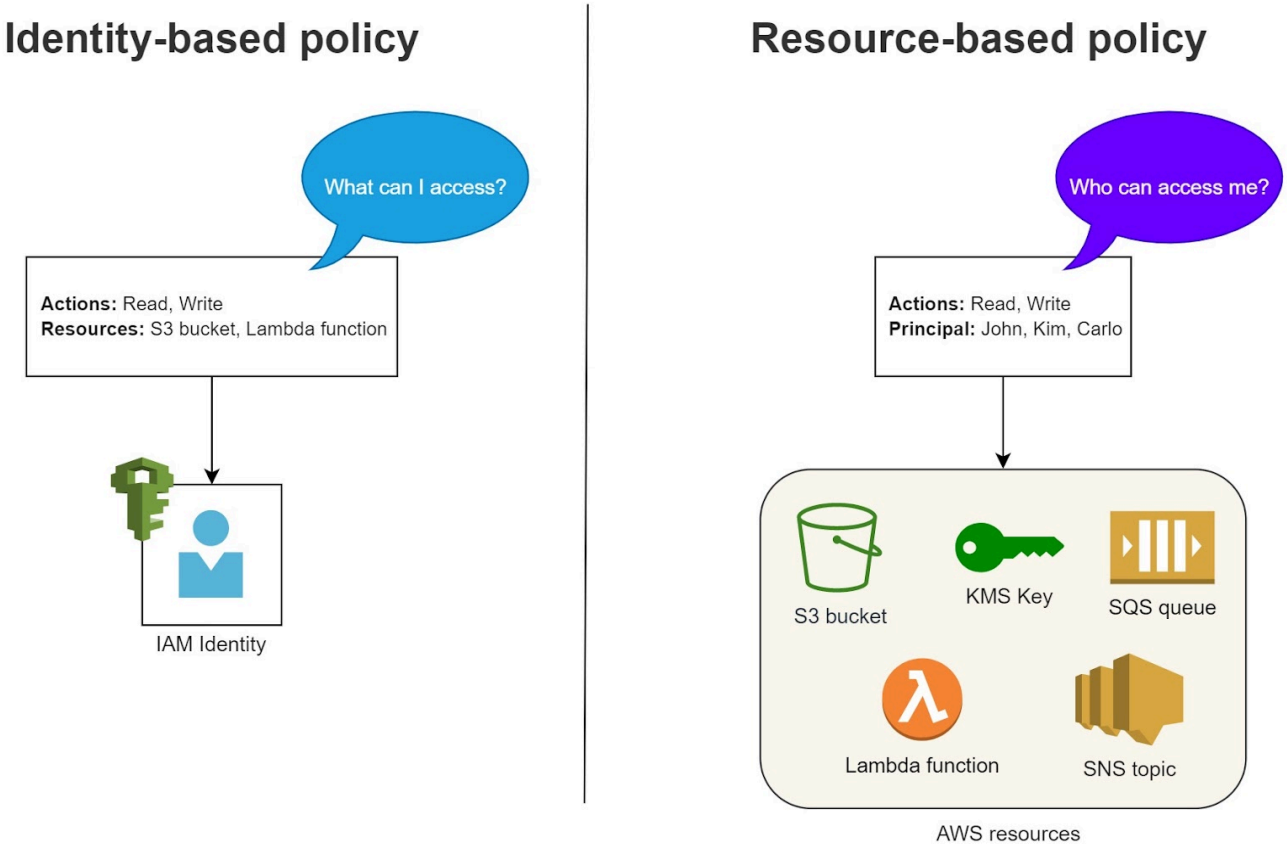
    - `StringEquals` - Exact string matching and case-sensitive
    - `StringNotEquals` - Negated matching
    - `StringLike` - Exact matching but ignoring case
    - `StringNotLike` - Negated matching

- `Bool` - Lets you construct Condition elements that restrict access based on true or false values.
- `IpAddress` - Matching specified IP address or range.
- `NotIpAddress` - All IP addresses except the specified IP address or range
- `ArnEquals, ArnLike`
- `ArnNotEquals, ArnNotLike`
- Use a `Null` condition operator to check if a condition key is present at the time of authorization.
- You can add `IfExists` to the end of any condition operator name (except the `Null` condition)—for example, `StringLikeIfExists`.

References:
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-policy-structure.html
https://aws.amazon.com/blogs/security/back-to-school-understanding-the-iam-policy-grammar/

Identity-based Policy vs. Resource-based Policy

There are six types of IAM Policy: *Identity-based policies, Resource-based policies, IAM permissions boundaries, Service Control Policies, Session policies, and Access Control Lists (ACLs).* For the purpose of the exam, we will focus only on Identity-based and Resource-based policies. The difference between them is pretty simple. The Identity-based policies are policies you attach to an IAM Identity, such as IAM user or IAM role. On the other hand, resource-based policies are attached to AWS resources, such as an S3 bucket (bucket policy), KMS key (key policy), or a Lambda function. Take note that not all AWS resources support resource-based policies.



An **Identity-based policy** defines the resources and actions that an IAM user or role has access to. Since the policy is attached to the IAM user or role, the Principal element doesn't need to be explicitly specified. In other words, the IAM user or role that the policy is attached to is implicitly considered the principal.

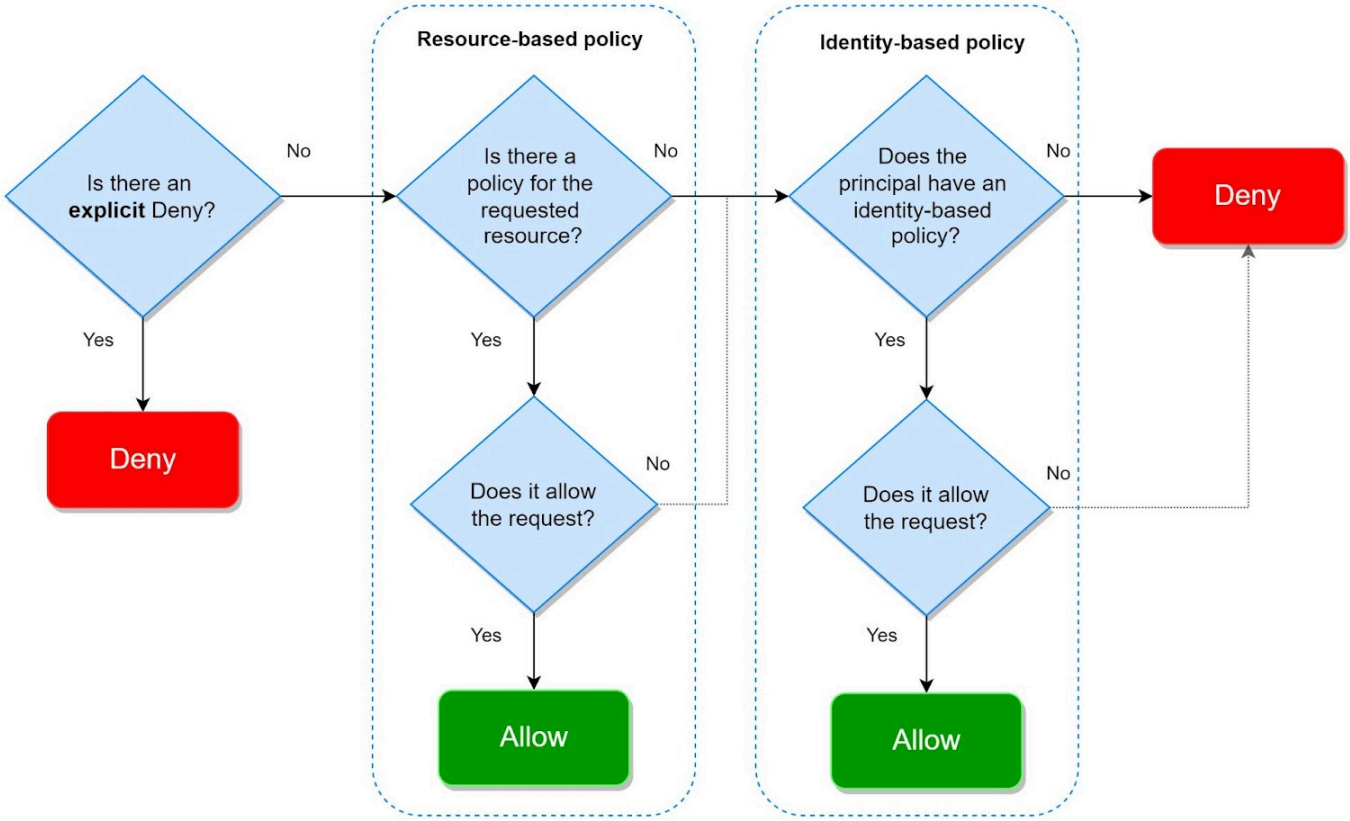
A R**esource-based policy**, on the other hand, must include both the Principal and Resource elements. The Principal element specifies which IAM identities are allowed to access the resource, while the Resource element specifies which resources users are allowed to perform actions on. For example, a bucket policy is a type of resource-based policy that is attached to an Amazon S3 bucket. The policy specifies the actions that are allowed on the bucket and the IAM users or roles that are allowed to perform those actions.

The following S3 bucket policy is an example of how the Resource element is used to grant resource-level permissions. You can see that the `tdojo` S3 bucket contains two folders, `john-folder` and `dave-folder`, to which IAM users John and Dave have read access, respectively.

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::123456789000:user/john"},
      "Action": ["s3:GetObject","s3:GetObjectVersion"],
      "Resource": ["arn:aws:s3:::tdojo/john-folder"]
    },
    {
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::123456789000:user/dave"},
      "Action": ["s3:GetObject","s3:GetObjectVersion"],
      "Resource": ["arn:aws:s3:::tdojo/dave-folder"]
    }]}
```

An important aspect of IAM that you need to understand is how different policies are evaluated. Before determining whether your request is allowed or not, AWS evaluates all explicit `DENY` statements first on all policies that are involved. If the requester is specified in a DENY statement in either of the policies, IAM denies the request. Second, between the resource-based and identity-based policies, resource-based policies are evaluated first. For example, an IAM user with an empty IAM policy will still be able to access an S3 bucket as long as its bucket policy allows the IAM user to perform actions on the bucket.

Here's a stripped-down version of the AWS flow chart on policy evaluation. Only resource-based and IAM-based policies are included for simplicity.



References:
https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_identity-vs-resource.html
https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic.html

Cross-account access

The policy evaluation logic discussed in the previous section is for Intra-account access which involves users and AWS resources interacting with each other within the same account. The evaluation of policies is quite different when it comes to cross-account access.

Cross-account access refers to access between two or more separate AWS accounts. This could be an EC2 instance in one AWS account accessing an S3 bucket in a different AWS account or a user in one account assuming a role in another account to access resources in that account.

Cross-account for IAM users

Imagine an IAM User named 'Tucker' in Account A who needs to download documents from the `private/documents` folder of an S3 bucket in Account B.

To grant this permission, the following actions must be configured:

1. In Account B, the S3 bucket owner creates an S3 bucket policy that specifies Tucker is allowed to download the documents. The policy should be attached to the `private` S3 bucket in Account B.

   Example S3 bucket policy in Account B:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::<Account A ID>:user/Tucker" },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::private/documents/*"
    }
  ]
}
```

2. In Account A, Tucker must have an IAM policy that grants permissions to access the S3 bucket in Account B. The policy allows Tucker to access the S3 bucket without assuming an IAM role.

   Example IAM policy for Tucker in Account A:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DownloadDocuments",
            "Effect": "Allow",
            "Action": [
                "s3:GetOb
```

```
            "Resource": "arn:aws:s3:::private/documents/*"
        }
    ]
}
```

After setting up the required permissions in both accounts, Tucker can now use AWS SDK or CLI to access the S3 bucket in Account B and download the documents.

How is access evaluated?

If Tucker makes a request to the bucket in Account B, AWS performs two evaluations to determine whether the request is allowed or denied, one in the trusting account (AccountB) and one in the trusted account (AccountA). In Account A, AWS checks the identity-based policy of Tucker and any policies that can limit actions that he's requesting. In AccountB, AWS evaluates the bucket policy and any policies that can limit the action being requested. The request is only allowed if both evaluations result in a decision of "Allow". If any policy evaluation returns a decision of "Deny", the access request is denied.

Cross-account for IAM roles

Alternatively, an administrator in Account B can create an IAM role for Tucker to assume. This method provides a more scalable and flexible solution, as the administrator can easily modify or revoke access to the S3 bucket as needed. For instance, if there are other IAM users in Account A that need similar access to the bucket, the administrator can simply grant permission for them to assume the IAM role rather than modifying the S3 bucket policy for each user. This approach reduces administrative overhead in granting access to the bucket.

Here are the steps for how you might grant any IAM users in Account A to assume an IAM role in Account B.

1. In Account B, the IAM administrator creates an IAM role with a policy that grants access to the S3 bucket.

   Example S3 bucket policy in Account B:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DownloadDocuments",
            "Effect": "Allow",
            "Action": [
```

```
                "s3:GetObject
            "Resource": "arn:aws:s3:::private/documents/*"
        }
    ]
}
```

2. The IAM administrator must also attach a trust policy to the IAM role, specifying which AWS accounts and entities are allowed to assume the role. In this case, the trust policy must include Account A as a trusted account.

Example trust policy in Account B (Note that the "root" refers to any Principals in Account A, excluding the root user):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::AccountA:root"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3. The policy for an IAM user in Account A must allow the `sts:AssumeRole` action on the ARN of the IAM role in Account B.

Example IAM policy for an IAM user in Account A:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
```

```
      "Resource": "arn:aws:iam::<Account B ID>:role/RoleName"
    }
  ]
}
```

How is access evaluated?

Once the trust policy is evaluated in Account B, AWS verifies whether the IAM user (in Account A) assuming the IAM role is authorized to do so. Once the IAM user assumes the role, AWS checks if the IAM role has sufficient permissions to access the requested resource in the bucket. After that, the S3 bucket policy is evaluated and access will be granted only if all policy evaluations result in a decision of "Allow".

References:
https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_evaluation-logic-cross-account.html
https://aws.amazon.com/blogs/security/how-to-use-trust-policies-with-iam-roles/

IAM:PassRole Permission

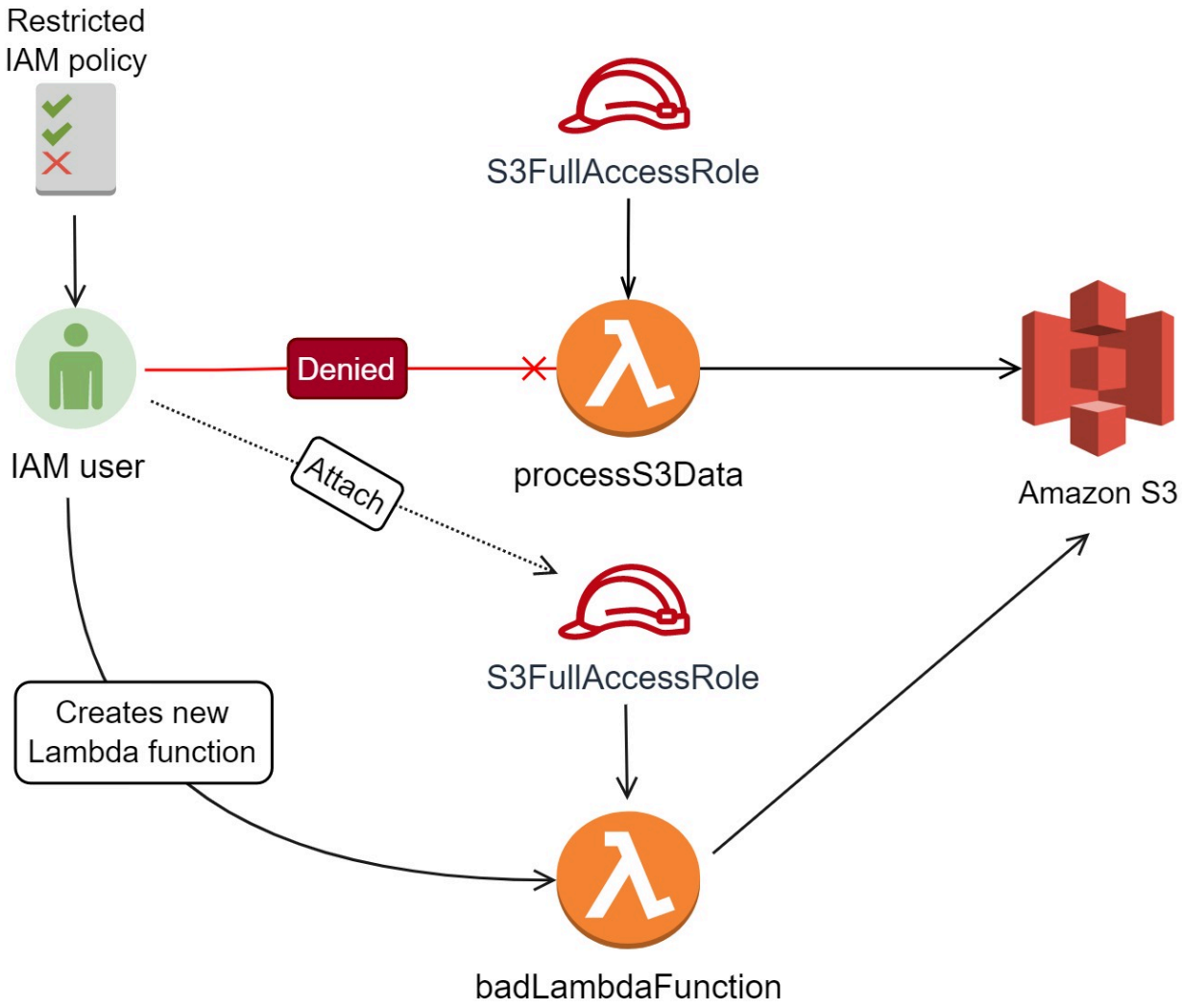
`PassRole` is a special type of IAM permission that permits a user to associate IAM roles to an AWS resource. This is a simple yet powerful permission that warrants due scrutiny when constructing IAM Policies. You can, for example, use the `PassRole` permission to prohibit certain IAM users from passing roles that have greater permissions than the user is allowed to have. Let me paint a scenario for you to explain this.

Say your company has a Lambda function that processes data stored on Amazon S3. An execution role with full S3 access is attached to the Lambda function. The Lambda function runs per schedule and only admins are authorized to make code changes to it. Even though you don't have access to the Lambda function, you can bypass the permissions given to you if the following policy is attached to your IAM user.

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

The preceding IAM policy allows you to pass any IAM roles that exist on your company's AWS account. This creates a security hole that can be exploited to elevate your access. Given you have enough permissions to

create Lambda functions, all you have to do is launch a fresh Lambda function and attach the execution role that has full S3 access. Doing so, even if your IAM user lacks S3 permissions, you'd still be able to access Amazon S3 using the Lambda function.



For better security, be more granular when it comes to providing access. For example, list the specific IAM roles that a user can pass rather than just using a wildcard, like what's shown below:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::123456789123:role/td-cloudwatch-agent-role",
                "arn:aws:iam::123456789123:role/td-s3-role"
            ]
        }
    ]
}
```

References:
https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_passrole.html
https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_examples_iam-passrole-service.html

Additional Notes:
In AWS, Role-Based Access Control (RBAC) is implemented using AWS Identity and Access Management (IAM). RBAC in AWS allows you to define permissions based on roles that align with specific job functions within your organization. Each role has an associated set of permissions specified by IAM policies. These roles can be allocated to anyone, including groups or services, granting them the access they need to carry out their responsibilities. As an example, you can define roles like Admin, Developer, or ReadOnly User, each of which has certain rights that limit what they can do and on which AWS resources. This method simplifies the management of access controls by grouping permissions based on roles rather than managing permissions for each individual user.

References:
https://docs.aws.amazon.com/prescriptive-guidance/latest/saas-multitenant-api-access-authorization/access-control-types.html

AWS STS

AWS Security Token Service (AWS STS) is a global web service that allows you to generate temporary access for IAM users or federated users to gain access to your AWS resources. These temporary credentials are session-based, meaning they're for short-term use only; once expired, they can no longer be used to access your AWS resources.

AWS STS can't be accessed on the AWS console; it is only accessible through API. All STS requests go to a single endpoint at *https://sts.amazonaws.com/*, and logs are then recorded to AWS CloudTrail.

STS API Operations

`AssumeRole`

The `AssumeRole` API operation lets an IAM user assume an IAM role belonging to your account or to an external one (cross-account access). Once the request is successful, AWS generates and returns temporary credentials consisting of an access key ID, a secret access key, and a security token. These credentials can then be used by the IAM user to make requests to AWS services.

`AssumeRoleWithWebIdentity`

The `AssumeRoleWithWebIdentity` API operation returns temporary security credentials for federated users who are authenticated through a public identity provider (e.g., Amazon Cognito, Login with Amazon, Facebook, Google, or any OpenID Connect-compatible identity provider). The temporary credentials can then be used by your application to establish a session with AWS. Just like the `AssumeRole` API, trusted entities who will be assuming the role must be specified. This time, instead of IAM users, it'll be an identity provider.

`AssumeRoleWithWebIdentity` does not require IAM Identities credentials, making it suitable for mobile applications that require access to AWS. The `AssumeRoleWithWebIdentity` is one of the APIs that Amazon Cognito uses under the hood to facilitate the exchange of token and credentials on your behalf. Because Amazon Cognito abstracts the hassles associated with user authentication, it is recommended that you use Amazon Cognito when providing AWS access to application users. However, you may just use `AssumeRoleWithWebIdentity` as a standalone operation.

`AssumeRoleWithSAML`

The `AssumeRoleWithSAML` API operation returns a set of temporary security credentials for federated users who are authenticated by enterprise Identity Providers compatible with SAML 2.0. The users must also use SAML 2.0 (Security Assertion Markup Language) to pass authentication and authorization information to AWS. This API operation is useful for organizations that have integrated their identity systems (such as Windows Active Directory or OpenLDAP) with software that can produce SAML assertions.

`GetFederationToken`

The `GetFederationToken` API operation returns a set of temporary security credentials consisting of a security token, access key, secret key, and expiration for a federated user. This API is usually used for federating access to users authenticated by a custom identity broker and can only be called using the programmatic credentials of an IAM user (IAM Roles are not supported). Although you can use the security credentials of a root user to call `GetFederationToken`, it is not recommended for security reasons. Instead, AWS advises creating an IAM user specifically for a proxy application that does the authentication process.

The default expiration period for this API is significantly longer than `AssumeRole` (12 hours instead of one hour). Since you do not need to obtain new credentials as frequently, the longer expiration period can help reduce the number of calls to AWS.

You can use the temporary credentials created by `GetFederationToken` in any AWS service except the following:

- You cannot call any IAM operations using the AWS CLI or the AWS API.
- You cannot call any STS operations except `GetCallerIdentity`.

`DecodeAuthorizationMessage`

`DecodeAuthorizationMessage` decodes additional information about the authorization status of a request from an encoded message returned in response to an AWS request.

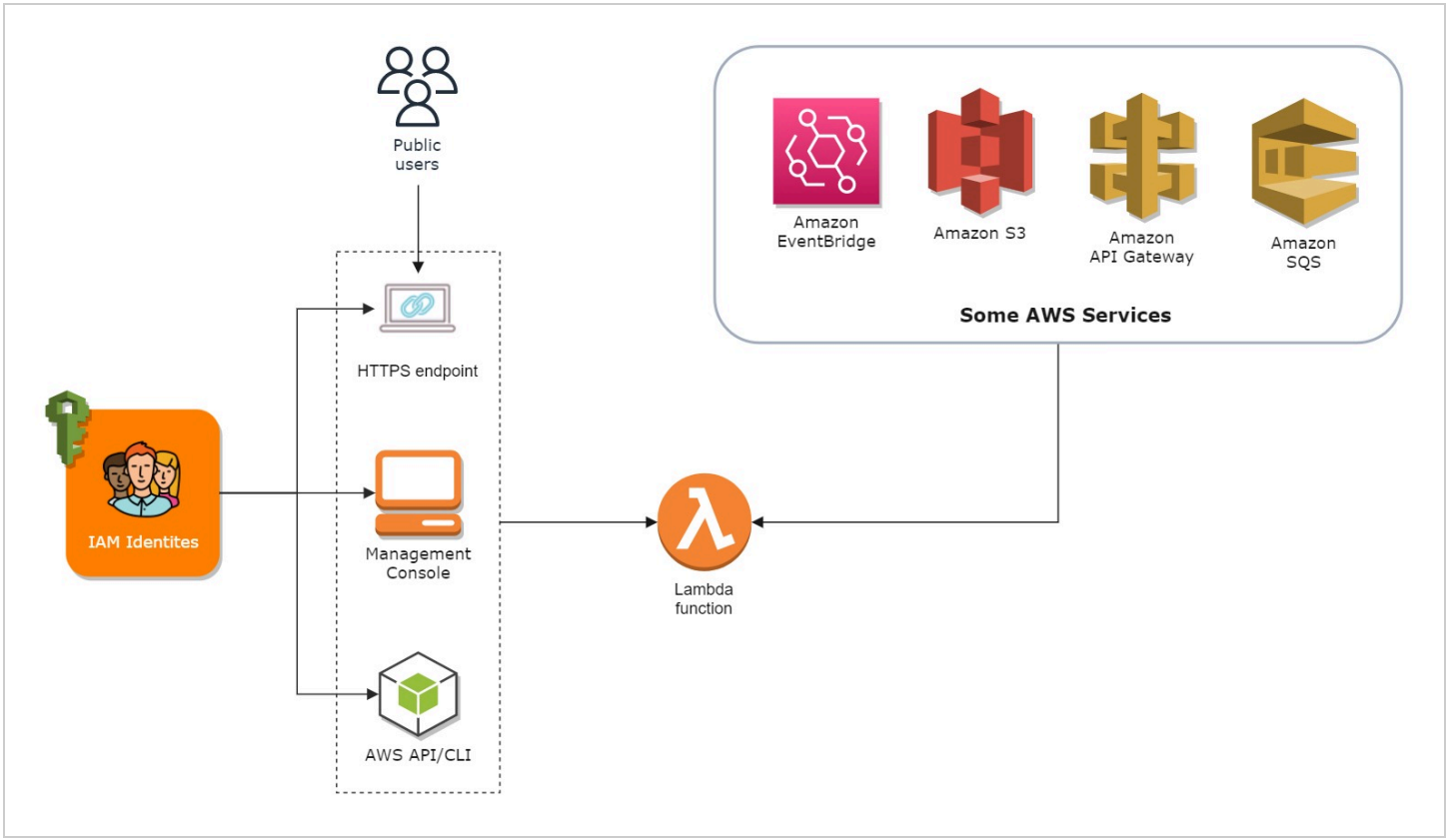
For example, a user might call an API to which he or she does not have access; the request results in a `Client.UnauthorizedOperation` response. Some AWS operations additionally return an encoded message that can provide details about this authorization failure. The message is encoded, so that privilege details about the authorization are hidden from the user who requested the operation. To decode an authorization status message, one must be granted permission via an IAM policy to request the `DecodeAuthorizationMessage` action.

References:
https://docs.aws.amazon.com/STS/latest/APIReference/API_Operations.html
https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_temp_request.html

AWS Lambda

AWS Lambda lets you run codes without managing servers. You don't need to worry about tasks such as scaling, patching, and other management operations that are typically done on EC2 instances or on-premises servers. You can allot the maximum memory available for a Lambda function, as well as the function's execution duration, before timing out. The memory, which scales proportionally to the CPU power, can range from 128 MB to 10,240 MB in 1-MB increments. The default timeout is three seconds, with a maximum value of 900 seconds (15 minutes).

A Lambda function can be invoked in different ways. You can invoke a function directly on the AWS Lambda console, via the `Invoke` API/CLI command, or through a Function URL. You can set up certain AWS Services to invoke a Lambda function as well. For example, you can create a Lambda function that responds to Amazon S3 events (*e.g., processing files as they are uploaded to an S3 bucket*) or set up an Amazon EventBridge rule that triggers a Lambda function every week to perform batch processing. Lambda functions are also commonly used as a backend for APIs that do not require constant load, such as handling login requests or on-the-fly image processing.

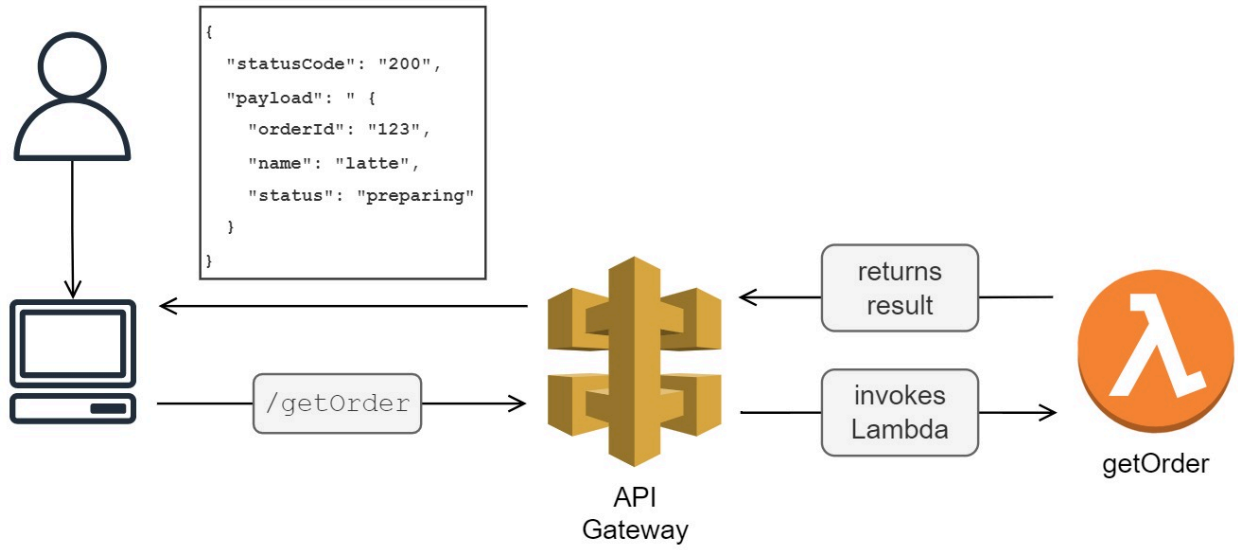


Synchronous vs. Asynchronous Invocations

There are two invocation types in AWS Lambda.

The first type is called Synchronous invocation, which is the default. Synchronous invocation is pretty straightforward. When a function is invoked synchronously, AWS Lambda waits until the function is done processing, then returns the result.

Let's see how this works through the following example:



The image above illustrates a Lambda function-backed API that is managed by API Gateway. When API Gateway receives a GET request from the `/getOrder` resource, it invokes the `getOrder` function. The function receives an event containing the payload, processes it, and then returns the result.

Considerations when using synchronous invocation:

- If you're planning to integrate AWS Lambda with API Gateway, take note that API Gateway's integration timeout is 29 seconds. If a function takes longer than that to complete, the connection will time out, and the request will fail. Hence, use synchronous invocations for applications that don't take too long to complete (e.g., authorizing requests, and interacting with databases).
- Synchronously-invoked functions can accept a payload of up to 6 MB.
- You might need to implement a retry logic in your code to handle intermittent errors.

To call a Lambda function synchronously via API/CLI, set `RequestResponse` as the value for the `invocation-type` parameter when calling the `Invoke` command, as shown below:

```
aws lambda invoke \
  --function-name testFunction  \
      --invocation-type RequestResponse \
          --cli-binary-format raw-in-base64-out \
              --payload '{ "input": "input_value" }' response.json
```

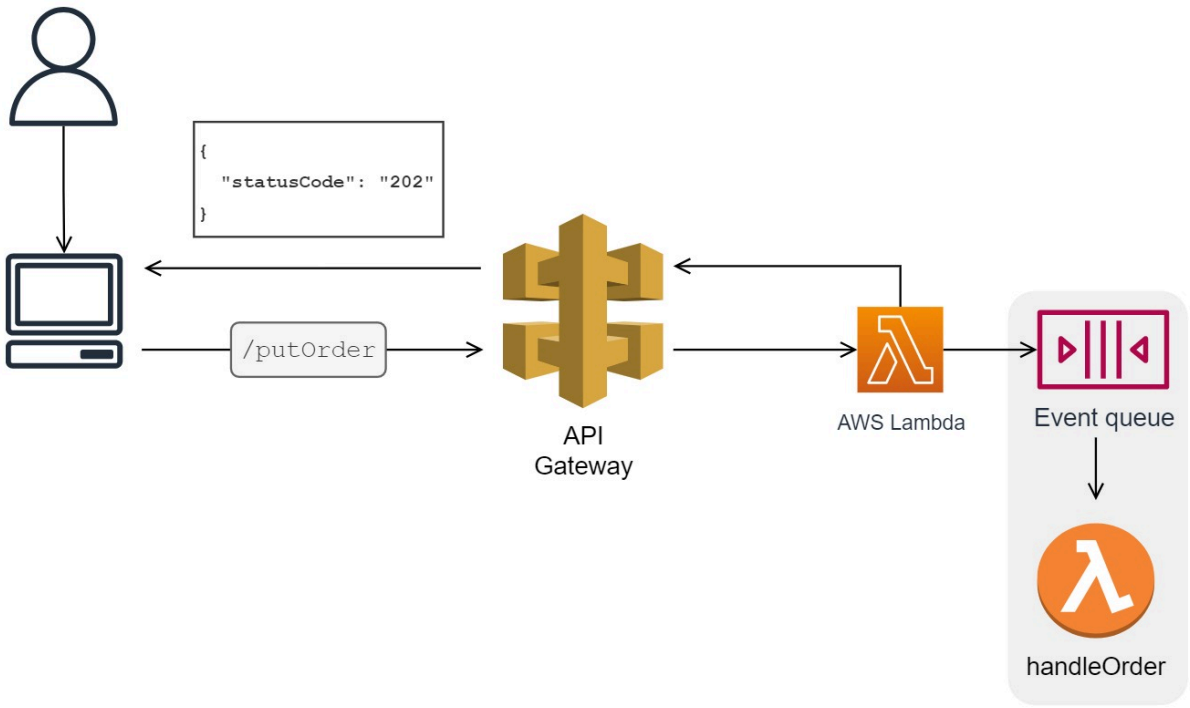
Alternatively, you may just omit the `invocation-type` parameter as AWS Lambda invokes functions synchronously by default.

Services that invoke Lambda functions synchronously: (*services irrelevant to the exam are excluded*):

- Amazon API Gateway
- Application Load Balancer
- Amazon Cognito
- Amazon Data Firehose
- Amazon CloudFront (Lambda@Edge)

An Asynchronous invocation is typically used when a client does not need to wait for immediate results from a function. Some examples of these are long-latency processes that run in the background, such as batch operations, video encoding, and order processing.

When a function is invoked asynchronously, AWS Lambda stores the event in an internal queue that it manages. Let's understand asynchronous invocation through the example below:



A `PUT` request is made to the `/putOrder` resource. Like the previous example, the request goes through API Gateway, which produces an event. This time, instead of API Gateway directly invoking the function, AWS Lambda queues the event. If the event is successfully queued, AWS Lambda returns an empty payload with **`HTTP 202 status code`**. The `202 status code` is just a confirmation that the event is queued; it's not indicative of a successful invocation. The client will not be required to wait for the Lambda function to complete. So, to improve user experience, you can create a worker that tracks order completion and sends notifications once the order is successfully processed.

To call a Lambda function asynchronously via the `Invoke` command, simply set `Event` as the value for the `invocation-type` parameter, as shown below:

```
aws lambda invoke \
  --function-name testFunction  \
      --invocation-type Event \
          --cli-binary-format raw-in-base64-out \
              --payload '{ "input": "input_value" }' response.json
```

Considerations when using asynchronous invocation:

- Asynchronously-invoked functions can only accept a payload of up to 256 KB.

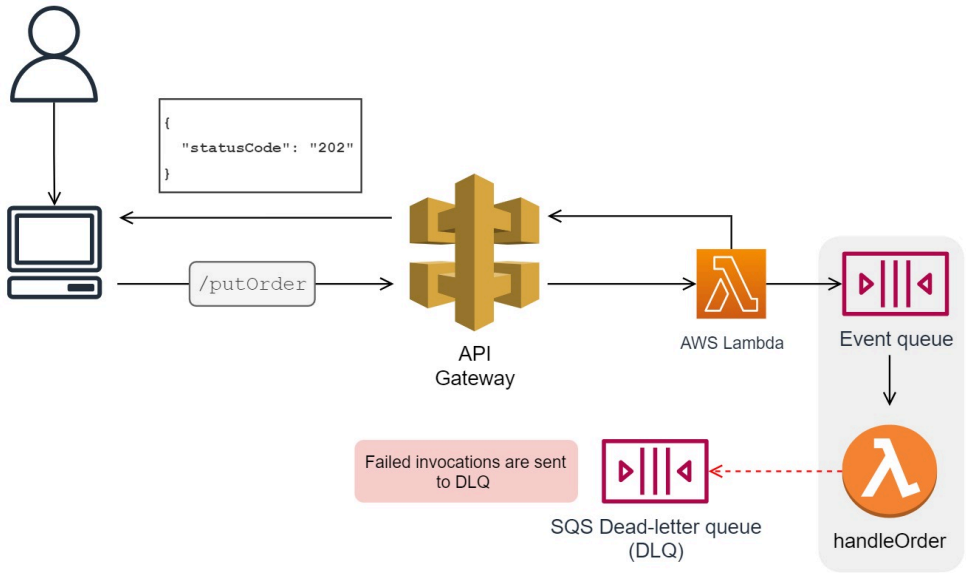
- The Lambda service implements a retry logic for asynchronously-invoked functions
- Good for applications that run in the background (batch processing, video encoding)

Services that invoke Lambda functions asynchronously: (*services irrelevant to the exam are excluded*):

- Amazon API Gateway (by specifying `Event` in the `X-Amz-Invocation-Type` request header of a non-proxy integration API)
- Amazon S3
- Amazon CloudWatch Logs
- Amazon EventBridge
- AWS CloudFormation
- AWS Config

Handling failed asynchronous invocations

AWS Lambda has a built-in retry mechanism for asynchronous invocations. If the function returns an error, Lambda will attempt to retry the request two more times, with a longer wait interval between each attempt.

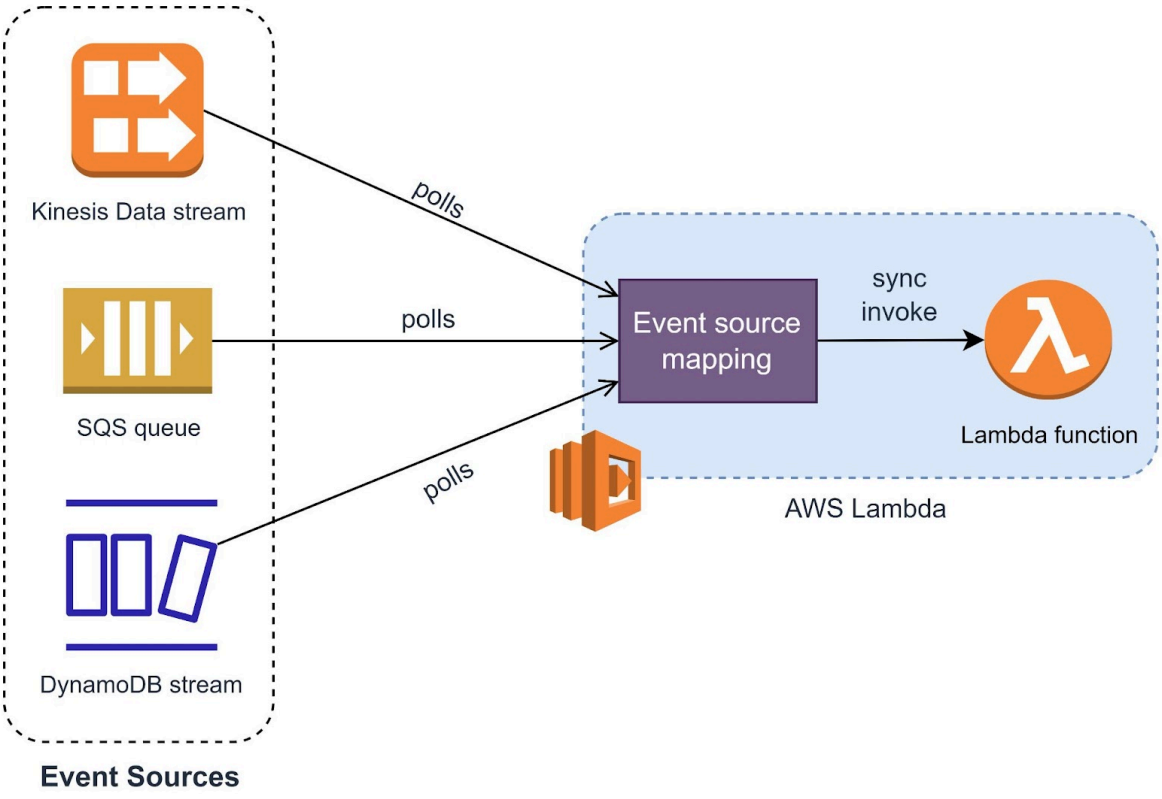


The request is discarded after AWS Lambda has exhausted all remaining retries. To avoid losing events, you may redirect failed request attempts to an SQS dead-letter queue so that you can debug the error later and have another function retry it.

References:
https://docs.aws.amazon.com/lambda/latest/dg/lambda-invocation.html
https://aws.amazon.com/blogs/architecture/understanding-the-different-ways-to-invoke-lambda-functions/

Event source mappings

Stream or queue-based resources such as DynamoDB streams, SQS queues, and Kinesis Data Streams streams do not invoke Lambda functions directly. Typically, we read records from these resources using pollers. A poller is an application that periodically checks a queue, pulls records from it (sometimes in batches), and sends them to a downstream service that will process them.



An event source mapping is a sort of polling agent that Lambda manages. Event source mappings take away the overhead of writing pollers from scratch to retrieve messages from queues/streams. This allows you to focus on building the domain logic of your application.

Event source mapping invokes a function synchronously if one of the following conditions is met:

1. **The batch size is reached** - The minimum batch size can be set to 1, but the default and maximum batch sizes vary on the AWS service that invokes your function.
2. **The maximum batching window is reached** - The batching window is the amount of time Lambda waits to gather and batch records. The default batch window for Amazon Kinesis, Amazon DynamoDB, and

Amazon SQS is 0. This means that a Lambda function will receive batches as quickly as possible. You can tweak the value of the batch window based on the nature of your application.

3. **The total payload is 6 MB** - Because event source mappings invoke functions synchronously, the total payload (event data) that a function can receive is also limited to 6MB (the limit for synchronous invocations). This means that if the maximum record size in a queue is 100KB, then the maximum batch size you can set is 60.

Event filtering

A Lambda function is billed based on how long it runs. How often functions are invoked plays a major factor as well. This is why Lambda is great for scheduled jobs, short-duration tasks, and event-based processes. But does this mean you shouldn't use them for high-volume traffic applications? There is no definitive answer as it will always be on a case-to-case basis. It usually depends on factors such as the requirements of your application and the cost trade-off you're willing to make. Regardless, if you ever find yourself wanting to use Lambda in a high-activity application like stream processing, it's good to know that there are methods available to offset the cost of running functions.

Batching is one cost optimization technique for Lambda functions. By increasing the batch size value, you can reduce the frequency at which your function runs. In addition, you can also filter events that are only needed by your function, thus lowering the cost even further.

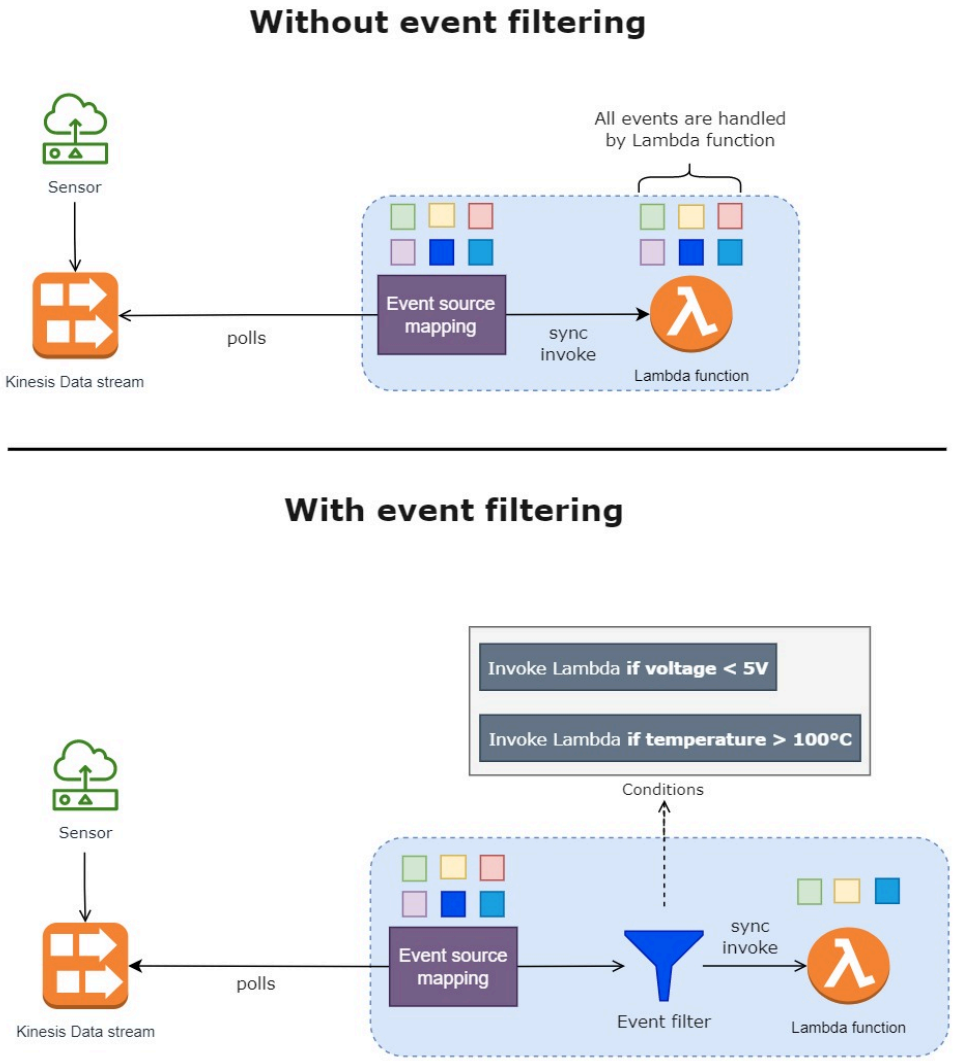
Consider the following scenario:

You're tasked to develop a function that reacts to voltage drops or temperature changes that might indicate faulty components in a site. Multiple sensors send readings to a Kinesis Data Stream stream, which must be consumed and processed by a Lambda function.

Your first instinct might be to implement filtering logic within the Lambda function, as shown below:

```
def lambda_handler(event, context):
      temperature = event["temperature"]
      voltage = event["voltage"]
      if temperature > 100 or voltage < 5:
            #do something
      else:
            #do nothing
```

The code may be valid, but it is deemed inefficient due to cost as the Lambda function will be invoked unnecessarily with every sensor data transmission, even if it doesn't require processing. Supposed sensor data is sent every second. This results in 60 invocations per minute at a batch size of 1 and batching window of 0. This can lead to a waste of time and resources if none of the readings contain the desired voltage and temperature values.



To address this, rather than doing conditional checks for each event at the function level, you should filter the events before they are passed down to your function. Going back to the scenario, since you're only interested in specific values of voltage and temperature, you can specify a filter pattern using the `filter-criteria` parameter of the `CreateEventSourceMapping` command.

```
aws lambda create-event-source-mapping \ --function-name process-sensor-readings \
--batch-size 10 \ --starting-position LATEST \ --event-source-arn
arn:aws:kinesis:us-east-1:123456789123:stream/sensors \ --filter-criteria
'{"Filters": [{"Pattern": "{\"temperature\": [{\"numeric\": [\">\",
100]}]}"},{\"voltage\": [{\"numeric\": [\"<\", 5]}]}]}'
```

A single event source can have up to five unique filters. If an event matches any of the filters, Lambda routes it to your function. Otherwise, the event is discarded. The exam won't expect you to be an expert in event filtering, but you should be familiar with it at a high level. You may also refer to this for the full list of filter syntax.

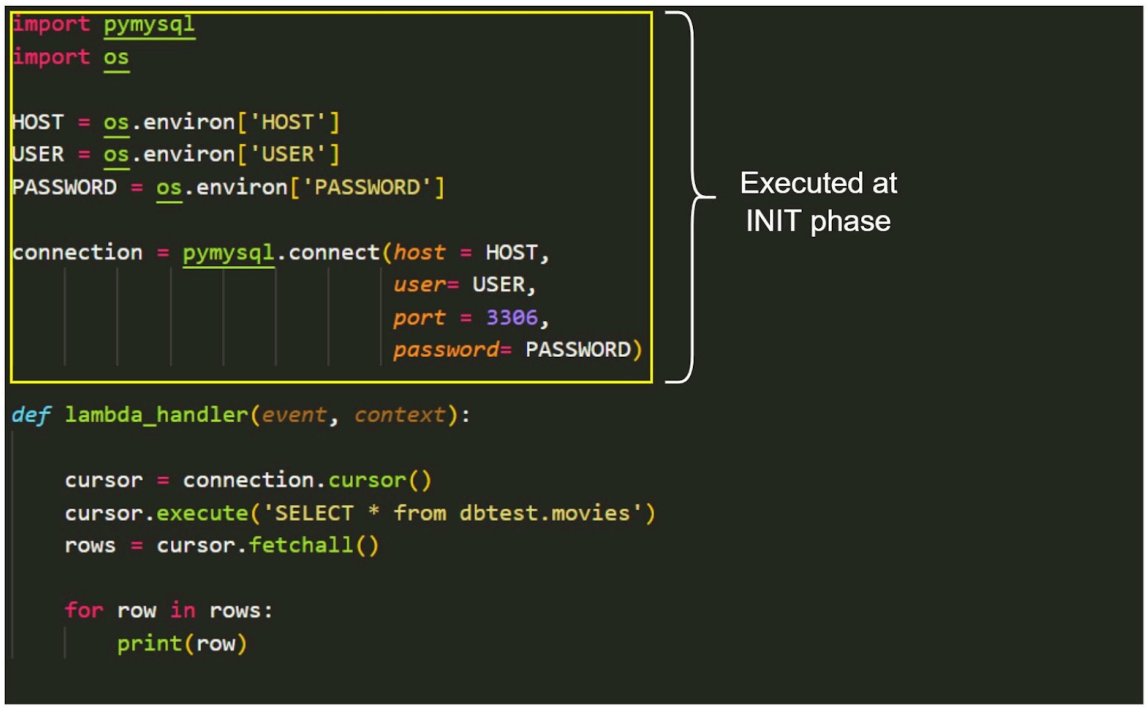
References:
https://docs.aws.amazon.com/lambda/latest/dg/invocation-eventfiltering.html
https://aws.amazon.com/blogs/compute/filtering-event-sources-for-aws-lambda-functions/

Execution Environment Lifecycle

Lambda functions undergo three phases when invoked. These are:

1. INIT Phase

- Occurs when a Lambda function is invoked for the first time after being deployed or after a long period of inactivity.
- Consists of two stages: *environment creation* and *code initialization*.
- Environment creation - AWS Lambda creates an instance of a function in an isolated and secure environment inside a micro virtual machine. This 'execution environment' is where the Lambda function code actually runs. Under the hood, AWS Lambda uses a virtualization technology called *Firecracker* to provision environments. Firecracker makes use of lightweight micro virtual machines, allowing AWS Lambda to create environments quickly, even with a high volume of requests, without sacrificing security or performance. The amount of CPU and memory allocated to the execution environment is determined by the memory settings that you've configured for your Lambda function.
- Initialization - after setting up the environment, Lambda pulls your code from Amazon S3 (Lambda function codes are securely stored in Amazon S3) and runs the initialization code. Initialization code is any code written outside of the handler function. These could be your imported dependencies, global variables, objects, etc.



- The time it takes for the INIT phase to complete adds latency to your function's total execution time. This 'delay' caused by the bootstrapping of execution environments is also known as *cold start*. AWS does not charge for the cold start that happens during the INIT phase; however, initialization times must not exceed 10 seconds.

2. INVOKE Phase

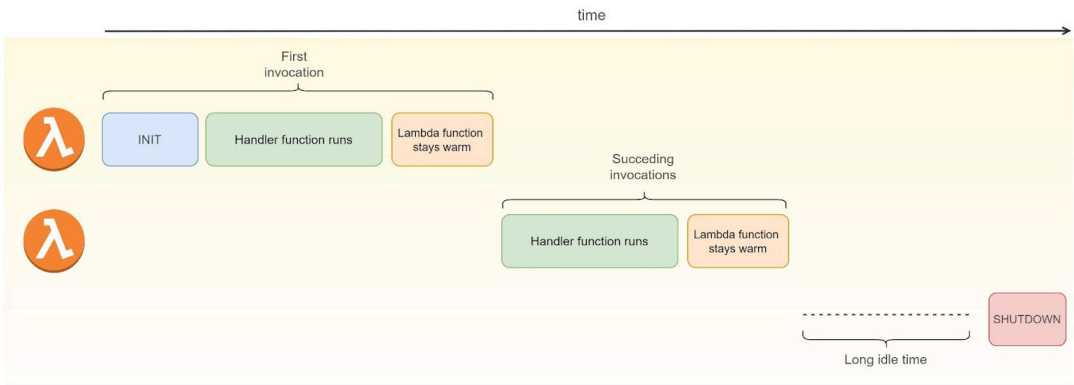
- This is the stage at which the handler function is run. Once the handler function is done processing, AWS Lambda keeps its execution environment warm (on standby) for a period of time. This allows the function to accept subsequent invocation requests without having to create a new execution environment. As a result, the total execution time of your function is shortened because Lambda does not have to repeat everything that was done during the INIT phase.

```
import pymysql
import os

HOST = os.environ['HOST']
USER = os.environ['USER']
PASSWORD = os.environ['PASSWORD']

connection = pymysql.connect(host = HOST,
                             user= USER,
                             port = 3306,
                             password= PASSWORD)

def lambda_handler(event, context):

    cursor = connection.cursor()
    cursor.execute('SELECT * from dbtest.movies')
    rows = cursor.fetchall()

    for row in rows:
        print(row)
```

Executed at INVOKE phase

3. SHUTDOWN Phase
    - When an execution environment stops getting invocation requests for quite some time, AWS Lambda terminates it. There is no documented time limit, but based on experience, execution environments are normally kept warm for 5-15 minutes, though this can vary.

The image below illustrates the lifecycle of a Lambda function.



Reference:
https://docs.aws.amazon.com/lambda/latest/dg/runtimes-context.html

Reducing Cold Starts

The more libraries/packages you include, the longer your function's cold start. That said, you have to be aware and careful in choosing which dependencies to import. There are two approaches you can take to deal with cold starts. The first one is to only load what you need. This method is completely free. You simply have to be more specific and concise when it comes to the resources that you include in your function. In doing so, fewer data will be initialized.

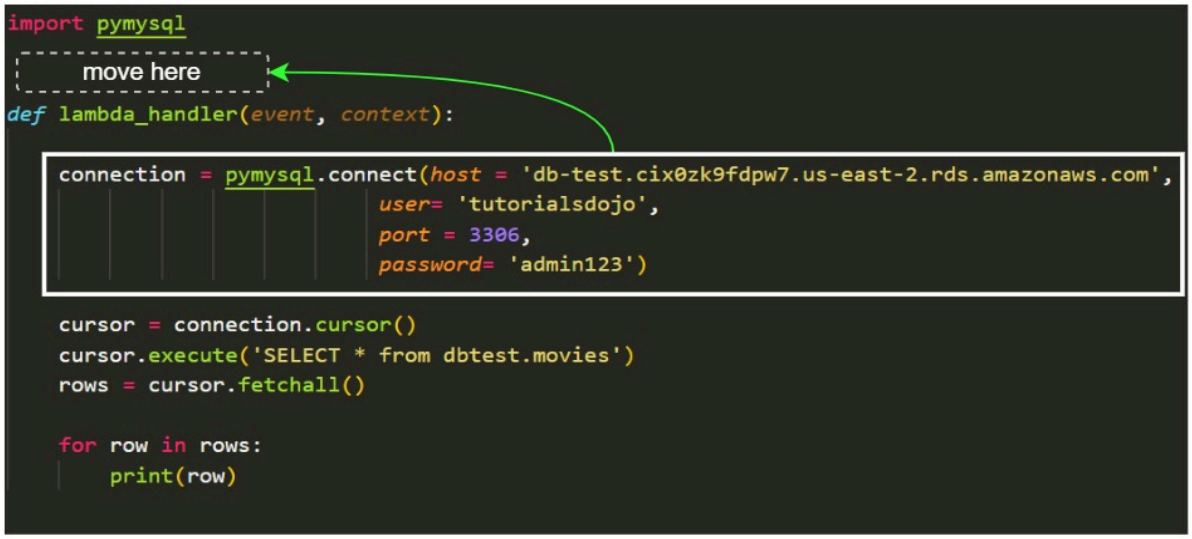
The second is by using Provisioned Concurrency. Provisioned Concurrency is a Lambda feature that allows you to allot pre-warmed environments for your Lambda functions. This will provide your application with constant latency, allowing it to reply to queries in the low double-digit milliseconds. Take note that Provisioned concurrency is different from Reserved concurrency. However, the concept of shared concurrency is the same. The concurrency limit tied to a Region is divided between Shared concurrency and Provisioned concurrency.

References:
https://aws.amazon.com/blogs/compute/operating-lambda-performance-optimization-part-1/
https://aws.amazon.com/blogs/compute/operating-lambda-performance-optimization-part-2/

Execution Environment Reuse

Re-using your existing execution environment is all about reusing the config settings, dependencies, global variables, or even database connections that were already initialized during the INIT phase. To reuse, simply place all global variables, database connections, or SDK clients that you have outside of your handler function.

Below is an example of a Lambda function code where an initialized resource is not taken advantage of. You can see that the database client is written inside the function handler.



If this is the case, your function will open a new database connection every time it is invoked, which is costly compute-wise and adds latency to the execution time. To eliminate this extra time, move the connection object outside the handler function so that it is only initialized once. This way, subsequent requests would be able to reconnect to the existing connection. You are charged for the time it takes your function to run, so a shorter execution duration means you'd pay less.

In addition, keep in mind that each execution environment offers a **512 MB - 10 GB of temporary storage** that can be accessed at the `/tmp` directory. This temporary storage is quite useful for caching any data that your function needs to process at every invocation. Imagine a Lambda function that downloads an S3 file into memory each time it is called. This is inefficient if the file is rarely updated and read by multiple users. Also, because of resource constraints or code performance issues, it is not always feasible to load large data into memory at once. To solve this, instead of directly consuming data from memory, store the data in the `/tmp` directory and process it in chunks. As a result, the next time the same file is requested, your Lambda function will simply retrieve it from its local storage rather than the S3 bucket.

References:
https://docs.aws.amazon.com/lambda/latest/operatorguide/execution-environment.html
https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html

Environment variables

An *environment variable* is a *key-value* pair that you can store and retrieve in your Lambda function. Environment variables are tied to a function and its specific version, which means no other functions can access it except for the function it is associated with. Environment variables are shared between execution environments that belong to the same function.

A common scenario where environment variables are useful is when you want to configure different parameters for an application without touching the code. Say you have a Lambda function that needs to be tested against a development API before using the production API. In this case, instead of hardcoding the API endpoints and keys within the function code, you create two environment variables and pull them using the methods available to the programming language that you're using. That way, you'd be able to change parameters tied to different environments without editing and redeploying your code. Just update the values of your environment variables. Making manual edits in the actual code isn't a big deal, but it can be cumbersome and can lead to errors, so this environment variable is a nifty feature to help you write clean and reusable serverless functions.

There are 4 things that you must consider when creating an environment variable:

1. Keys must start with a letter and are at least two characters.
2. Keys must consist of only letters, numbers, and the underscore character (_).
3. There are environment variables that AWS Lambda uses during the INIT phase. These are called reserved environment variables. A key for a reserved environment variable cannot be used in your function configuration. For example, if you're using an AWS SDK, you might define a variable called 'AWS_REGION'. This will end up to an error since AWS_REGION is a reserved variable.
4. The total size of all environment variables must not exceed 4 KB.

References:
https://docs.aws.amazon.com/lambda/latest/dg/configuration-envvars.html
https://aws.amazon.com/premiumsupport/knowledge-center/lambda-common-environment-variables/
https://aws.amazon.com/premiumsupport/knowledge-center/lambda-environment-variables-iam-access/

Lambda function URL

You can optionally configure an HTTPS endpoint (a.k.a function URL) that maps to a specific alias or version of your Lambda function. Like any other URLs, function URLs can be accessed via web browsers or through HTTP clients like `urllib3`, `request` (in Python), or `axios` (in Node.Js).

When you enable function URL, you get to choose between two auth types: `AWS_IAM` and `NONE`. `AWS_IAM`, as the name implies, is only applicable to IAM identities; the requester must be an IAM user or IAM role. AWS Lambda will make a decision on whether to grant access to a function based on the IAM user's or role's permissions.

Function URLs with a `NONE` auth type, on the other hand, let a Lambda function be invoked publicly. At first glance, it sounds risky, and you may wonder why anyone would ever do this. When you use the NONE auth type, AWS Lambda is no longer responsible for authenticating requests. This gives you the freedom to implement your own authentication logic. You may, for example, allow access to your Lambda function to only those who

are logged in to your website. Aside from that, you may also configure a CORS setting to specify the domain/s from which invocation requests must originate.

References:
https://docs.aws.amazon.com/lambda/latest/dg/lambda-urls.html
https://aws.amazon.com/blogs/aws/announcing-aws-lambda-function-urls-built-in-https-endpoints-for-single-function-microservices/

Deploying Codes with External Dependencies

As developers, whether you're coding at work or building your own project, it is highly likely that you import at least one module, framework, or library. Libraries save time and make you more productive. You wouldn't want to reinvent the wheel and build functions from scratch unless you're facing a unique case that has not been solved before. If you're going to be building applications in AWS Lambda, it's essential that you know how to include dependencies in your Lambda functions.

There are dependencies that come prebuilt to a specific runtime and there are also external ones. The former can be used out of the box. The latter is usually downloaded from a public repository and installed on your computer. Often, this is done with the help of a package manager (e.g., `npm` for `NodeJs`, `pip` for `Python`). AWS Lambda does not have a terminal where you can run `pip/npm` commands to install external dependencies on an execution environment. And it wouldn't make sense to have one since Lambda functions are run on temporary environments.

To deploy codes with external dependencies to AWS Lambda, do the following steps:

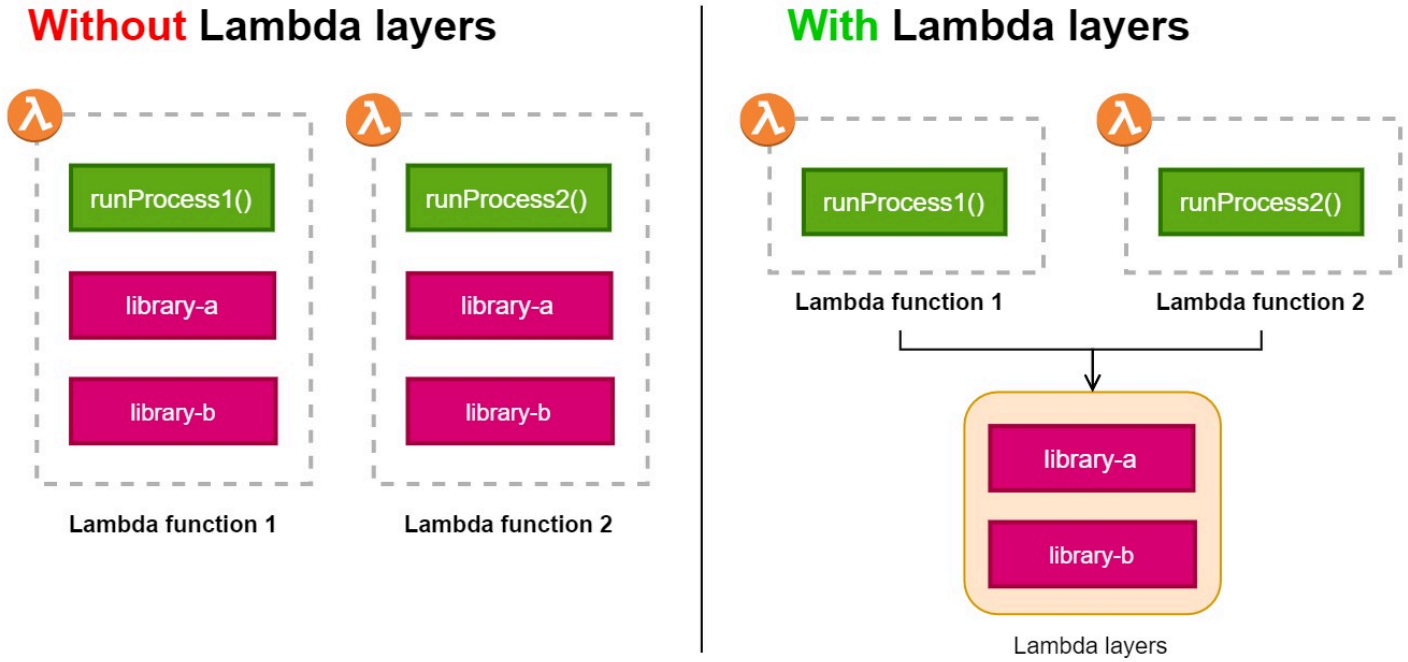
1. Install all external dependencies locally on your application's folder.
2. Create a deployment package by zipping up the project folder. You can achieve this using the native Windows `zip` utility or `zip` command in Linux.
3. Upload the deployment package to AWS Lambda. You can send the file directly to the AWS Lambda Console or store it first to Amazon S3 and deploy it from there.

The first step is always the most important. See to it that you install dependencies required by your application locally on your folder and make sure that you correctly reference them in your code. If not done correctly, you might encounter an `Unable to import module error`, which could mean two things: it's either you've placed the dependencies on a location that Lambda couldn't find, or it is simply non-existent.

References:
https://docs.aws.amazon.com/lambda/latest/dg/python-package.html
https://aws.amazon.com/premiumsupport/knowledge-center/build-python-lambda-deployment-package/

Lambda Layers

Bundling external dependencies along with the function code gets the job done, but it has some drawbacks. First, deploying multiple Lambda functions that have the same dependencies is inefficient and would just bloat the total size of your functions. Second, updating dependencies takes time and effort. Updating each function that shares similar dependencies can slow down the development process. Lambda Layers solves these problems.



Lambda Layers lets you store any additional code (e.g., external dependencies, custom runtimes) separately from your deployment package. Just like a deployment package, external dependencies must be in zip format before being uploaded to AWS Lambda.

Lambda layers can be shared among Lambda functions, making it convenient and easy to update dependencies. In addition, the deployment process becomes more modularized, and your deployment package becomes significantly smaller.

References:
https://docs.aws.amazon.com/serverlessrepo/latest/devguide/sharing-lambda-layers.html
https://aws.amazon.com/blogs/aws/new-for-aws-lambda-use-any-programming-language-and-share-common-components

Concurrency and Throttling

AWS Lambda is inherently scalable, but like many other things, there's a limit to how much it can scale to. It is important that you become aware of this, especially when you're building a high-throughput application using Lambda functions.

Concurrency is the number of Lambda function executions that can run simultaneously for a period of time. By default, AWS imposes a **limit of 1,000 unreserved concurrent executions** across all Lambda functions for every region per account. But you can overcome this limit and further increase concurrent executions up to hundreds of thousands by contacting AWS Support.

You can strategically distribute concurrencies across your functions, but the unreserved count should not go below 100. This means that the default maximum value of allowable **reserved concurrent executions** is limited to 900. The reserved concurrent executions allocated to a function set the maximum number of concurrent instances for that function.

Example:

Lambda function A

- reads and uploads files to Amazon S3.
- 150 concurrent executions are reserved for this function.

Lambda function B

- writes and updates items in a DynamoDB table
- 250 concurrent executions are reserved for this function.

Lambda function C

- processes HTTP requests from API Gateway
- unreserved concurrency.

Simply adding the concurrent executions of functions A and B gives us a total reserved concurrency of 400. That leaves us with 600 unreserved concurrency executions. So what happens when the demand for Lambda function A exceeds 150 concurrent executions? This is where throttling comes into play. When a function hits its maximum concurrency limit, AWS Lambda will reject incoming invocations and return a `429 status code` throttling error. It is not possible for Lambda function A to borrow from the remaining unreserved concurrency pool.

When computing the Concurrency limits for a Lambda function, consider two things:

1. Execution time
2. Number of requests handled per second (requests per second)

Example:

Lambda function

- processes HTTP requests from API Gateway with unreserved concurrency
- Expects an average execution time of 10 seconds
- Expects 150 requests per second.

Multiplying the average execution time by the number of requests per second gives us 1,500 concurrency executions. This is beyond the default limit. If you bring this function to production, you'll end up with a lot of failed requests due to throttling. One of the things you could try doing in this scenario is to optimize the Lambda code and hope to reduce the execution time. A sure alternative would be to increase the default limit by contacting AWS Support.

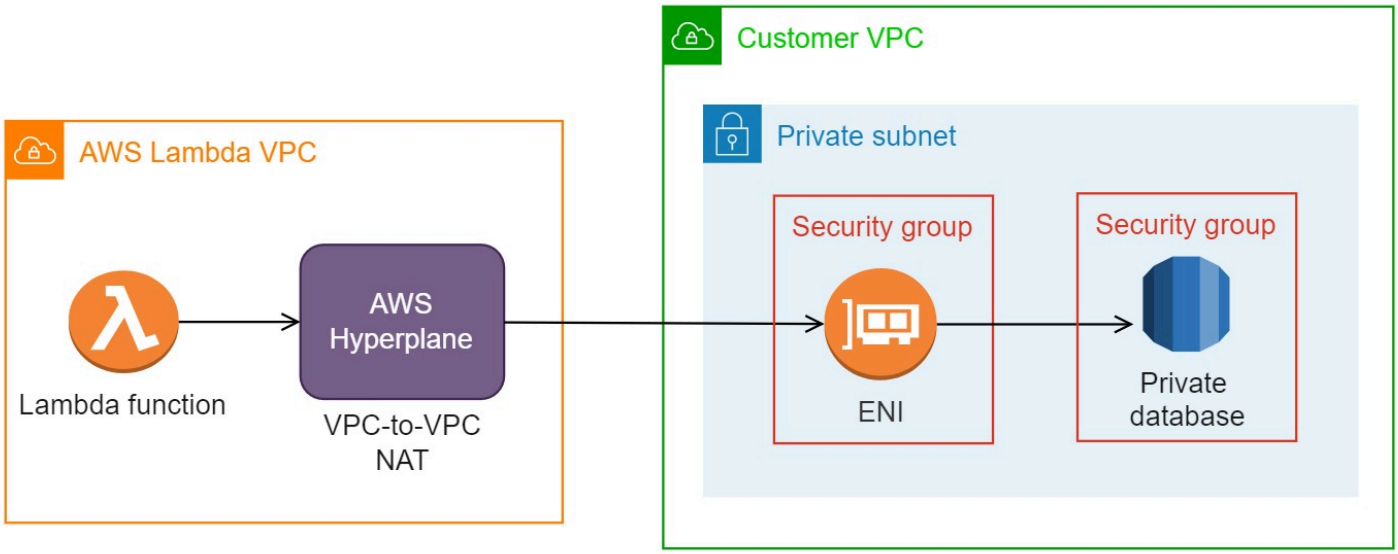
References:
https://docs.aws.amazon.com/lambda/latest/dg/configuration-concurrency.html
https://aws.amazon.com/blogs/compute/managing-aws-lambda-function-concurrency/

Connecting a Lambda Function to a VPC

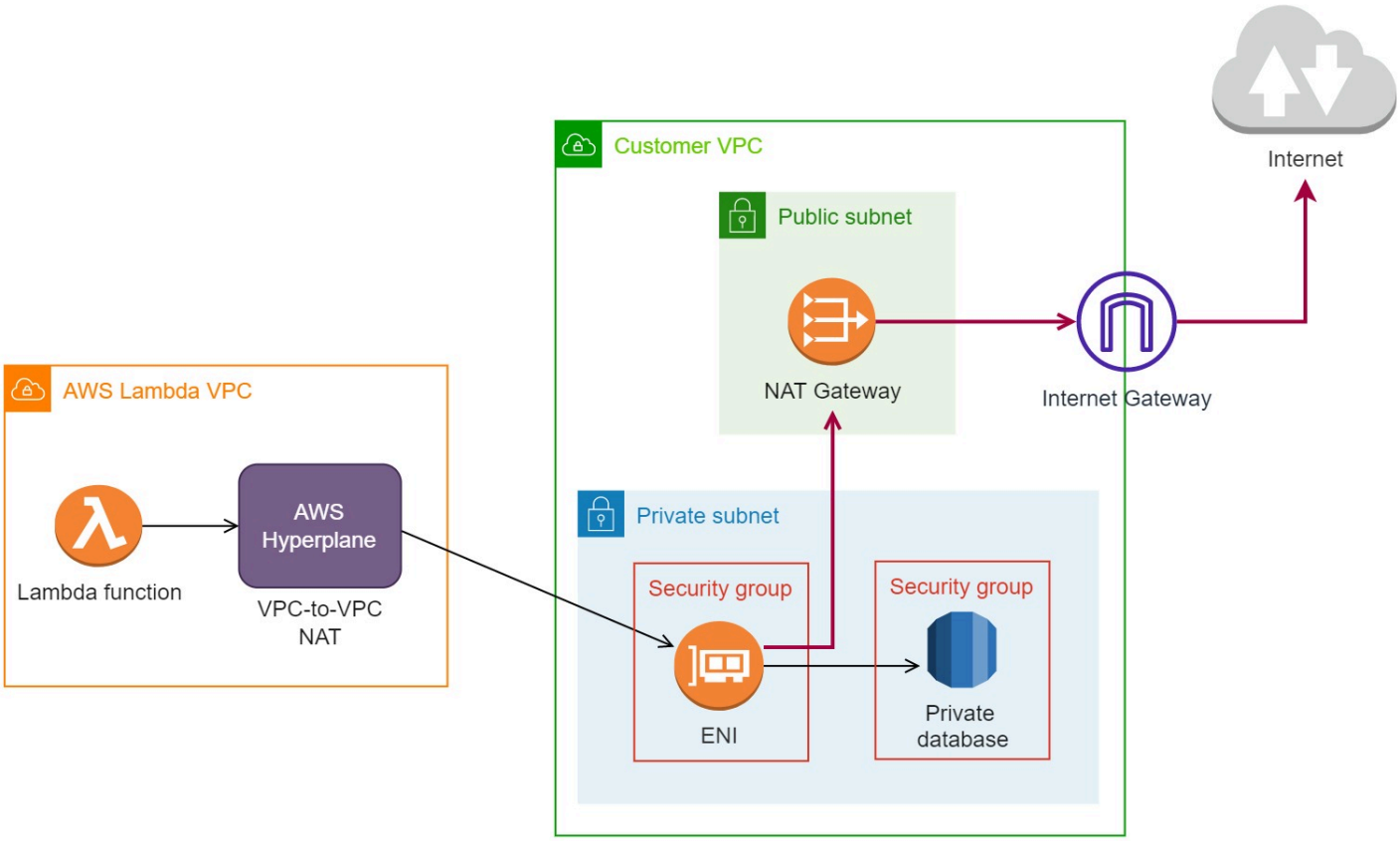
If you want your Lambda function to interact with resources (e.g., RDS database, EC2 instance) inside a private subnet, you won't be able to do so by default. The reason for this is that Lambda functions live in an isolated and secured VPC managed by AWS. This is why when you create a Lambda function, you don't go through any networking configurations (VPC, subnet, ENIs), unlike when creating EC2 instances. You cannot establish a VPC peering connection between the VPC where Lambda functions are run and the VPC where your private resources are located because the former is not accessible to customers. The proper way is to configure your function to connect to a VPC.

To connect your Lambda function to a VPC, first, make sure that your function's execution role has the required permissions to manage the creation and deletion of Elastic Network Interfaces (ENI). This is needed because AWS Lambda creates and deletes elastic network interfaces on subnets that you specify in your function's VPC configuration. AWS uses an internal service called AWS Hyperplane, which serves as a NAT service, connecting Lambda functions to the ENIs in your VPC. Thankfully, there's the **`AWSLambdaVPCAccessExecutionRole`** managed IAM policy, which contains the permissions needed for the job.

Next, specify the VPC where your private resources are located under the Lambda function's network settings. When you create a VPC configuration, you get to choose the subnets where the ENIs are deployed and a security group that controls the traffic between your Lambda function and VPC.



Once connected, your Lambda function will lose internet access. This happens due to the fact that AWS Lambda only assigns private IP addresses to ENIs that it creates. Even if your function is connected to a public subnet, your VPC's internet gateway will still be unable to route traffic between the internet and your function. To give your Lambda function internet access, create a NAT Gateway in the public subnet of your function's VPC and add an entry to the private subnet's route table. Set `0.0.0.0/0` as the destination and the NAT Gateway as the target.



References:
https://aws.amazon.com/premiumsupport/knowledge-center/internet-access-lambda-function
https://docs.aws.amazon.com/lambda/latest/dg/configuration-vpc.html

Versions and Aliases

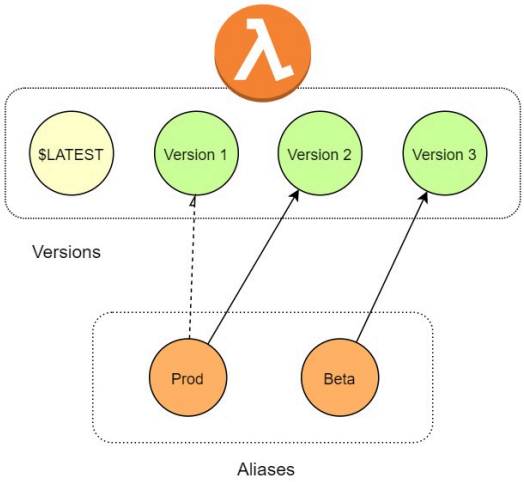
Revisions are almost unavoidable when developing applications, whether you're adding new features or fixing bugs. Instead of directly making changes to the stable version of your Lambda function, you can publish new versions of it where you can play around and experiment at will without affecting the stable version.

A version is a snapshot of a Lambda function's state at a given time. When you publish a new version, a `:version-number` suffix is appended to your function's ARN, which indicates its version. An example is shown below:

```
arn:aws:lambda:us-east-2:123456789123:function:cool-function:1
```

AWS Lambda assigns version numbers monotonically. This means that if you delete a version and publish another one, the sequencing of numbers as you add versions will not reset; rather, it'll just increment.

The unpublished version is also referred to as the `$LATEST` version. If you call a function without any suffix, AWS Lambda will implicitly invoke the `$LATEST` version. Take note that published versions are immutable; any modifications to the code or function configuration are not possible. If you want to make code or config updates, apply the changes to the `$LATEST` version and publish it as another version.



The problem with versions is that you have no control over what is appended to the function's ARN. It can be bothersome to have to update the function's ARN in your code or app parameters every time a new version is released. Wouldn't it be great if there's a pointer that we can use to switch between versions? In this way, you can simply set and forget the ARN of the function. This is where Aliases come into play. You can think of an alias as a nickname that you give to a version number.

To invoke a function with an alias, simply append a `:alias-name` suffix to your function's ARN, just like the example below:

```
arn:aws:lambda:us-east-2:123456789123:function:cool-function:PROD
```

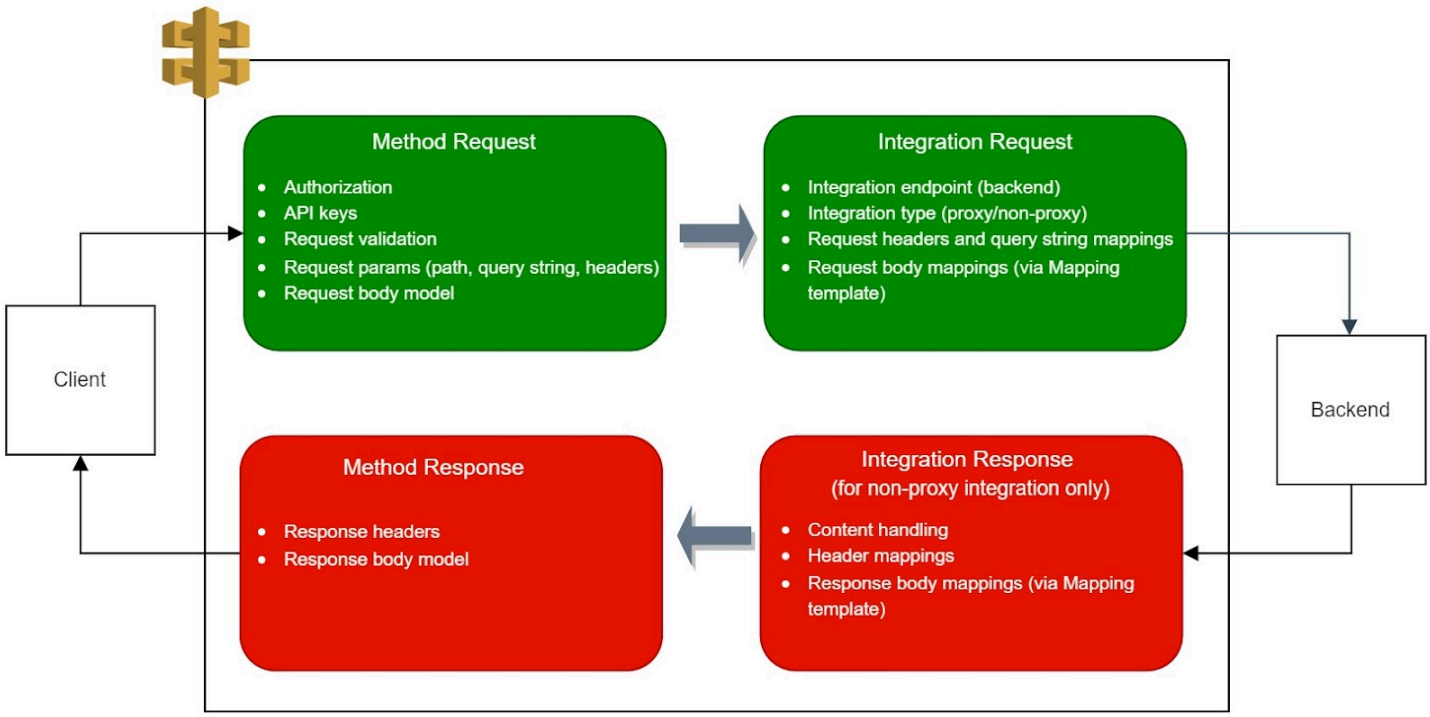
You can change the version that the alias is pointing to in the Lambda Console or via the `UpdateAlias` API.

References:
https://docs.aws.amazon.com/lambda/latest/dg/configuration-aliases.html
https://docs.aws.amazon.com/lambda/latest/dg/configuration-versions.html

Amazon API Gateway

Amazon API Gateway is a fully managed service that allows you to publish, maintain, monitor, and secure your RESTful APIs. It serves as the entry point for your back-end services that are powered by AWS Lambda, Amazon EC2, Amazon ECS, AWS Elastic Beanstalk, or any web application.

The following diagram illustrates how a request flows through API Gateway when an API is called.



1. `Method Request`
    - This section is where client requests are validated. Here, you can set up the authorization type (`AWS_IAM`, `NONE`, Lambda authorizer, Cognito authorizer) to control API access, enable the usage of API keys, or set up a request body validator using a Lambda function.
    - You can also declare in the `Method Request` any input body, query string parameters, and HTTP headers that your API can accept.
2. `Integration Request`
    - The `Integration Request` section contains settings about how API Gateway communicates with the backend of your choosing (Lambda function, HTTP endpoint, Mock, AWS Service, VPC Link) and the integration type (proxy or non-proxy) API Gateway uses.
    - For non-proxy integration, you have the option the use mapping templates to model the structure of the request data that gets forwarded to the backend.
3. `Integration Response`

- This section only applies to a non-proxy integration. The Integration response intercepts the result returned from the backend before it's returned to the client.
- You must configure at least one integration response. The default response is `Passthrough`, which instructs API Gateway to return the response as-is. You may also transform the response to another format (base64 or text).
- Similarly to the Integration request, you have the option to transform the response data before it is returned to the client.

4. `Method Response`
   - Like Method Request, the Method Response is where you can define which HTTP headers the method can return.

REST API vs. HTTP API vs. Websocket API

When you create an API, you get to choose between a REST API, HTTP API, and a WebSocket API endpoint.

- **REST API**
  - For standard use cases, the REST API is what you'll want to use.
  - This gives you complete control over the request and response along with other API management capabilities like caching, creating API keys, and usage plans.
- **HTTP API**
  - Cheaper than REST API
  - Designed for simple applications
  - Lacks other API Gateway features
- **WebSocket API**
  - Typically used for real-time applications (e.g., chat applications, market trading applications)

References:
https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html
https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-basic-concept.html

Proxy vs. Non-proxy integration

In a Proxy integration, client's request is transmitted as is to the backend, including any headers or query parameters. No modification is done to the request data. As for the response, your backend is responsible for returning the response's status code, headers, and the payload to the client.

In a Non-Proxy integration, API Gateway has control over how client data is formatted before it's passed down to your integration backend or before it's returned to the client. For example, instead of feeding the entire request data to your backend, you can filter it first using mapping templates at the `Integration Request`

level to get only the portion that care for. The Non-Proxy integration is a bit more complex to implement and requires you to have knowledge of the Apached Velocity Template Language (VTL), which is the engine that API Gateway uses for mapping templates.

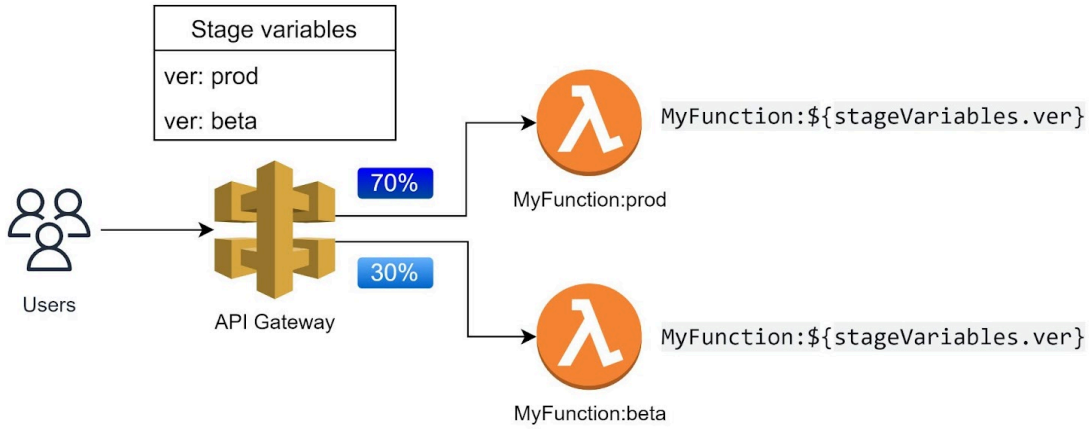
Reference:
https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-api-integration-types.html

Stage variables

A Stage variable is a key-pair value that you can associate with a deployment stage of a REST API. You can think of them as environment variables where you store different parameters or configuration values that your API can access at runtime.

Doing canary releases within the same REST API stage is a great application of stage variables. Canary is a deployment approach in which traffic is split into two parts, one of which carries a larger portion of the traffic and is routed to the production version, while the smaller portion points to the environment to which the new version is deployed. It can be a 90/10 split, 70/30 split, or even a 50/50 split. It all depends on how aggressive you want your deployment to be. This kind of setup allows developers to expose new features or bug fixes to a subset of users and receive feedback from them without having to shut down or make direct changes to the production version.

Consider a REST API stage with a Lambda function as the backend. Assume you're about to release a new version of your API and want to test it with a subset of your users while continuing to serve the majority of traffic with the old version. You can shift all traffic to the new version once you’re satisfied with it. As shown in the diagram below, the goal is to have a single API endpoint that dynamically interacts with two distinct versions of the Lambda function.



This is where stage variables come into play. In the example, we have a Lambda function with two aliases (prod and beta). Rather than writing the actual aliases in your integration backend's settings, we can use the stage variable `ver` as a placeholder. You can then switch between different values of `ver` in the Canary setting of the API Stage and control the amount of traffic that goes to different aliases.

References:
https://docs.aws.amazon.com/apigateway/latest/developerguide/stage-variables.html
https://docs.aws.amazon.com/apigateway/latest/developerguide/canary-release.html

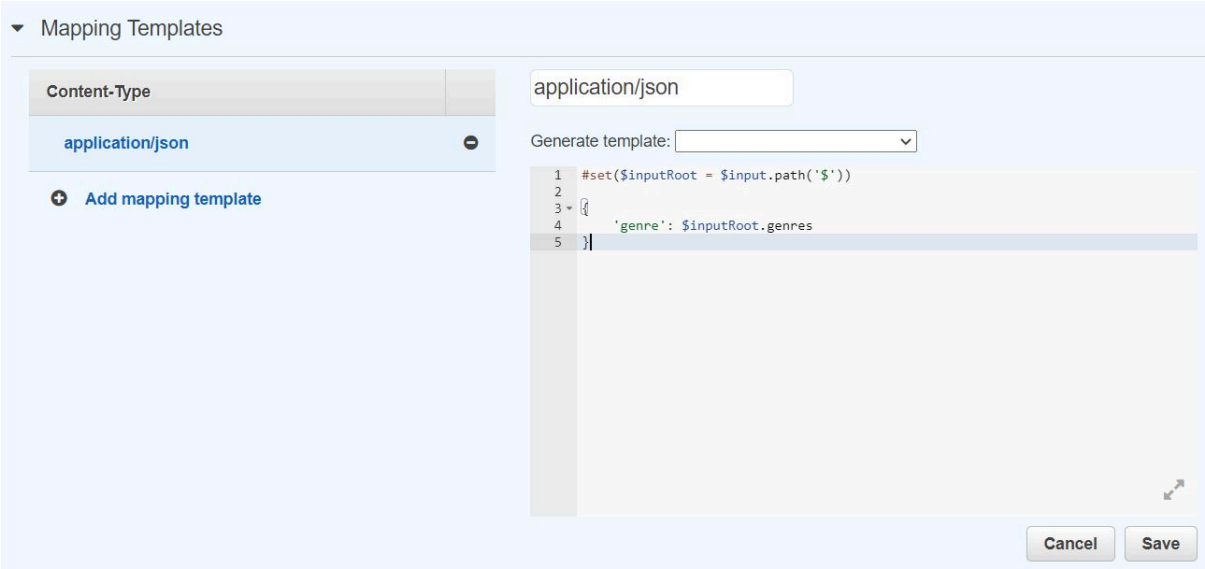
Mapping Templates

Mapping templates allow you to modify any request data before it is forwarded to your integration backend. Conversely, it can be used for transforming the response data before it is returned to the client. Let's further understand how this works with an example.

Say you've built an API for a music application that returns various details about an artist. To keep it simple, consider the following JSON data:

```
{
     "id": 1234,
     "artist": "Queen",
     "popularity": 90,
     "genres": ["Progressive rock", "Pop rock", "Glam rock"]
}
```

If a user wants to obtain the popularity score or the genre of a certain artist, there's no way for him/her to retrieve the information that he/she only cares for. To solve this, you can use mapping templates to modify the response before it is sent back to the user. Assuming that your API has a resource path named `/{id}`, a child resource path called `/genres` can be created under it. In the integration response settings, add a mapping template to construct the new response.



After redeploying the API, users can now send `GET` requests to the `/{id}/genres` resource. In our example, the response would simply be:

```
{
   'genre': ["Progressive rock","Pop rock","Glam rock"]
}
```

Mapping templates are written in the Velocity Template Language or VTL — a Java-based template engine developed by Apache. If you aren't familiar with it, getting comfortable with mapping templates may take you some time. Don't worry, as knowledge of VTL is not required in the CDA exam; you must only understand what a mapping template is and what it is used for at a high level.

Keep in mind that mapping templates **only work for non-proxy integrations**. This is because in proxy integrations, API Gateway simply passes the data it receives to both ends and is unaware of how the request/response is modeled.

You can also use mapping templates to modernize legacy applications. If you have a legacy application that you want to expose, say a SOAP web service that processes XML data, you can have clients send JSON-formatted requests and then have Amazon API Gateway transform that JSON data to XML using mapping templates. As for the integration backend, you can use a Lambda function as middleware to transmit the XML as a payload to the SOAP web service.

Invalidating Cache

Caching often leads to data inconsistency, which is a problem commonly faced. When content is cached, API Gateway does not update the cache entries until the Time-To-Live (TTL) expires. As a result, any changes made to the database will not immediately reflect on the client-side, leading to a disparity between the actual content and what is displayed on the application. However, you can take steps to mitigate this issue by sending an invalidation request to your API endpoint. This will prompt API Gateway to refresh its cache instead of waiting for the TTL to expire.

To invalidate a cache entry, simply include the `Cache-Control` header in a request with a `max-age` of 0, as shown in the example below that uses the Fetch API in Javascript.

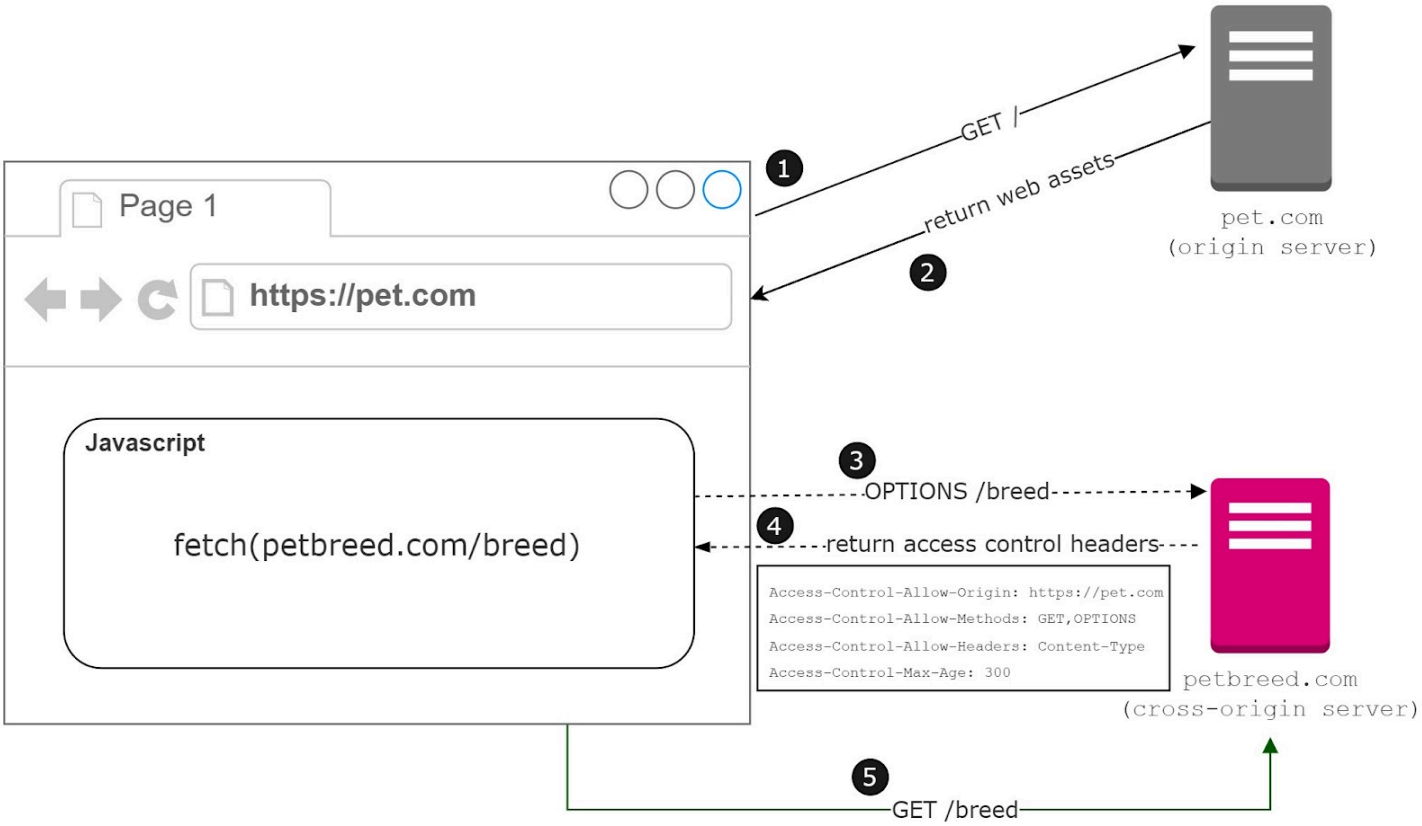
```
fetch('https://aaaa4vb5wf.execute-api.us-east-1.amazonaws.com/v1', {
            method: 'GET',
            headers: {
                'Cache-Control': 'max-age=0'
            }
        });
```

Reference:
https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-caching.html#invalidate-method-caching

Cross-Origin Resource Sharing (CORS)

CORS a security mechanism that most web browsers such as Google Chrome or Mozilla Firefox enforce to relax the restrictions of the same-origin policy. The same-origin policy is a browser security feature that limits scripts loaded from an origin to only interact with resources from the same origin. While the intention is good, sometimes it can be too restrictive. Businesses today usually rely on third-party APIs to quickly add features to their applications. This would not be possible with the Same-Origin Policy in place. To solve this problem, engineers came up with the idea of Cross-origin resource sharing to loosen up the Same-Origin Policy restrictions.

How does CORS work?



Say you visit a website called `pet.com` using Google Chrome. Upon loading, Google Chrome will download the required assets (HTML, Javascript, images, fonts, etc.) from the website's server to render the webpage. As you browse the website, you come across a fun feature that uses Artificial Intelligence (AI) to identify dog or cat breeds based on an image you upload. This feature is powered by a third-party API (`petbreed.com`), which is accessed via a Javascript file. Google Chrome will not immediately send a GET request to `petbreed.com` after you submit an image. Instead, it will first send a preflight OPTIONS request to confirm if `petbreed.com` does indeed allow `pet.com` to make GET requests. At the `petbreed.com` server's end, the allowed domains and methods can be specified through the **`Access-Control-Allow-Origin`** and the **`Access-Control-Request-Method`** headers. The API developer can select which values to put in those headers. For example, `pet.com` can be defined as an `Access-Control-Allow-Origin` header value and `GET` as an `Access-Control-Request-Method` header value. This will explicitly grant `pet.com` to make `GET` requests to `petbreed.com`. Next, `petbreed.com` returns the list of allowed API methods and domains to Google Chrome. If `pet.com` is specified in the `Access-Control-Allow-Origin`, only then will Google Chrome send the actual `GET` request.

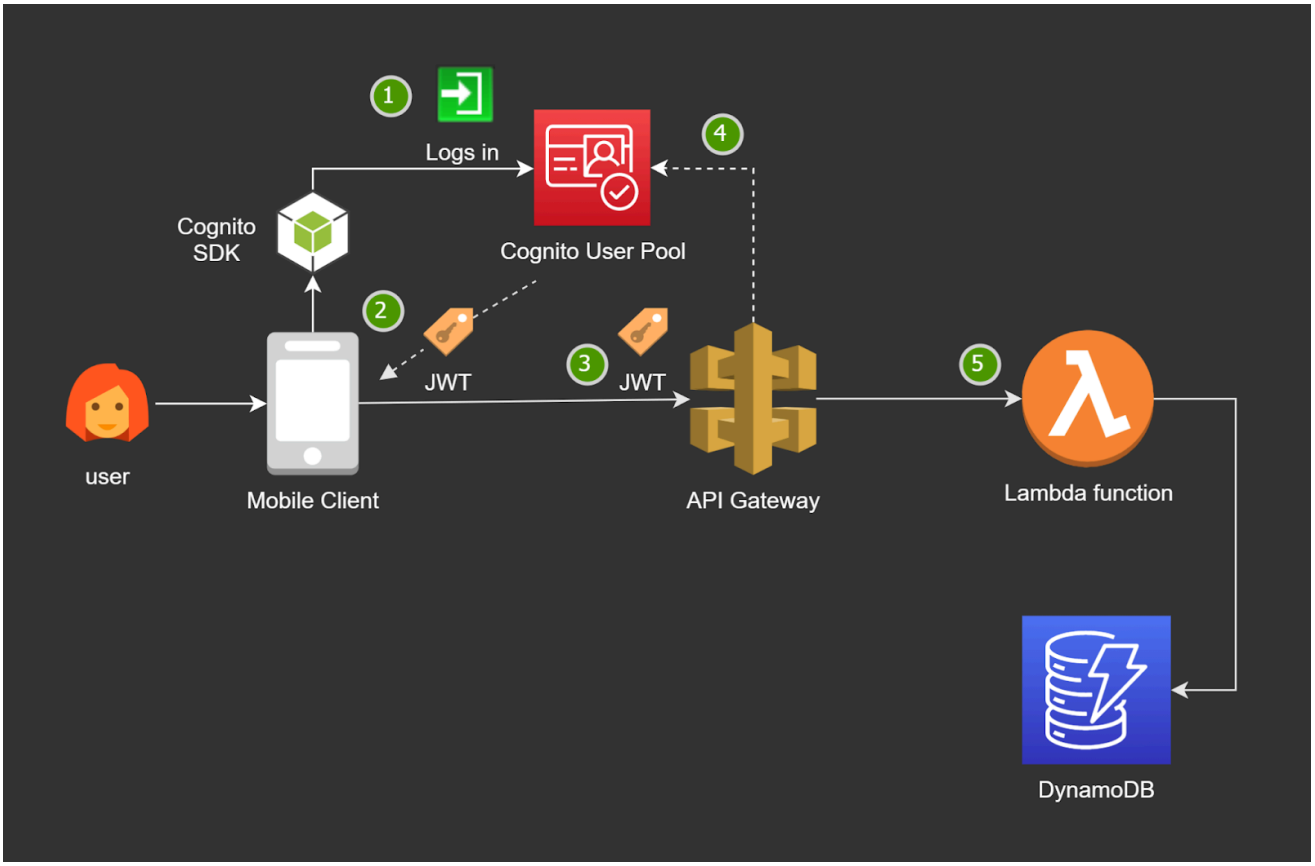
Just like `petbreed.com`, you can also configure CORS in API Gateway. Please note that CORS is disabled by default. CORS is configured at the resource method level when using non-proxy integrations. You must specify the proper access control headers in the header mappings of your API's integration response. On the other hand, if you're using a proxy integration, you must explicitly declare the access control headers in the response returned by your backend.

References:
https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-cors.html
https://aws.amazon.com/blogs/compute/configuring-cors-on-amazon-api-gateway-apis/

Authorizers

API Gateway lets you use Cognito User Pool or a Lambda function to authorize client requests.

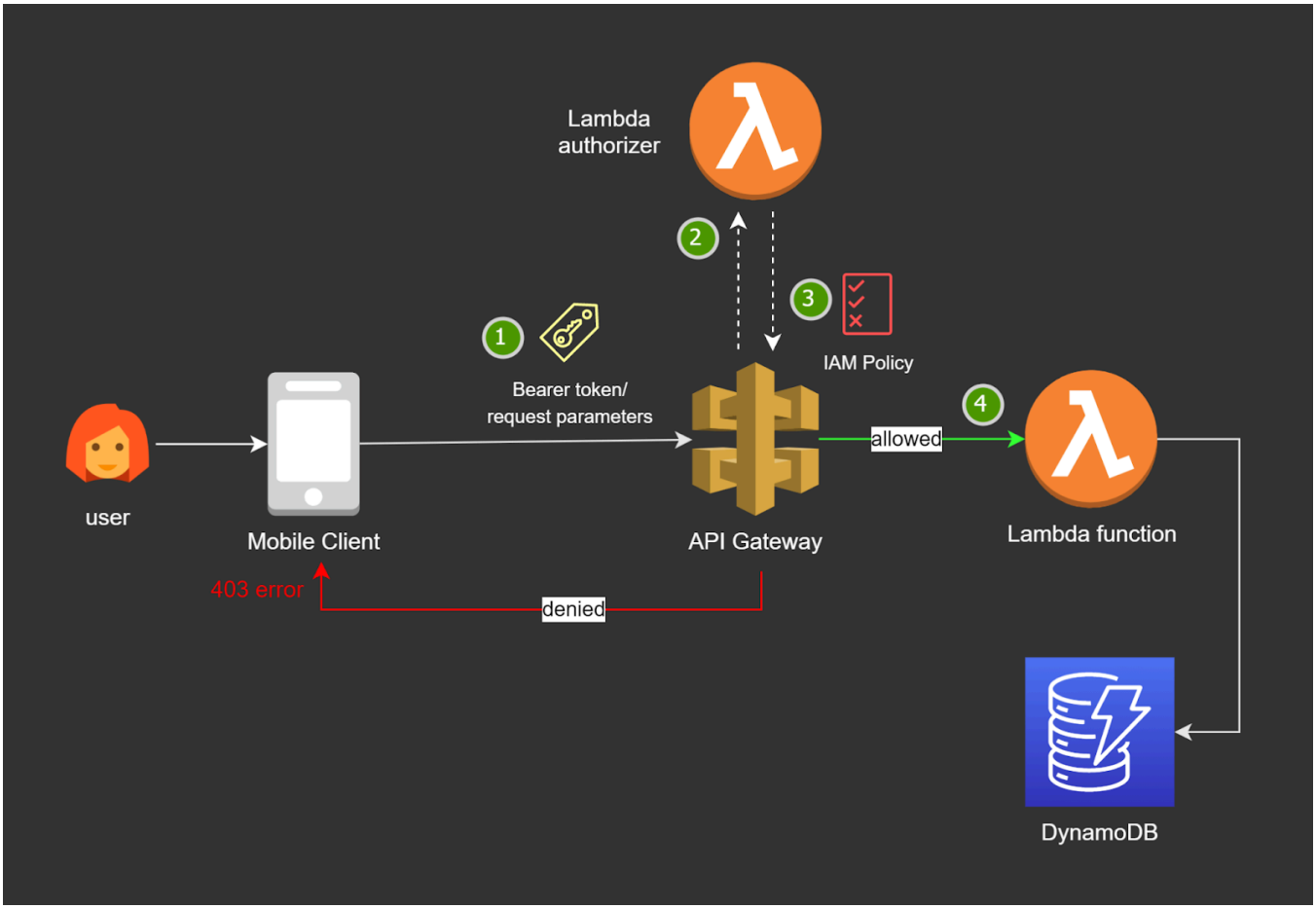
How Cognito User Pool authorizer works:



1. When a user logs in to the User Pool, Cognito checks if the credentials the user has submitted are valid.
2. If the login is successful, Cognito returns a JSON web token (JWT) to the client.
3. The JSON web token (JWT) is passed to a custom header which is included as part of the API request.
4. API Gateway will look for the custom header and verify its validity from Amazon Cognito.
5. Once verified, API Gateway accepts the request and performs the API call. In this case, the Lambda function, which is the backend, is invoked. If there are no issues, API Gateway returns a `200 status code` along with the response from the Lambda function back to the client.

You might want to implement a Lambda Function Authorizer to enforce custom authorization logic, such as those that employ bearer token authentication strategies (OAuth) or something that uses request parameters to determine the caller's identity. Since this is a custom method, you have to write the logic that carries out the authorization process. As a result, this takes more development effort on your end compared to using the Cognito User Pool.

How a Lambda function Authorizer works



1. The application sends a `GET` method to API Gateway, along with a bearer token or request parameters.
2. API Gateway will check whether a Lambda authorizer is enabled for the method. If it is, API Gateway calls the Lambda function that authorizes the request.
3. The Lambda function authenticates the request. If the call succeeds, the Lambda function grants access by returning an output object containing at least an IAM policy and a principal identifier.
4. API Gateway evaluates the policy. If access is denied, API Gateway returns a `403 forbidden status code` and If access is allowed, API Gateway performs the GET request.

Reference:
https://aws.amazon.com/blogs/compute/introducing-custom-authorizers-in-amazon-api-gateway/

Usage Plans

If you have an idea for an API that you'd like to expose or sell, perhaps you've built an AI service that can help out other businesses, then you might want to look at Usage Plans. Usage plans is an API Gateway feature that can help you control different levels of access to an API. For example, you can create three subscription plans for your API: basic, standard, and premium plan. A usage plan can be used to set distinct throttling and quota limitations based on a subscription, which is then enforced on specific client API keys.

Aside from that, you can sell your API as a product in the AWS SaaS Marketplace so other users or companies can see your product and subscribe to it. Your API subscribers are billed based on the number of requests made to the usage plan.

References:
https://aws.amazon.com/blogs/aws/new-usage-plans-for-amazon-api-gateway/
https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-api-usage-plans.html

Amazon Aurora

Amazon Aurora is a fully managed relational database engine compatible with both MySQL and PostgreSQL. It's designed to be both faster and more cost-effective than traditional relational database engines. Aurora is part of Amazon Web Services (AWS) and provides high durability, availability, and security features. Here are some key aspects of Amazon Aurora:

- **Performance and Scalability:** Aurora delivers up to five times the throughput compared to standard MySQL and triples the performance of standard PostgreSQL. It dynamically adjusts your storage requirements as needed, ranging from 10GB to a massive 128TB.

- **High Availability and Durability**: Aurora is built to maintain more than 99.99% uptime. It spreads your data across several AWS Availability Zones and constantly saves it to Amazon S3, ensuring robust data durability.
- **Compatibility:** Aurora seamlessly works with your pre-existing MySQL and PostgreSQL tools and applications, making migrations straightforward.
- Security: Aurora incorporates several security layers, including network isolation with Amazon VPC, data encryption at rest through AWS Key Management Service (KMS), and data encryption in transit with SSL.
- **Automated Backups and Snapshots:** Aurora automatically handles database backups without affecting the performance. You have the flexibility to create snapshots manually as well.
- **Monitoring and Maintenance:** Aurora uses Amazon CloudWatch to track your database's performance. It also automates common database management tasks like patching and backups, reducing your workload.

References:
https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/CHAP_AuroraOverview.html

Amazon DynamoDB

Amazon DynamoDB is a serverless non-relational database. It is called non-relational because it does not follow a fixed tabular schema design like how a relational database would force you to. There are different variations of non-relational databases, such as key-value database, in-memory database, document database, and a graph database. You can think of DynamoDB as a mix of a key-value and document database model. In a key-value database, data is stored as key-value pairs. The key acts as an identifier so DynamoDB would know where in the storage it will locate and retrieve the value associated with the key. The value for a key can be a simple `String` data, a `number`, a `Boolean` value, a binary, a list, or even a complex data structure like a nested JSON document.

DynamoDB provides high-throughput and single-digit latency performance at any scale, so it's great for use cases requiring fast retrieval access, such as high-web traffic or gaming applications. Like any other serverless product in AWS, DynamoDB is fully managed and requires zero administration, so tasks such as patching and updating are no longer your concern.

By default, DynamoDB replicates data across multiple availability zones within a region. It has built-in fault tolerance capability and can automatically adjust capacity based on the volume of requests. Unlike a regular relational database, you don't need to provide a database endpoint, username, or password when connecting to DynamoDB. You simply have to specify the table name and use DynamoDB API commands that correspond to a CRUD operation that you need. Permissions to access DynamoDB are handled by the IAM service.

Core Components

Table

- a collection of related items
- can have zero or more items

Item

- represents a single record that you want to insert into a table
- each item can have one or more attributes

Attribute

- a fundamental data element of an item

Primary keys

1. Partition key (Required)
    - uniquely identifies each item in a table
    - a partition key value is used as an input to the internal hash function in DynamoDB. The output from that hash function determines the partition or the physical internal storage in which the item will be stored or retrieved.
2. Sort key (Optional)
    - gives you additional flexibility when querying data
    - a table with both Partition key and Sort key can have multiple items with similar partition key values given that they have unique sort key values.
    - can sort data in order.

References:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html

Local Secondary Index vs. Global Secondary Index (LSI vs. GSI)

DynamoDB supports two secondary indexes: Global Secondary Index (GSI) and Local Secondary Index (LSI).

A **Global Secondary Index** can have a different partition key and sort key from that of your base table. You can think of it as a secondary table that can have a completely different schema design. A Global Secondary Index is 'global' in the sense that a query on the index can span all of the data in the base table across all partitions.

A **Local Secondary Index** allows you to create an index where you can use a non-key attribute from its base table as a sort key. The data partitions in LSI are automatically organized by DynamoDB using the partition key

of its base table. You only need to specify the sort key and the attributes that you'd like to project. Unlike GSI, the only time that you can create a Local Secondary Index is during the creation of a table. Creating an LSI on an existing table is not possible.

Key differences:

	Global Secondary Index	Local Secondary Index
Key Attributes	A secondary index can have a partition key, sort key, and non-key attributes.	A secondary index can have a sort key and non-key attributes only
Span query	Queries span all data in the base table, across all partitions	Queries are scoped on the partition key of its base table
Index operations	Can be created anytime	Can be created only during the creation of a table
Size restrictions per partition key value	No restriction	Total size of indexed items under a partition key value must not exceed 10GB
Read consistency	Eventual Consistency	Supports both Eventual and Strong Consistency
Provisioned Throughput consumption	Has its own provisioned throughput for read and write activities	Consumes capacity units from its base table
Projected Attributes	You can only request attributes that are projected in the GSI	Requested attributes that are not projected into the LSI are fetched from the base table by DynamoDB automatically

References:
https://aws.amazon.com/blogs/database/how-to-design-amazon-dynamodb-global-secondary-indexes/
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SecondaryIndexes.html
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/LSI.html

Projections

When creating an LSI or GSI you set the list of attributes that you want to project or copy from the base table to the secondary index. The primary keys (partition and sort key) are **always** projected into the secondary index, but you can also specify non-key attributes that will be projected into the index.

Three options in projecting attributes:

1. `KEYS_ONLY` - all items of an index will only contain the base table's primary keys that you set.
2. `INCLUDE` - allow you to choose other non-key attributes that will be included along with the primary keys of your base table.
3. `ALL` - all attributes from your base table will be copied into your secondary index.

Reference:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html

Scan & Query operations

A Query operation performs a direct look-up to specific items you want to search for based on a partition key. It's akin to skimming through the index of a book to determine which page(s) a keyword can be found on. A Scan operation, on the other hand, is analogous to going through the pages one by one to find a certain piece of information. In the case of DynamoDB, a Scan operation will literally read and return every item in a table. You can optionally provide a filter expression when requesting a Scan to return only a subset of items in the table. However, the filtering occurs only after the scan is completed. In effect, you will still be charged for all items read.

In terms of efficiency, cost, and speed, Query easily outperforms Scan, particularly on big table sizes. As the number of your items grows, the slower and costlier a scan would get. As a best practice, use the Query operation along with secondary indexes as much as possible. Scan should only be used when absolutely necessary.

References:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Scan.html
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.html

Calculating the Required Read and Write Capacity Unit for your DynamoDB Table

Read Capacity Unit

On-Demand Mode

When you choose the on-demand mode, DynamoDB instantly accommodates your workloads as they ramp up or down to any previously reached traffic level. If a workload's traffic level hits a new peak, DynamoDB adapts rapidly to accommodate the workload. The requested rate is only limited by the DynamoDB throughput default table limits, but it can be raised upon request.

For on-demand mode tables, you don't need to specify how much read-throughput you expect your application to perform. DynamoDB charges you for the reads that your application performs on your tables in terms of read request units.

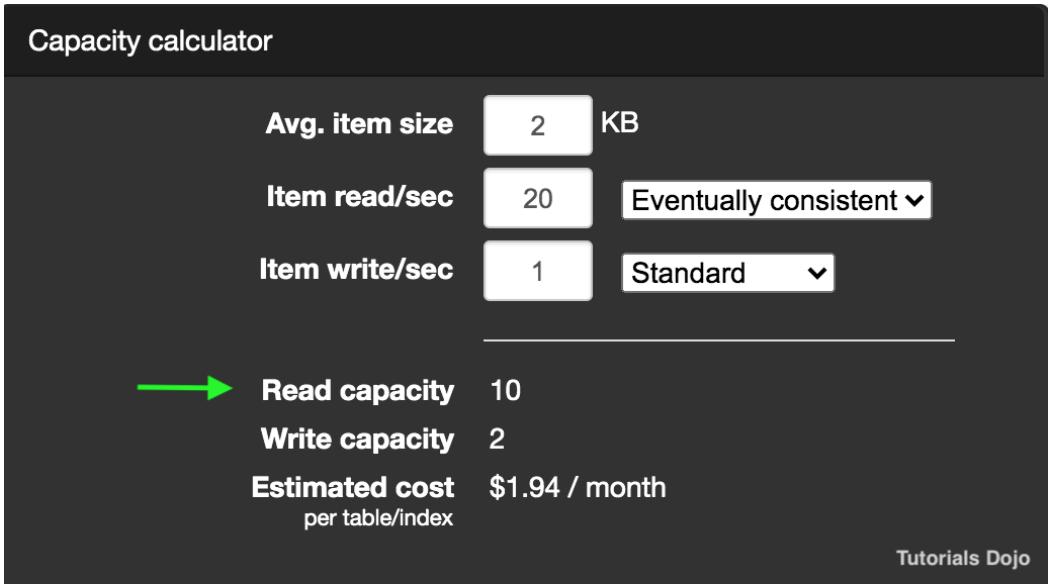
- 1 read request unit (RRU) = 1 strongly consistent read of up to 4 KB/s = 2 eventually consistent reads of up to 4 KB/s per read.
- 2 RRUs = 1 transactional read request (one read per second) for items up to 4 KB.
- For reads on items greater than 4 KB, total number of reads required = (total item size / 4 KB) rounded up.

Provisioned Mode

If you choose provisioned mode, you specify the number of reads and writes per second that you require for your application. You can use auto-scaling to adjust your table's provisioned capacity automatically in response to traffic changes.

For provisioned mode tables, you specify throughput capacity in terms of read capacity units (RCUs).

- 1 read capacity unit (RCU) = 1 strongly consistent read of up to 4 KB/s = 2 eventually consistent reads of up to 4 KB/s per read.
- 2 eventually consistent reads of up to 4 KB/s per read is also equivalent to 1 eventually consistent read of up to 8 KB/s per read.
- To get the *RCU per Item*, you have to divide the average item size by 4 KB (for strong consistency) or 8 KB (for eventual consistency).
- 2 RCUs = 1 transactional read request (one read per second) for items up to 4 KB.
- For reads on items greater than 4 KB, total number of reads required = (total item size / 4 KB) rounded up.



Write Capacity Unit

On-Demand Mode

For on-demand mode tables, you don't need to specify how much write throughput you expect your application to perform. DynamoDB charges you for the writes that your application performs on your tables in terms of write request units.

- 1 write request unit (WRU) = 1 write of up to 1 KB/s.
- 2 WRUs = 1 transactional write request (one write per second) for items up to 1 KB.
- For writes greater than 1 KB, the total number of writes required = (total item size / 1 KB) rounded up

Provisioned Mode

For provisioned mode tables, you specify throughput capacity in terms of write capacity units (WCUs).

- 1 write capacity unit (WCU) = 1 write of up to 1 KB/s.
- 2 WCUs = 1 transactional write request (one write per second) for items up to 1 KB.
- For writes greater than 1 KB, the total number of writes required = (total item size / 1 KB) rounded up.

Example: An application is writing **75** items to a DynamoDB table every second, where each item is **11.5** KB in size. To get the WCU, you just have to follow these three simple steps below:

Step #1 Get the Average Item Size

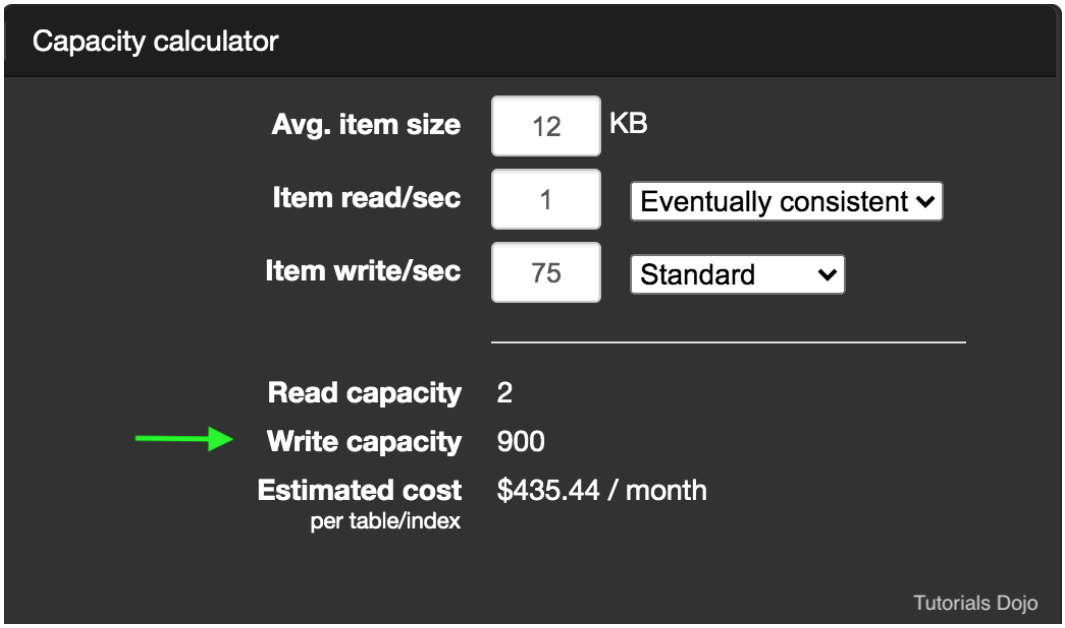
= 11.5 KB = **12 KB** (Rounded up)

Step #2 Get the *WCU per Item* by dividing the Average Item Size by 1 KB

Divide the Average Item Size by 1KB and round up the result:
= 12 KB / 1 KB
= **12 WCU per Item**

Step #3 Multiply the WCU per item to the number of items to be written per second

= **12** WCU per item × **75** writes per second
= **900** WCU



Reference:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadWriteCapacityMode.html

DynamoDB Streams

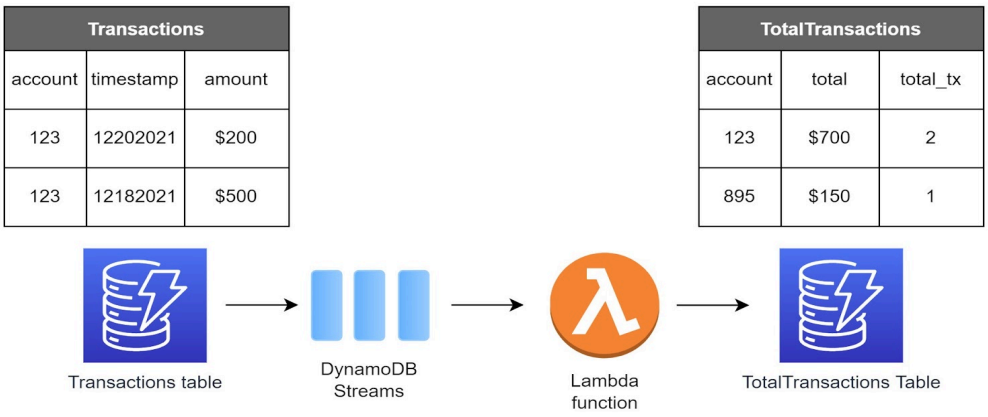
DynamoDB Streams is a feature that captures item-level changes in a DynamoDB table in near-real-time. When an item is modified using any of the write operations (`PutItem, UpdateItem, or DeleteItem`), DynamoDB detects it as an event and sends the modified record to a transaction log. Each stream record

appears exactly once in the log and is retained for 24 hours. Additionally, the order in which items are modified in the table is preserved as they are written in the log.

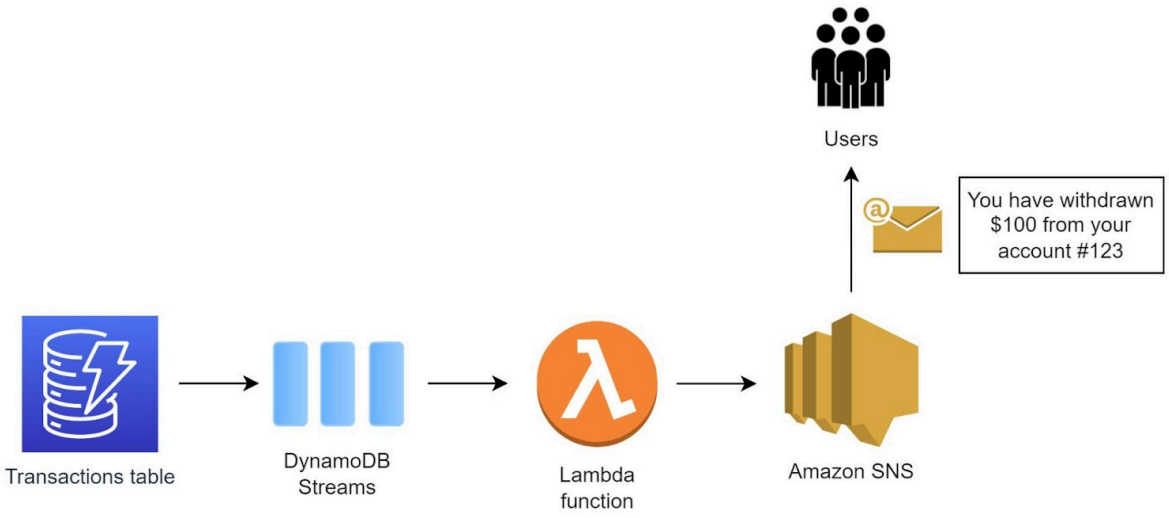
Take note that an actual modification must be made to an item for it to be considered an event. If you send an UPDATE request that does not change anything, DynamoDB simply ignores it, and a record of that event won't be created. In a way, this prevents the generation of duplicate records if ever a change is accidentally applied more than once to an item.

Use cases

1. Aggregation of data for auditing - in the example below, the Transactions table is the source table. When new transactions are inserted into it, new records will fill the DynamoDB streams. The Lambda function reads the records from the stream and increments the `total` and `total_tx` attributes of the `TotalTransactions` table.



2. Notification - in the example below, the Lambda function reads from the DynamoDB Streams, checks the transaction details, constructs a notification message and sends it to an SNS topic. Users subscribed to the SNS topic get alert messages via email regarding their transaction activities.



There are four Stream View Types that you can choose from when using DynamoDB Streams. The Stream View Type determines what kind of data you want to capture.

1. **KEYS_ONLY** - only the key attributes (Partition Key + Sort Key) of the modified item are captured.
2. **NEW_IMAGE** - the latest state of the entire modified item will be captured.
3. **OLD_IMAGES** - the entire item as it appeared prior to the update is captured.
4. **NEW_AND_OLD_IMAGES** - both the latest and the previous state prior to the update is captured.

If you need a more complex streaming application, you might want to configure DynamoDB to send records to a Kinesis Data stream. Doing so will give you access to more advanced features like longer data retention (> 24 hours), built-in functions for processing large amounts of data, and integrations with other Kinesis services (Amazon Data Firehose). Please note that the integration with Kinesis Data Stream is a different feature from DynamoDB Streams

References:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html
https://aws.amazon.com/blogs/database/dynamodb-streams-use-cases-and-design-patterns/
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/kds.html

DynamoDB Accelerator (DAX)

DAX is a highly available, in-memory cache that is purposely built for DynamoDB. A DynamoDB table without any caching layer already gives you response times in the single-digit millisecond range. Now, with DAX in place, you could even reach response times in microseconds for millions of requests per second.

Enabling DAX isn't that complicated. However, the steps are not as straightforward as those for other DynamoDB features like DynamoDB Streams or GSI, where you simply click a button on the AWS Console. To use DAX, you first have to create a DAX cluster. A DAX cluster is simply a group of EC2 instances (also called nodes). DynamoDB is responsible for the management and operation of the cluster. You're the one who will set the size of the cluster, the number of nodes, and their distribution across availability zones. Deploying a single-node DAX cluster to a single AZ is possible, but it's not recommended because such a configuration won't survive a machine or AZ failure. It's important to note that a DAX cluster is housed within a VPC, meaning it is not publicly accessible by default unless you configure it. If you wish to read from a DAX cluster using a Lambda function privately, you must first configure that function to connect to your cluster's VPC.

After creating the DAX cluster, you need to make slight adjustments to your application code. First, you need to download and import the DAX client SDK. Then, point your existing DynamoDB API calls to the DAX cluster endpoint by replacing the existing DynamoDB client with a new DAX client. You don't have to populate the cache or write any cache logic such as lazy-loading or cache invalidation. All of the heavy operations are done for you by the DAX client.

```
import boto3
import amazondax

DAX_endpoint = 'dax://tdojo.aaabb2.dax-clusters.us-east-2.amazonaws.com'

#DynamoDB Client
#dynamodb_client = boto3.client('dynamodb')

#DAX Client
dax_client = amazondax.AmazonDaxClient(endpoint_url=DAX_endpoint)

#GetItem using the DAX Client
response = dax_client.get_item(
    TableName='Music',
    Key={
        'artist': {'S': 'Foo Fighters'},
        'song': {'S': 'Walk'}
    })
```

Use DAX if:

1. You have a read-intensive application that requires response time in the microsecond range.
2. You have a large set of data that are frequently read.
3. Your application can eventually tolerate consistent reads.

Don't use DAX if:

1. Your application requires strongly consistent reads.
2. Your application performs more writes than reads.

References:
https://aws.amazon.com/dynamodb/dax/
https://aws.amazon.com/blogs/database/how-to-increase-performance-while-reducing-costs-by-using-amazon-dynamodb-accelerator-dax-and-aws-lambda/

DynamoDB Transactions

ACID (Atomicity, Consistency, Isolation, Durability) are four database properties that guarantee data integrity and validity even in the event of a power outage or system failure. Most of the relational databases (MySQL, SQL Server, PostgreSQL) that we see today adhere to the principle of ACID, making them suitable for business-critical applications such as those that deal with financial transactions like banking, debit card processing, or order execution. Despite being a non-relational database, you can get the same ACID support for DynamoDB through DynamoDB Transactions.

But first, what is a transaction?

A transaction is nothing but a collection of queries that effectively acts as a single unit of work. To explain this, consider the following example:

Account A

Id	Balance
165	$500

Account B

Id	Balance
785	$800

Above are two simple tables that belong to imaginary bank accounts. Let's say Account A wants to transfer $100 to Account B. Before proceeding, it's only logical that we check first if Account A has sufficient balance to support the transaction. DynamoDB has a conditional operation that we can use to verify if the balance of Account A is greater than $100. If it is, then we proceed to the next step. Otherwise, we abort the transfer. The second step would be to subtract $100 from Account A, making its new balance to $400. The third and final step is to add the $100 to Account B, making its new balance to $900. Pretty simple right? In real-world scenarios, CRUD operations are actually not executed individually, especially for critical workloads such as financial transactions.

In our example, we needed three sequentially executed operations to complete the transfer. A problem emerges if a failure occurs during the third step. $100 would be deducted from Account A, but you wouldn't see the that amount added to Account B.

With DynamoDB Transactions, you can wrap those three individual operations into a single function. This function equates to 1 transaction. As a result, if one of the operations fails, the transaction as a whole also fails. The condition for the transaction to succeed is quite simple. All three operations must succeed — it's all or nothing.

DynamoDB provides two transaction operations:

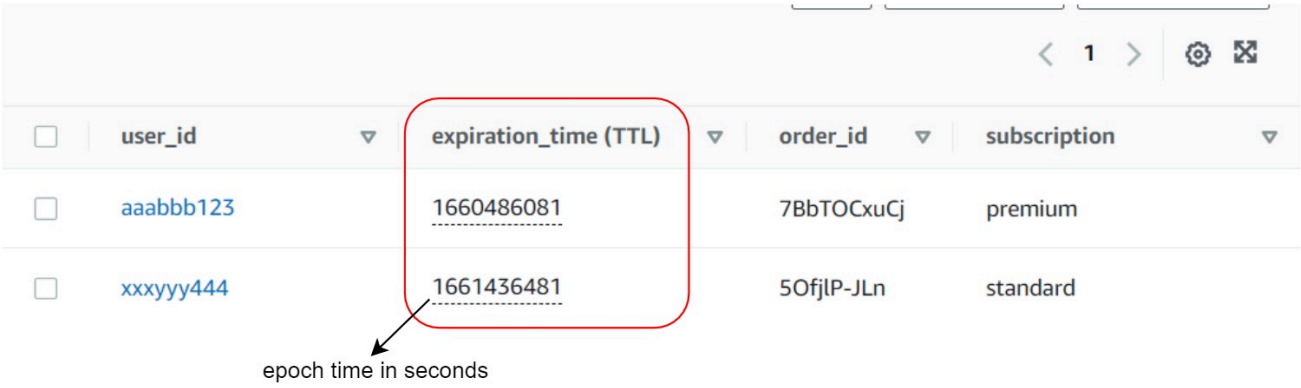
1. `TransactWriteItems` - you can group up to 100 write actions in a single all-or-nothing operation. The actions that you can add are the `PutItem`, `UpdateItem`, `DeleteItem`, and `ConditionCheck` operations.
2. `TransactGetItems` - contains a read set, with one or more GetItem operations. Each transaction can include up to 100 unique Get items or up to 4 MB of data, including conditions.

Transactions are 2 times more expensive than regular reads or writes because DynamoDB needs to perform two underlying reads/writes of every item; one to prepare the transaction and one to commit the transaction.

References:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/transaction-apis.html
https://aws.amazon.com/blogs/aws/new-amazon-dynamodb-transactions/
https://aws.amazon.com/blogs/aws/new-amazon-dynamodb-transactions/

DynamoDB Time-To-Live (TTL)

Time-To-Live or TTL allows you to automatically expire an item based on a defined timestamp. TTL can be a powerful tool for cleaning up your DynamoDB table to keep its size small and manageable over time. Table size directly affects the cost of storage and running queries, so having fewer data to read and store means more cost savings for you.



TTL is also helpful in situations wherein you only need to keep data for a certain period of time, and after that, you get rid of them. For example, if you have an application that offers access to a time-limited service, instead of coding the logic to expire user access, it might be easier for you to just enable TTL and let DynamoDB handle it for you.

How to use TTL?

1. Enable TTL via the DynamoDB Console or using the `UpdateTimeToLive` API.

2. Assign the name of the TTL attribute that DynamoDB will look for in determining if an item is eligible for expiration. You can pick any name for a TTL attribute, but it should be the same for all items that you want to remove eventually because TTL applies at a table level.

    - The TTL attribute must be a `Number` data type, and its value must be in Unix epoch time format.

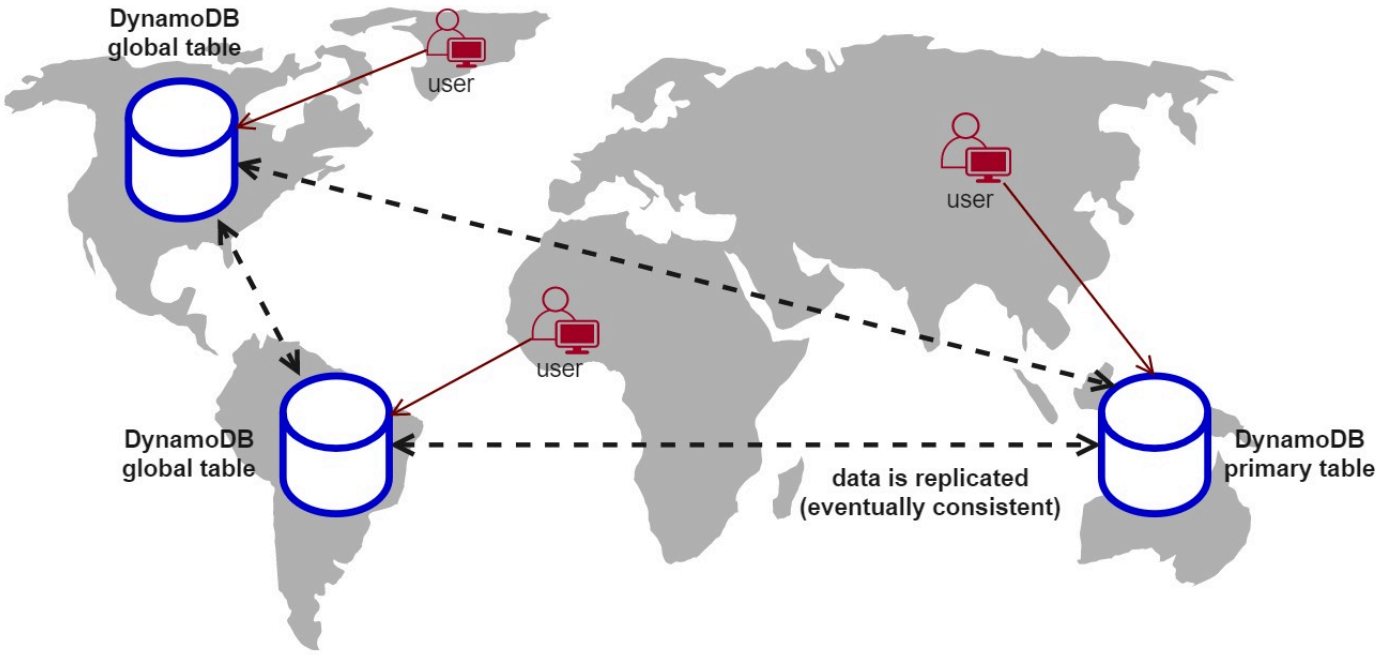
    - Items are not deleted instantly at the moment the TTL expires. TTL has a delay that increases as your table size expands. According to official documentation, DynamoDB deletes expired items within 48 hours after expiration, something that you might take into consideration when designing an application.

References:
https://aws.amazon.com/blogs/aws/new-manage-dynamodb-items-using-time-to-live-ttl/

https://aws.amazon.com/blogs/aws/new-manage-dynamodb-items-using-time-to-live-ttl/

DynamoDB Global Tables

DynamoDB Global Tables allows you to automatically replicate your DynamoDB table across AWS Regions of your choice. If you're leaning towards a multi-region architecture for an application, the advantage for you as a developer is that you no longer have to supervise the replication of data yourself. DynamoDB uses DynamoDB Streams to manage the synchronization of tables. Hence, before you can use Global tables, you have to enable DynamoDB Streams with the `NEW_AND_OLD_IMAGES` `StreamViewType`.



Why use Global Tables?

1. **To protect your application from regional failures** - DynamoDB Global Tables can be a part of your disaster recovery plan as it could help you mitigate the impact of AWS regions outages. If one region experiences some issues that affect your application, you can always shift the traffic of affected users to a working region.

- *The redirection of traffic to tables in other regions is not done automatically by DynamoDB. You still need to implement the failover logic in your application.*

2. **To improve the performance of latency-sensitive applications** - if you have a latency-sensitive application with global reach, you'd want to keep latency as low as possible regardless of where your users are located. Latency is largely affected by distance. Imagine having a DynamoDB table in Singapore (ap-southeast-1). Those in Singapore and neighboring countries will have faster response times than users in, say, the United States. This is due to people being geographically closer to their data. Hence, it's important to bring data as close to your users as possible.

References:
https://aws.amazon.com/dynamodb/global-tables/
https://aws.amazon.com/blogs/database/how-to-use-amazon-dynamodb-global-tables-to-power-multiregion-architectures/

AWS CloudFormation

Spinning up AWS resources through the AWS console may not seem like a big deal if you're used to the process; regardless, it can still take a lot of time, especially if you have to set up a large environment with many components. Not to mention, manual processes are prone to human errors. Rather than manually launching the resource, you can describe them in a text file, making it much easier to create predictable environments. With CloudFormation, you can model your entire cloud environment in a CloudFormation template using either JSON or YAML. To create an environment, simply upload your CloudFormation template to Amazon S3. CloudFormation will take the template from S3 and create the resources specified in it.

Stack

The individual resources that CloudFormation creates are treated as a single unit called *'stack'*. By default, if one of the resources fails to be created during stack creation, the stack will revert to its last working state. If there is no last working state, all resources are terminated, including those that were successfully created prior to the failure.

Reference:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacks.html

Parts of CloudFormation Template

- `AWSTemplateFormatVersion` - lets you set the version of the template that you want to use, which determines the capabilities of the template. If you don't assign a version, CloudFormation automatically assumes that you want to use the latest version.

- `Description` - allows you to include comments about your template to help you better describe and add more details about your template.

- `Metadata` - allows you to provide objects that provide details about the template or even implementation details of specific resources.

- `Resources` (**REQUIRED**) - This is where you define all resources that you want to include in your stack. AWS resources such as EC2 instance, Lambda function, S3 bucket are declared in the Resources section. Take note that the Resources section is the only REQUIRED section in a CloudFormation template. All other sections are optional.

- `Mappings` - matches a key to a corresponding set of named values.

- `Parameters` - when writing CloudFormation templates, it's very unlikely to be hardcoding values such as AMI ID, security group ID, and so on. Parameters enable users to pass values dynamically based on their use case.

- `Conditions` - defines conditions by using the intrinsic condition functions. These conditions determine when AWS CloudFormation creates the associated resources.

- `Transform` - you can use this section to write templates in the AWS Serverless Application Model (AWS SAM) syntax. The syntax used in SAM is different from the standard CloudFormation template. Behind the scenes, CloudFormation converts and 'transforms' the templates written under the Transform section into a regular CloudFormation template.

References:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-anatomy.html
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/gettingstarted.templatebasics.html

Intrinsic Functions and Pseudo Parameters

CloudFormation offers several built-in functions and predefined pseudo parameters that you can use to create dynamic and flexible templates. In this section, We'll go over some of the most common ones and see how they're actually used.

1. `!Ref` - returns the value of a parameter or resource. When you specify a resource's logical name, it outputs a value that you can use to refer to that resource (e.g., instance id, bucket name, API ID). Usually, in a stack, there'll be dependencies between resources. For example, if you want to attach an Elastic IP address (EIP) to an EC2 instance, you must specify the instance using its instance id. The thing is, the instance id won't be available until the EC2 instance is created, so hardcoding it in the CloudFormation template is not possible. To get around this, we can use the Ref function to retrieve the instance id once it's available and have CloudFormation insert it during stack creation.

```
Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-0cd490fb43e2559ed
      InstanceType: t2.micro
  MyEIP:
    Type: AWS::EC2::EIP
    Properties:
      InstanceId: !Ref MyEC2Instance
```

2. **`!FindInMap`** - this one only works with the Mappings section. `FindInMap` returns a named value based on a specified key. For example, suppose you have custom AMIs in us-east-1 and us-east-2. And you want to make sure their image ids are accessible at runtime regardless of the region the stack is deployed. First, create a two-level map under the Mappings section. Each map is a key that points to a name-value pair. Make the regions your top-level keys, followed by a name-value pair that represents the AMI name and its corresponding image id. Retrieve the image ids with FindInMap using the syntax below:

   !FindInMap [MapName, TopLevelKey, SecondLevelKey]

```
Mappings:
  MyRegions:
    us-east-1:
      CustomUbuntu: ami-0cd490fb43e2559ed
    us-east-2:
      CustomUbuntu: ami-01581ffba5551cdf3

Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: !FindInMap [ MyRegions, !Ref "AWS::Region", CustomUbuntu ]
      InstanceType: t2.micro
```

In the snippet above, `MyRegions` is the map name. For the top-level key, the `AWS::Region` pseudo parameter is used instead of a static value to grab the Region where the stack will be deployed. Lastly, the `CustomUbuntu`

key maps to the actual image id. This way, if a user chooses to deploy the stack in either us-east-1 or us-east-2, there won't be any problem.

3. **`!GetAtt`** - returns the value of an attribute from a resource. This intrinsic function works similar to how the !Ref function works. Except that this time, there are more values to choose from depending on the resource.

   Syntax for the function name (shorthand):

   ```
   !GetAtt logicalNameOfResource.attributeName
   ```

   The AWS::EC2::Instance resource contains attributes such as the availability zone where the specified instance is launched, its private and public IP address, and public DNS name. If you want to retrieve the public IP of the MyEC2Instance resource, use the `!GetAtt MyEC2Instance.PublicIp` function. Always check the documentation for the attributes available to a particular resource.

References:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference.html
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/pseudo-parameter-reference.html

Using dynamic references to securely pass credentials

Dynamic references allows you to reference external values in your stack template. These values can be things like configuration parameters, database connection strings, or secrets such as passwords or API keys that are stored in Systems Manager Parameter Store and AWS Secrets Manager. With dynamic references, you don't need to hardcode credentials into the parameter section of your stack template. Instead, you can use placeholders to reference them. When you deploy the stack, CloudFormation retrieves the actual values from the external service and substitutes them into the placeholders. This prevents secrets from being exposed in the CloudFormation stack's resources or logs.

CloudFormation supports the following reference key names:

1. `ssm` (SSM Parameter Store plaintext parameter)
2. `ssm-secure` (SSM Parameter Store secure string parameter)
3. `secretsmanager` (Secrets Manager secret)

Suppose you are building a stack for an application that is made up of a Lambda function and an RDS database. To protect the database credentials from being exposed, you shouldn't store them in plain text in the template, function code, or the function's environment variables. Instead, you can store the credentials as a secret in AWS Secrets Manager and then reference it in your template via the built-in `secretsmanager` reference key.

```
MyRDSInstance:
    Type: 'AWS::RDS::DBInstance'
    Properties:
      DBName: MyRDSInstance
      AllocatedStorage: '20'
      DBInstanceClass: db.t2.micro
      Engine: mysql
      MasterUsername: '{{resolve:secretsmanager:DBcreds:SecretString:username}}'
      MasterUserPassword: '{{resolve:secretsmanager:DBcreds:SecretString:password}}'
```

You can pass the secret name (in this example, "DBcreds") as an environment variable to the Lambda function.

```
AWSTemplateFormatVersion: '2010-09-09'
Resources:
  MyLambdaFunction:
    Type: 'AWS::Lambda::Function'
    Properties:
      Code:
        S3Bucket: name-of-your-s3-bucket
        S3Key: my-lambda-function.zip
      Handler: index.handler
      Role: !GetAtt MyLambdaExecutionRole.Arn
      Runtime: python3.9
      Environment:
        Variables:
          SECRET_NAME: 'DBCreds'
```

The Lambda function can then retrieve the secret value from AWS Secrets Manager at runtime using the `GetSecretValue` API.

```
import json
import boto3
import os

def lambda_handler(event, context):
    # Get the name of the secret from an environment variable
    secret_name = os.environ['SECRET_NAME']

    # Create a Secrets Manager client
    secretsmanager = boto3.client('secretsmanager')

    # Get the username and password from the secret
    secret_value = secretsmanager.get_secret_value(SecretId=secret_name)
    secret_dict = json.loads(secret_value['SecretString'])
    username = secret_dict['username']
    password = secret_dict['password']
```

References:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/dynamic-references.html
https://aws.amazon.com/premiumsupport/knowledge-center/cloudformation-stack-dynamic-parameters/

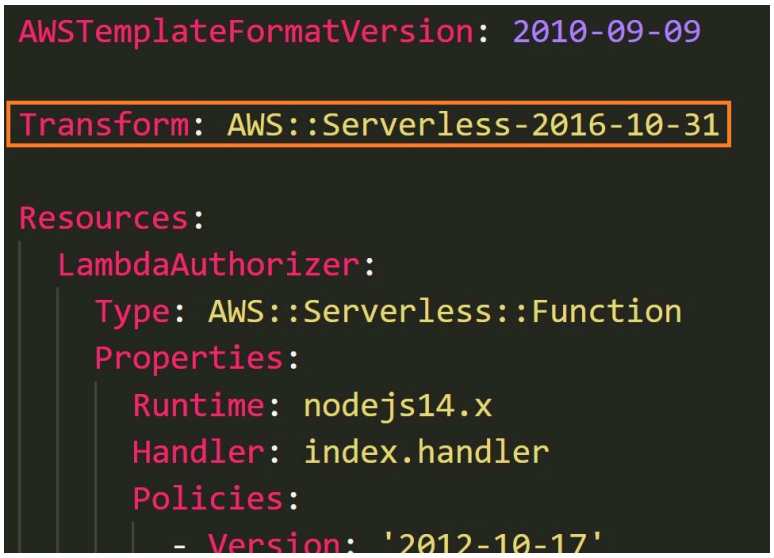
AWS Serverless Application Model (SAM)

The AWS Serverless Application Model, or AWS SAM for short, is an open-source framework for building serverless applications. SAM is divided into two parts: the SAM template and the SAM Command Line Interface (CLI).

SAM Template

SAM templates give you access to custom resource types, which are basically wrappers for the existing ones available in CloudFormation. These custom resources types enable you to construct serverless applications in fewer lines of code. The SAM resource `AWS::Serverless::Function`, for example, does more than just create a Lambda function. This resource type can generate a Lambda function, an IAM execution role, an event source, and connect them all together. As a result, the development time is significantly reduced.

Before you can access the custom resources in SAM, you must declare the **`AWS::Serverless transform`** macro in the Transform section of a CloudFormation template.



When you deploy a SAM template, CloudFormation will look first for the Transform section, and if there is one, all resources defined under that are expanded into their native CloudFormation resources. SAM has limited options in terms of resource types. It's very specific to serverless resources such as Lambda functions, API Gateway stages, Amazon S3 buckets, and DynamDB tables.

SAM resources can be written alongside the native CloudFormation resources. So if ever there's a need for you to create resources not available in SAM, like an EC2 instance, you can declare them in their original CloudFormation syntax. You can also use intrinsic functions and pseudo parameters in a SAM template just like you would in a native CloudFormation template.

References:
https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/sam-specification.html
https://aws.amazon.com/blogs/mt/build-deploy-serverless-app-using-aws-serverless-application-model-and-aws-cloudformation/

Commonly used SAM CLI commands

1. `sam init` - generates pre-configured AWS SAM templates.
2. `sam package` - bundles your application code and dependencies into a .zip file (deployment package). The .zip file is then uploaded to the S3 bucket that you specify. After that, it returns a copy of your AWS SAM template, replacing references to local artifacts with the Amazon S3 location where the artifacts are uploaded. Take note that the `sam package` is a legacy command. You may not need to run this because its functionality is already built into the `sam deploy` command. However, you might still get questions about `sam package`, so it's still good to know what it is and what it does.

3. `sam deploy` - bundles your application code into a deployment package and deploy it as a SAM application.
4. `sam local invoke` - allows you to test Lambda functions and SAM-based serverless applications locally in a Lambda-like execution environment.

Reference:
https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-reference.html

AWS Step Functions

Step Functions lets you orchestrate the components of an application into a serverless workflow. A **workflow** is a series of steps with the output of one step acting as input into the next step. A workflow defines the process of how your application accomplishes its goal from start to end.

In Step Functions, we refer to the workflow as **State machines**, and each step in a State machine is called a **state**.

A State Machine is written using the Amazon States Language or ASL, which has a similar syntax to a JSON document. If you don't want to write in ASL, Step Functions also provide a drag-and-drop interface that further simplifies how you build your state machine. Step Functions will write the state machine definition for you.

States

There are eight state types that you can use to build a State machine:

- **Task state** - represents a single unit of work in your State machine. Application code that needs to be executed by your Lambda function is done by the Task state. A Task state can only include a single Lambda function, an ECS task, SNS notification, DynamoDB action, etc.

- **Choice state** - adds a branching logic to your workflow. You can think of it as an if-then-else statement where there is more than one possible outcome that can occur when evaluating a parameter against a condition that you set.

- **Parallel state** - performs two branches of execution at the same time. A use case for this is when you want to run two independent activities that perform different processes to an input from the same node.

- **Wait state** - adds a delay to your State machine before it continues executing states. This is similar to how a Sleep function in Python works.

- **Fail State** - stops the execution of the state machine and marks it as a failure. One example where you could use this is when you have a registration process and you want to deny any invalid form that a user submitted. You can mark the registration as a failure and have a custom error message like 'InvalidUsername'.

- **Succeed State -** stops the execution successfully.

- **Pass state -** can use the Pass state to pass the input from the previous state to its output without performing work. This is useful when you're debugging a portion of your state machine and you want to have visibility of the output of a particular state.

References:
https://docs.aws.amazon.com/step-functions/latest/dg/concepts-states.html
https://aws.amazon.com/blogs/aws/new-aws-step-functions-workflow-studio-a-low-code-visual-tool-for-building-state-machines/

Input and Output Processing

When you submit a Step Function execution request, you provide input in the form of a JSON to your workflow's initial state. The state consumes the input data and forwards whatever the output is to the next state, which then becomes its input. To get the most out of the Step Function, one should take time to understand how data flows from one state to another. In addition to this, knowing the specific input that each state actually needs is critical in designing an effective and good workflow. Recognizing what data each state requires is only half of the solution; controlling it is the other half. Step Functions simplifies this part for us.

Step Function has an optional feature that allows you to manipulate and filter data as it moves in and out of a state. It is useful when you have a large chunk of nested data but only want a portion of it for a specific state to work on. For example, let's say you have an image processing state machine that is triggered by an S3 event.

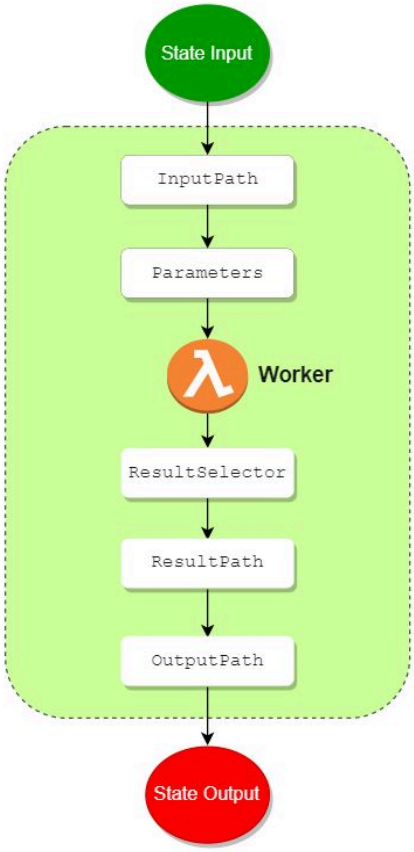
The following JSON data will be passed to the first state of your state machine. Please keep in mind that the following data has been truncated for clarity:

```
{
  "version": "0",
  "id": "3e4bfd68-4504-fee5-d557-fee0e690feac",
  "detail-type": "AWS API Call via CloudTrail",
```

```
  "source": "aws.s3",
  "account": "123456789012",
  "time": "2018-12-18T00:23:05Z",
  "region": "us-east-1",
  "eventTime": "2018-12-18T00:23:05Z",
  "eventSource": "s3.amazonaws.com",
  "eventName": "PutObject",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "0.0.0.0",
   "requestParameters": {
      "bucketName": "media-tutorialsdojo",
      "key": "cat.jpg"
    }
}
```

Let's say that the first state is a Lambda function and it needs to know the S3 path of `cat.jpg`. Knowing this, discarding all unnecessary information except for the `bucketName` and `key` will be your initial goal. *"But can we just parse the event data inside the Lambda function?"* you might ask. Step Functions is designed to make it easier to stitch together various parts of your application. We want each state to focus only on doing the task that it was intended to with less overhead as possible. And one of the overheads we aim to remove is data parsing.

We can use the following filters to control the data as it moves through different states.



Input Filters:

InputPath

- Selects a portion of the JSON state input.
- Success and Fail states are not supported.

Example:

Original input:

```
{
  "message": "hello world",
  "from": "carlo",
  "timestamp": 1639078426
}
```

JSONPath expressions are used to extract key values from a JSON text. Use the `$.` prefix followed by the key name.

*To get the value of the `message` key, we use the **`$.message`** expression:*

```
"hello world"
```

Parameters

- allows you to construct a new JSON payload that is passed as an input to a state.
- Only Map, Parallel, and Pass states are supported.

Example:

Original input:

```
{
  "message": "hello world",
  "from": "carlo",
  "timestamp": 1639078426
}
```

Creating a new JSON input using Parameters:

```
{
  "user.$": "$.from",
  "newMessage": "test123"
}
```

The values can be either static or derived from the original JSON input. When referencing key-value pairs from a JSON input, use `.$` at the end of the key and the start of the value.

Result:

```
{
  "user": "carlo",
  "newMessage": "test123"
}
```

We can use the following filters to control the output of a state.

Output Filters:

ResultSelector

- used to build a new JSON object based on a state's result.
- Only Task, Map, Parallel states are supported.

Example:

Original result:

```
{
  "ExecutedVersion": "$LATEST",
  "Payload": {
    "statusCode": "200",
    "body": "Hello!"
  },
  "SdkHttpMetadata": {
    "HttpHeaders": {
      "Connection": "keep-alive",
      "Content-Length": "43",
      "Content-Type": "application/json",
      "Date": "Thu, 16 Apr 2022 17:58:15 GMT",
        },
    "HttpStatusCode": 200
  },
  "SdkResponseMetadata": {
    "RequestId": "88fba57b-adbe-467f-abf4-daca36fc9028"
  },
  "StatusCode": 200}
```

Applying `ResultSelector`:

```
{
  "newPayload": {
      "body.$": "$.Payload.body",
      "statusCode.$": "$.Payload.statusCode"
  }
}
```

Result:

```
{
  "newPayload": {
      "body": "Hello!",
      "statusCode": "200"
  }
}
```

ResultPath

- used to include the original input to the state's output.
- Only Task, Map, Parallel, and Pass states are supported.

Example:

Before output filter:

```
{
  "newPayload": {
      "body": "Hello!",
      "statusCode": "200"
  }
}
```

If `ResultPath` is left blank or `$`, the result becomes the output, and the state input is ignored, otherwise, the state input is included to the result.

Applying the `$.StateResult` path expression:

```
{
  "message": "hello world",
  "from": "carlo",
  "timestamp": 1639078426,
  "StateResult": {
    "newPayload": {
      "body": "Hello!",
      "statusCode": "200"
    }
  }
}
```

*The yellow text represents the original input.

OutputPath

- Filters the final result before it is sent to the next state.
- Success and Fail states are not supported.

Example:

Before output filter:

```
{
  "message": "hello world",
  "from": "carlo",
  "timestamp": 1639078426,
  "StateResult": {
    "newPayload": {
      "body": "Hello!",
      "statusCode": "200"
    }
  }
}
```

Applying the `$.StateResult.newPayload` expression:

```
{
      "body": "hello, world!",
      "statusCode": "200"
}
```

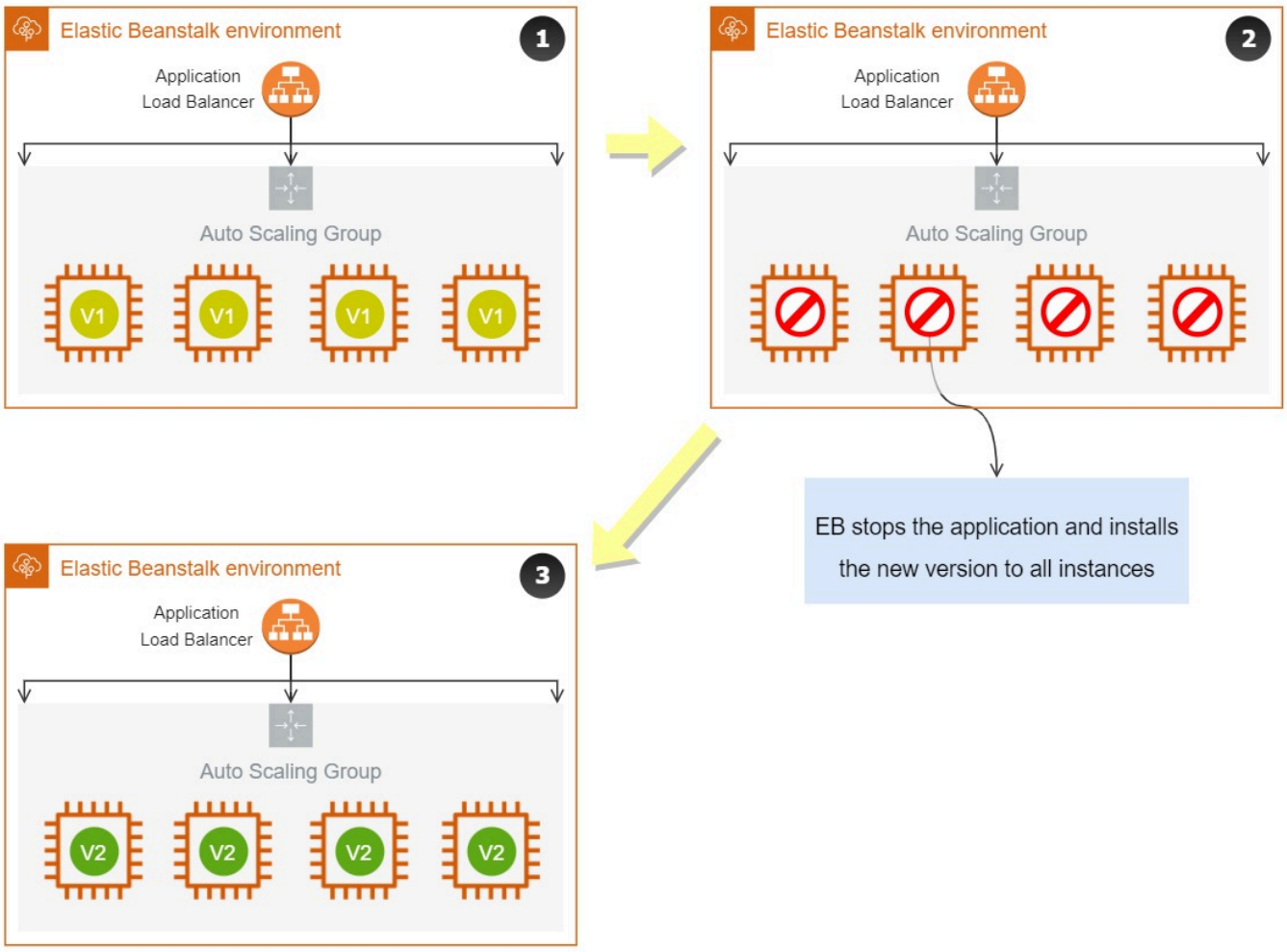
References:
https://docs.aws.amazon.com/step-functions/latest/dg/concepts-input-output-filtering.html
https://aws.amazon.com/blogs/compute/modeling-workflow-input-output-path-processing-with-data-flow-simulator/

AWS ElasticBeanstalk

AWS Elastic Beanstalk is a managed platform that allows you to upload your application code in AWS and provision the required cloud environment easily. You only need to upload your application package that is written in Java, .NET, PHP, Node.js, Python, Ruby, Go, or Docker, and then Elastic Beanstalk will deploy the necessary resources to run your application. You can either run a **Web Server environment** or a **Worker environment**. A Web Server environment runs a static website, a web app, or a web API that serves HTTP requests, while a worker environment, on the other hand, runs a worker application that processes long-running workloads on demand. The latter also performs tasks on a schedule that you define and can be integrated with the Amazon SQS queue.

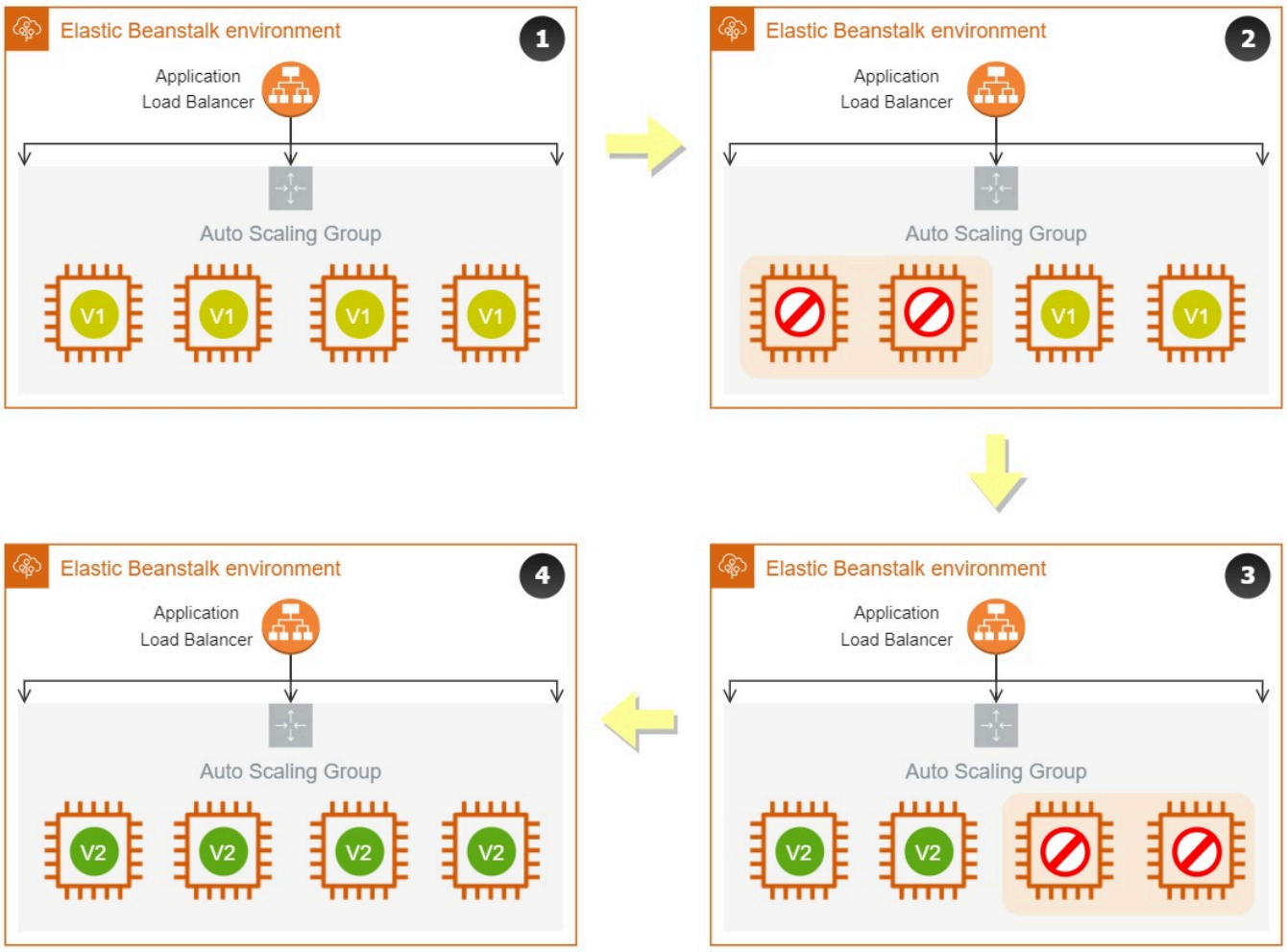
Deployment policies

1. **All at once**



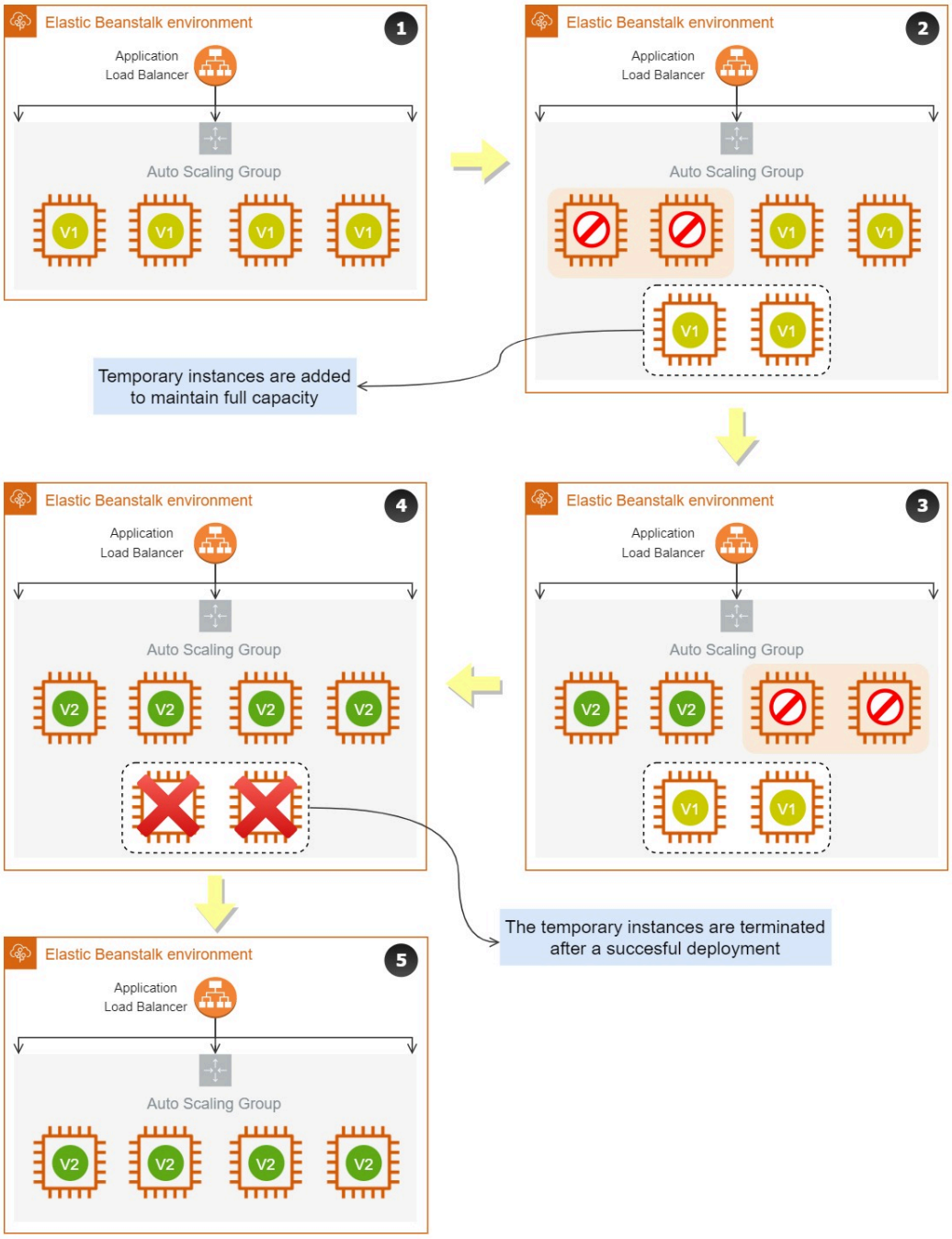
- The latest version is rolled out to all instances in your Elastic Beanstalk environment at the same time.
- This is the fastest method of deployment, making it ideal for test environments where iterative releases are made and reviewed as quickly as possible.
- During the deployment, your application becomes unavailable for a period of time, depending on how long it takes to deploy your new code successfully. Hence, this deployment isn't typically seen in production.

2. Rolling



- The latest version is deployed in batches.
- The impact of failed deployment is lower compared to All at once since rollbacks are applied in batches as well.
- Unlike All out once, this deployment policy does not have any downtime as long as the reduced capacity can serve the volume of traffic during deployment.

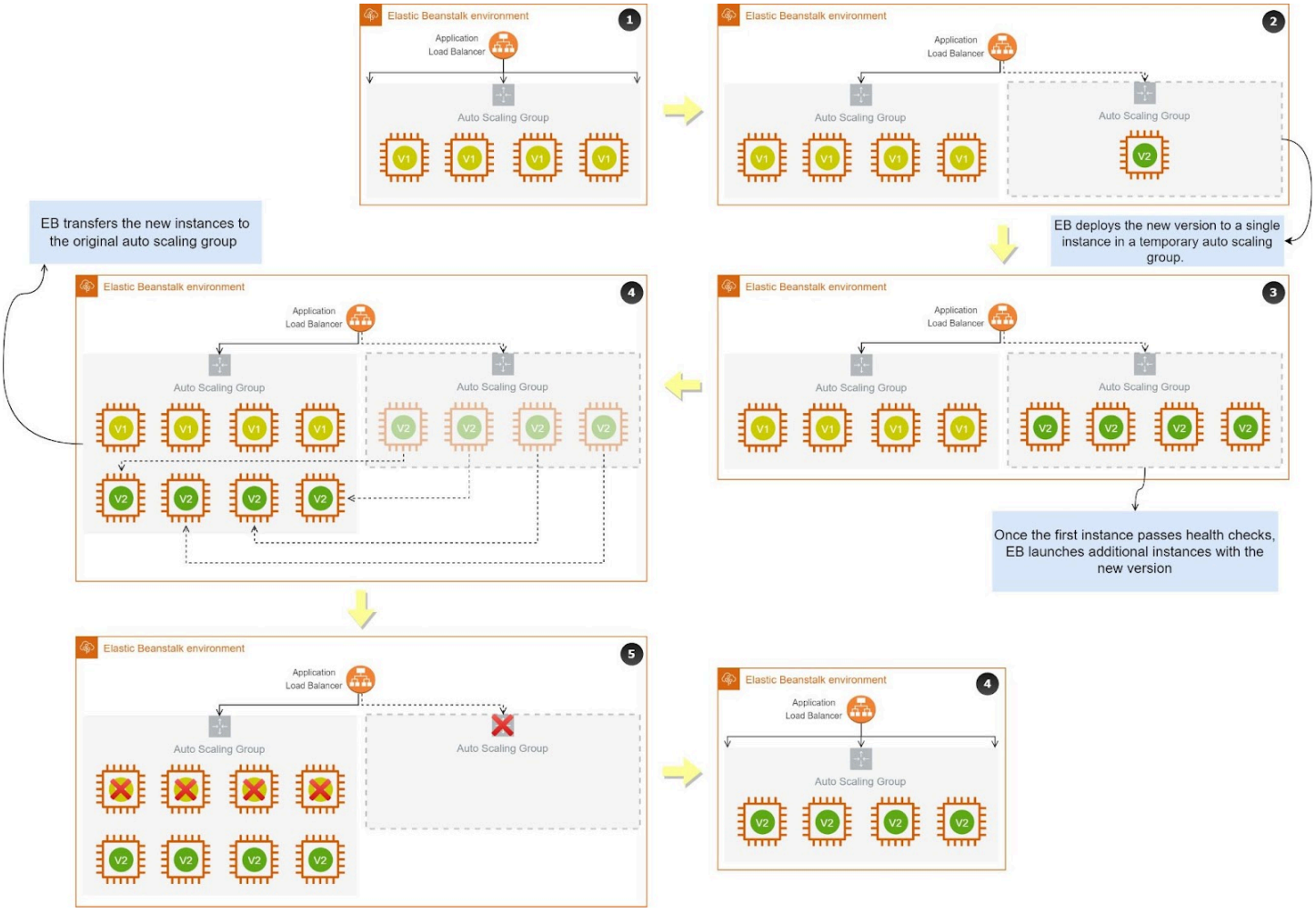
3. Rolling with an additional batch



- A variation of Rolling that solves the problem of applications running at a reduced capacity during deployment.

- When a new version is deployed, instead of taking an instance out of service, Elastic Beanstalk launches a temporary batch of instances with the current version. This way, the application always runs at full capacity.
- The new version serves traffic while the deployment is in progress; however, some users may continue to see the old version.
- Since Elastic Beanstalk spins new instances during deployment, make sure that your EC2 limits have enough capacity for the deployment to proceed. For example, say you have a quota of 20 instances per region, and you're already running 19. If your batch size is 2 instances, Elastic Beanstalk won't be able to create those extra instances because it would exceed the EC2 limits in your account.

4. Immutable

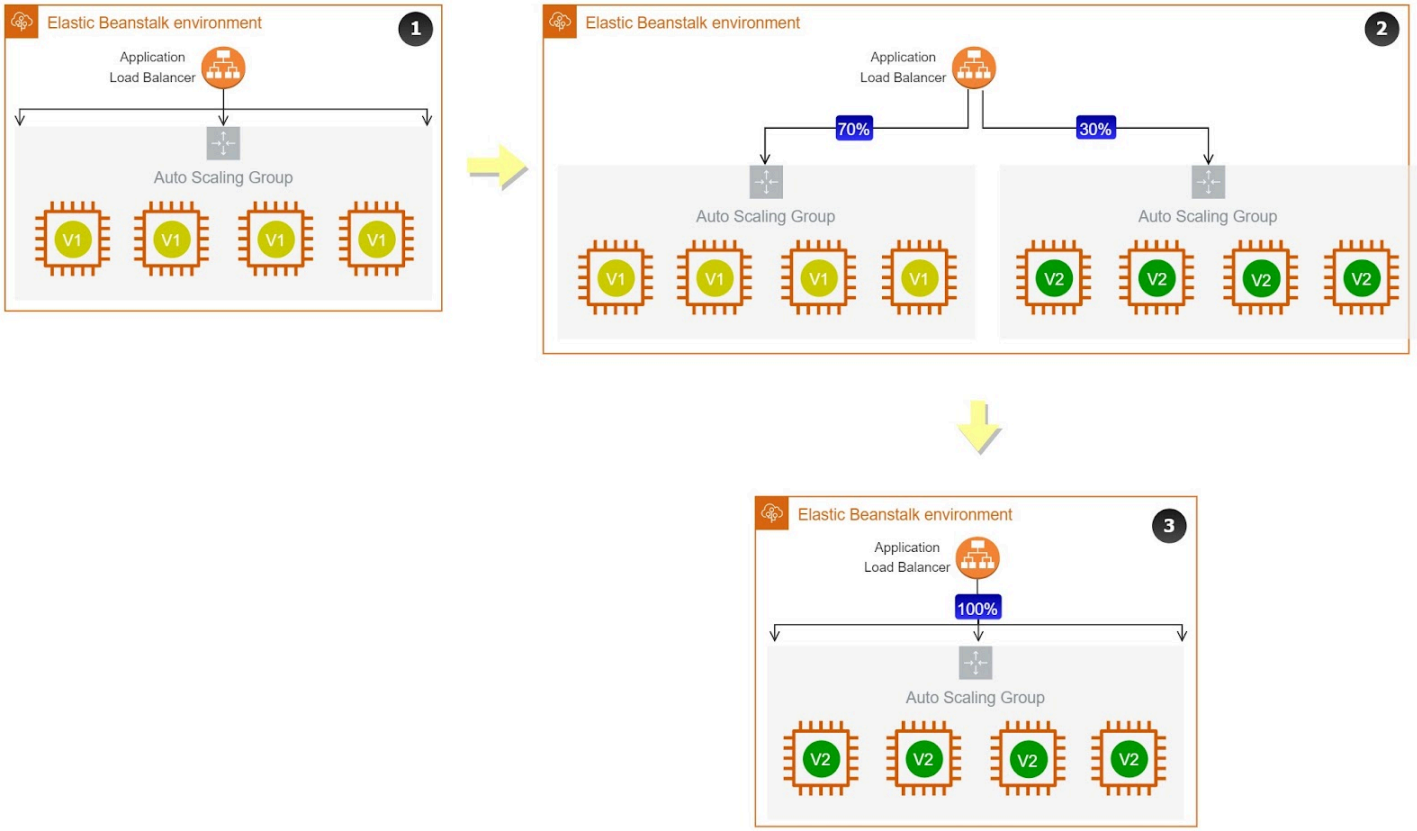


- Elastic Beanstalk creates a secondary Auto Scaling Group behind the environment's load balancer. The new version is first tested in a single instance within the new auto-scaling group; if the instance passes health checks, EB continues the deployment to additional instances. EB

then transfers the new instances to the original auto scaling group. The old instances and the temporary auto-scaling group are terminated upon a successful deployment.

- Like Rolling with an additional batch, there's a period of time in which both the new and old versions are serving requests.
- The rollback process does not cause downtime since the auto-scaling group hosting the stable version is isolated from the group running the new version. In case the deployment fails, simply terminate the temporary auto scaling group.
- Application also runs at full capacity during the deployment.
- This deployment strategy, however, is costly and subjected to on-demand EC2 limits since capacity is doubled for a brief time. It's also the slowest deployment method.

5. Traffic Splitting



- Suitable for A/B or canary testing
- Elastic Beanstalk installs the new version to a new set of instances and then shifts a percentage of traffic that you specified towards the new instances for a certain evaluation period. If the

instances pass the health checks until then, the deployment is successful. Elastic Beanstalk will route the rest of the traffic to the new instances and terminate the old instances. If the new instances do not pass the health checks or if you decide to cancel the deployment, Elastic Beanstalk will simply reroute the traffic back to the old instances and terminate the new instances.

References:
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.rolling-version-deploy.html
https://docs.aws.amazon.com/whitepapers/latest/introduction-devops-aws/aeb-deployment-strategies.html

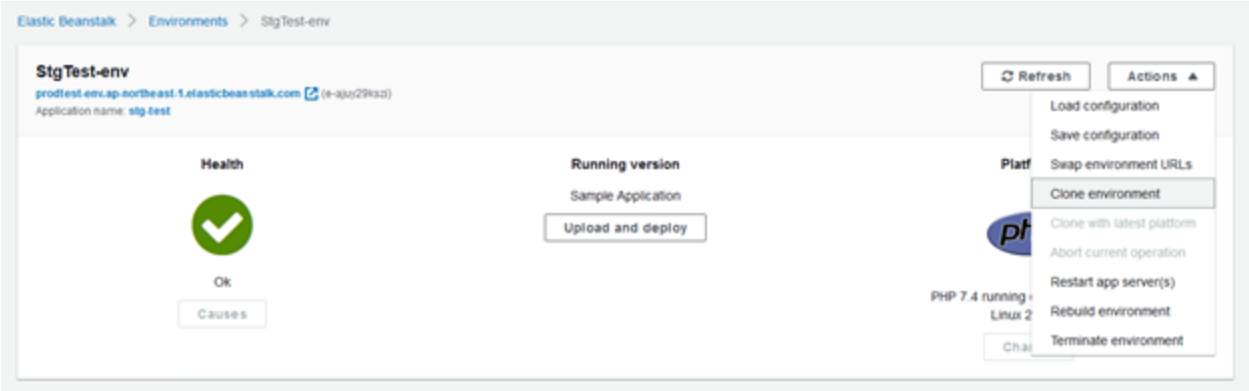
Blue/Green deployment

Elastic Beanstalk, by default, performs an in-place update when you deploy a newer version of your application. This can cause a short downtime since your application will be stopped while Elastic Beanstalk performs the application update. Blue/Green deployments allow you to deploy without application downtime.

In a Blue/Green deployment, you have two identical environments that are isolated from each other. The “Blue” environment refers to the environment to which current user traffic is directed, while the “Green” environment is where the latest application version is deployed. After the new version has been deployed and tested, you can shift the traffic away from the blue to the green environment.

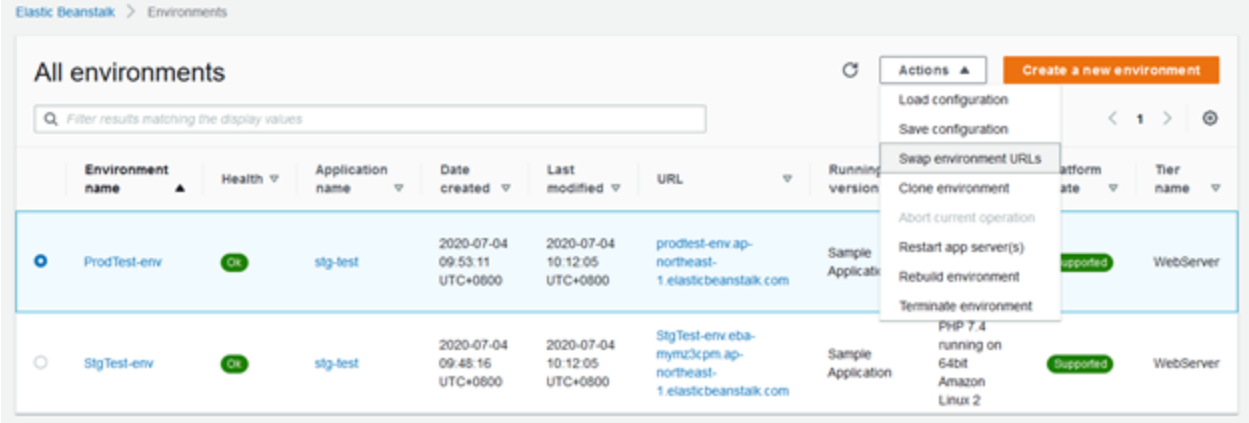
To implement a Blue/Green deployment for your Elastic Beanstalk application, you can perform the following steps:

1. Create another environment on which you will deploy the newer version of your application. You can clone your current environment for easier creation.

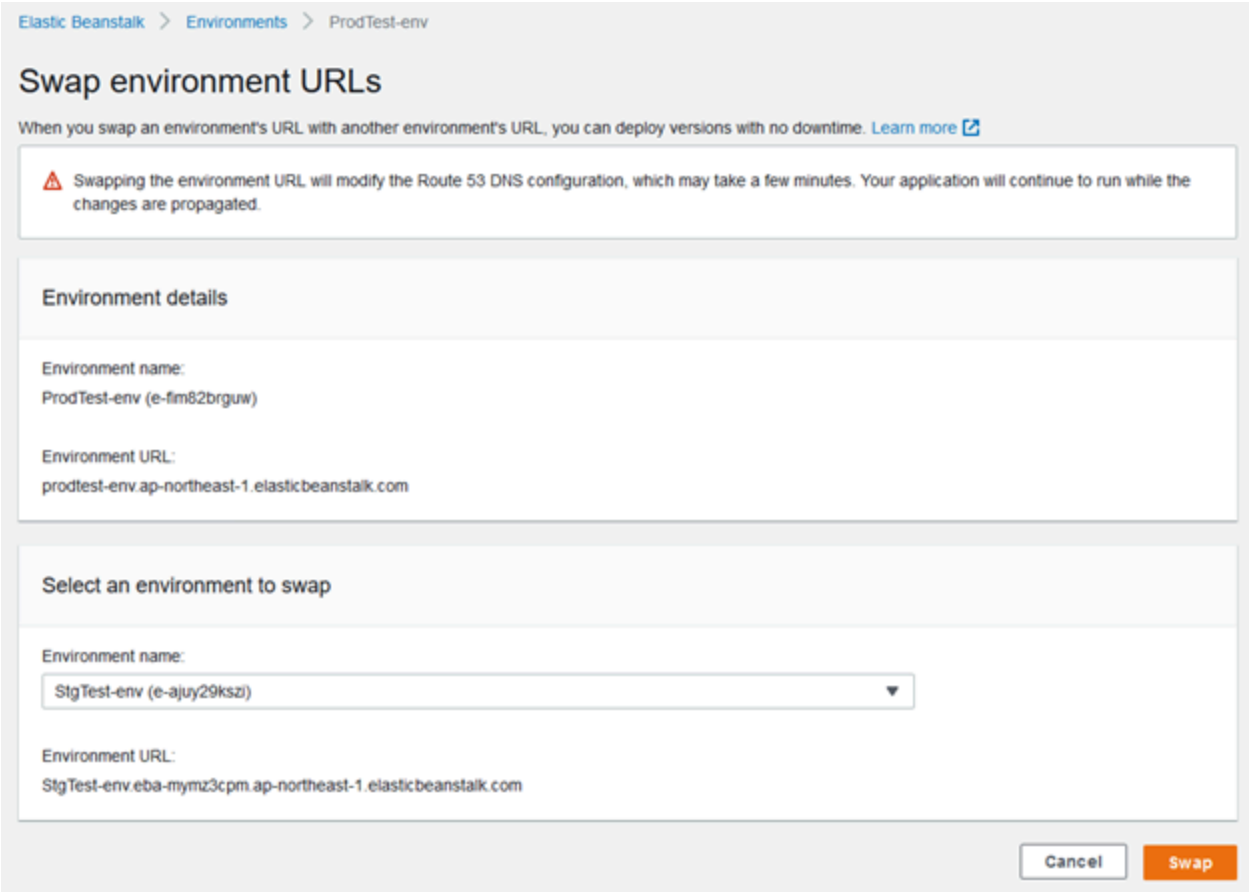


2. Once the new environment is ready, deploy a new version of your application. Perform your tests on the URL endpoint of your new environment.

3. After testing, select your Production environment, and click Actions > Swap environment URLs.



4. On the Swap Environment URLs page, select the newer environment and click Swap to apply the changes.



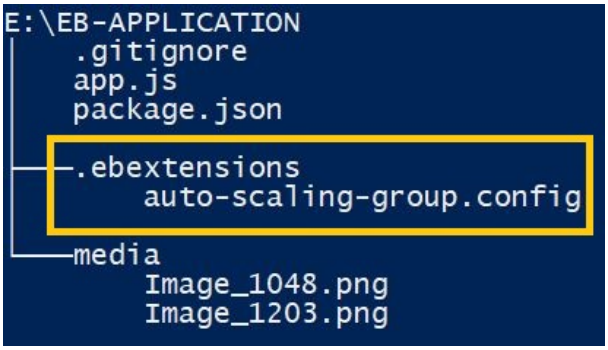
The biggest advantage of a blue/green deployment is that switching to a new application version won't involve any downtime since you're just redirecting traffic, unlike in-place deployment. The second advantage is that rolling back to the previous environment in case something goes wrong is much smoother because the switching is done at the network level

References:
https://docs.aws.amazon.com/whitepapers/latest/overview-deployment-options/bluegreen-deployments.html
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.CNAMESwap.html

`.ebextensions`

Customizing Elastic Beanstalk environments can be done in a variety of ways. You can alter some configuration parameters using the AWS Console, the EB CLI, or the AWS SDK. ebextensions is simply another method of customizing your environment, this time through the use of configuration files. These configuration files, which can be written in JSON or YAML, must be placed in a folder named .ebextensions. Although config files are written using YAML or JSON, their file extension must always be `.config`. When creating the

`.ebextensions` folder, always make sure to place it at the root level of your application source bundle. It shouldn't be located on sub-folders; otherwise, Elastic Beanstalk won't recognize them.



Aside from customizing EB environments, config files may also be used to customize the web servers on your EC2 instances. You can, for example, modify the Nginx configuration provided by Elastic Beanstalk without having to SSH into each of your instances. You can think of .ebextensions as Ansible for Elastic Beanstalk (although both can be used together), and it could be your primary tool for server configuration management.

References:
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/ebextensions.html
https://aws.amazon.com/blogs/compute/introducing-a-new-generation-of-aws-elastic-beanstalk-platforms/

Application Lifecycle Policy

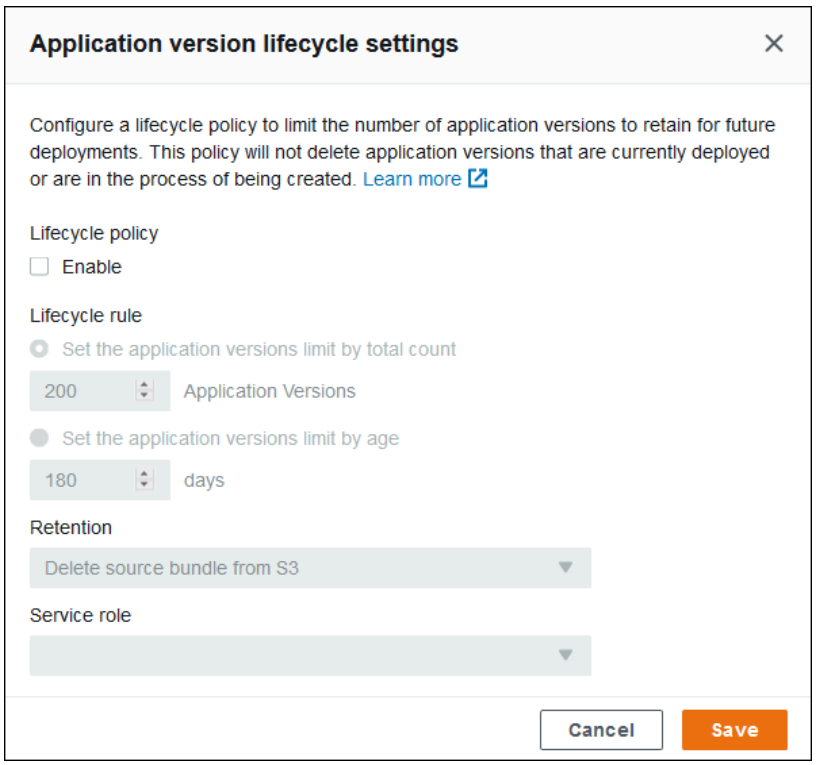
Each application version that you deploy in Elastic Beanstalk are labeled uniquely. It's easy to lose track of the number of versions growing under your application, especially if you're running iterative tests and often releasing modest code changes. Over time, the number of versions you will almost never use will increase and eventually hit the application version quota.

The application version quota is the maximum number of versions that can be deployed in an AWS Region. This is set to 1,000 by default and applies across applications; once reached, you will be unable to upload new application versions.

There are two options for dealing with this issue. First, you can ask AWS Support for a quota increase or just delete the versions you no longer need. Deleting old versions frees up the number of versions available for use by other applications. This is where the *application lifecycle policy* comes in handy. Instead of manually deleting versions that your application will likely no longer be using, you can set an application lifecycle policy that does it for you.

The application lifecycle policy specifies how long old versions should be retained before being deleted permanently. It also instructs Elastic Beanstalk to delete versions if the total number of application versions exceeds a specified limit. Imagine you have 10 Elastic Beanstalk applications in an AWS region with the default version quota. You may consider creating a lifecycle policy that limits the number of versions per application to 99. This way, the total version will always be less than 1,000. The reason why we want to keep the total number of versions less than the version quota is that the lifecycle policy only kicks in once Elastic Beanstalk has successfully created an application version. Once the quota has been reached prior to the deployment of a new version, the lifecycle policy won't take effect, and you must delete some versions manually.

There are two lifecycle rules that you may choose from if you enable Lifecycle policy.



- `Set the application versions limit by total count` - here you specify the maximum number of versions that you want to allocate to an application. Elastic Beanstalk will create a new version and delete the old one if you haven't met the quota in that region.
- `Set the application versions limit by age` - any versions older than the value you put here will be deleted once you upload a new version.

In the `Retention` option, you can choose whether or not to keep the source bundle of your application version in Amazon S3. If you choose to keep the source bundle in S3, Elastic Beanstalk will still delete the application version from its record, but the version's source code will remain in Amazon S3. And if you choose to delete the

source bundle in S3, then both the application version in Elastic Beanstalk and the source code in S3 will be deleted.

Reference:
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/applications-lifecycle.html

EB Command Line Interface (CLI)

The EB CLI is Elastic Beanstalk's own set of commands that you can use to manage your application environments from a terminal. Take note that EB CLI is different from the Elastic Beanstalk APIs accessible via the AWS SDK. EB CLI simplifies the process of deploying changes to your EB environment. For example, if you want to deploy a change that you've recently committed to your local repository, you can use the `eb deploy` command, and under the hood, EB CLI will create a ZIP archive of that change and deploy it to your instances. No need to package your code and upload it manually to the Elastic Beanstalk Console.

Aside from a local repository, you can also use the `eb deploy` command to deploy changes from a GitHub, GitLab, or Bitbucket repository. You can set the repository by configuring authentication and integrating it with AWS services. When using GitHub/GitLab/Bitbucket, every time you run `eb deploy`, EB CLI will push new commits to the repository and use the HEAD revision of your branch to create the archive that it deploys to the EC2 instances in your environment. As a result, you can deploy incremental code updates without having to upload the entire project each time.

References:
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3.html
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb3-cmd-commands.html

Amazon Simple Queue Service (SQS)

Standard vs. FIFO queue

Amazon SQS supports two message queue types: *standard queues* and *first-in-first-out* (FIFO) *queues*. The Standard queue delivers messages at least once. Although this ensures that no messages are lost, it comes at the cost of the same messages being delivered more than once, resulting in duplicate messages in the queue. Standard queue supports best-effort ordering, meaning the messages received in the queue can sometimes be in a different sequence than the one in which they were sent. The SQS FIFO queue, on the other hand, is designed to preserve the order of the messages arriving in the queue. Additionally, messages are delivered exactly once to help you avoid duplicates.

In terms of throughput, the Standard queue beats FIFO queue by a large margin. Standard queues can handle an unlimited number of transactions per second (TPS) per API action, whereas FIFO queues can only support up to 3,000 messages per API action.

References:
https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/standard-queues.html
https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/FIFO-queues.html

Concepts

Visibility Timeout

SQS does not delete messages automatically after they have been processed by an application. A message remains in a queue until the retention period expires or if the consumer deletes it manually. For this reason, if you have multiple consumers polling a queue, it is likely that a message could be retrieved and processed more than once. To mitigate this issue, you can adjust the value of your queue's visibility timeout. The visibility timeout is the period of time during which Amazon SQS blocks other consumers from receiving and processing messages. You can modify the visibility timeout of your queued messages in the SQS Console or use the `ChangeMessageVisibility` API. There is no guarantee, however, that you will never receive the same message twice. It's up to you to figure out the sweet spot that works for your application.

Short Polling vs. Long Polling

Polling is the method by which SQS retrieves messages from the queue and sends them to the requesting consumer. SQS supports two polling methods: *long polling* and *short polling (default)*. With short polling, the ReceiveMessage request searches only a subset of the SQS servers to find messages to include in the response. SQS sends the response right away, even if the query finds no messages. And since only a subnet of servers is searched, a request might not return all of your messages. Short polling is best for time-sensitive applications or batch applications that can send another query if it received an empty response previously. To enable Short polling, set your wait time to 0.

With long polling, the ReceiveMessage request searches all of the SQS servers for messages. SQS returns a response after it collects at least one available message, up to the maximum number of messages specified in the request, and will only return an empty response if the polling wait time expires. The maximum long polling wait time is 20 seconds. Long polling helps reduce the cost of using SQS by eliminating the number of empty responses and false empty responses. To enable Long polling, set your wait time to be greater than 0.

Amazon SQS Extended Client Library for Java

The maximum payload of a message that SQS supports can be up to 256KB in size and in any format. You are charged for each 64KB chunk of payload as 1 request. This means that when you retrieve a 256KB message, it is treated as 4 separate requests. In some situations, this limit can be a bottleneck. Most of the solutions that I see on the internet leverage the power of Amazon S3 in which payloads larger than 256KB are stored in an S3 bucket instead of an SQS queue. The link of the S3 object is then sent to the queue. A simpler alternative to this is by utilizing the Amazon SQS Extended Client Library for Java. This library wraps the same functions of the solution I described above in APIs. The only drawback, of course, is the limited language support. If you're using a different language like NodeJs, you're gonna have to code the solution yourself.

References:
https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-short-and-long-polling.html
https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-visibility-timeout.html
https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-s3-messages.html

Amazon Simple Notification Service (SNS)

Amazon Simple Notification Service (Amazon SNS) is a messaging gateway that allows transmitting messages from producers, known as *publishers*, to subscribers, referred to as *consumers*. This communication occurs asynchronously, with publishers issuing messages to a topic—a designated channel and point of access that facilitates the message flow. Amazon SNS is designed to support a seamless, asynchronous exchange of messages between the initiating parties and their subscribers, accommodating a wide array of endpoints, including Amazon SQS, AWS Lambda, HTTP, email, mobile push notifications, and text messaging (SMS).

Amazon Simple Notification Service (Amazon SNS) offers a robust suite of features for messaging and notifications across various platforms. Here are some key features and capabilities:

- **Application-to-application messaging** - This allows for seamless messaging between different applications using AWS endpoints like Lambda functions, SQS queues, and HTTP/S endpoints, which can be crucial for integrating various systems within a cloud infrastructure.
- **Application-to-person notifications-** SNS provides the capability to send notifications directly to users through mobile apps, SMS, and email, making it versatile for direct user engagement.
- **Standard and FIFO topics** - FIFO topics ensure the sequence of message ordering and also prevent duplicates, whereas standard topics are appropriate where the management of message sequencing is not necessary.
- **Message durability -** Messages in Amazon SNS are stored redundantly across multiple servers and data centers to ensure high durability. If messages fail to deliver, SNS implements a Delivery Retry Policy. Futhermore, to handle messages that are not delivered before the expiry of the delivery retry policy, you can set up a dead-letter queue. This ensures that undelivered messages are preserved and managed appropriately.
- **Message archiving, replay, and analytics -** Messages can be archived directly to services like Amazon S3 or analyzed with Amazon Redshift. This is facilitated through integrations like Kinesis Firehose delivery streams.
- **Message attributes -** This will enable you to attach structured metadata to messages distributed through the service. This metadata can include various details such as timestamps, geographic data, signatures, and identifiers, providing context and additional information about each message.
- **Message filtering-** This enables subscribers to receive only those messages that meet specific criteria defined in their subscription filter policies. This feature effectively reduces unnecessary data processing and network traffic by ensuring subscribers only get messages relevant to their needs.
- **Message security -** Server-side encryption in Amazon SNS safeguards message contents stored in topics by utilizing encryption keys managed through AWS Key Management Service (KMS).

References:
https://docs.aws.amazon.com/sns/latest/dg/welcome.html
https://docs.aws.amazon.com/sns/latest/dg/welcome-features.html

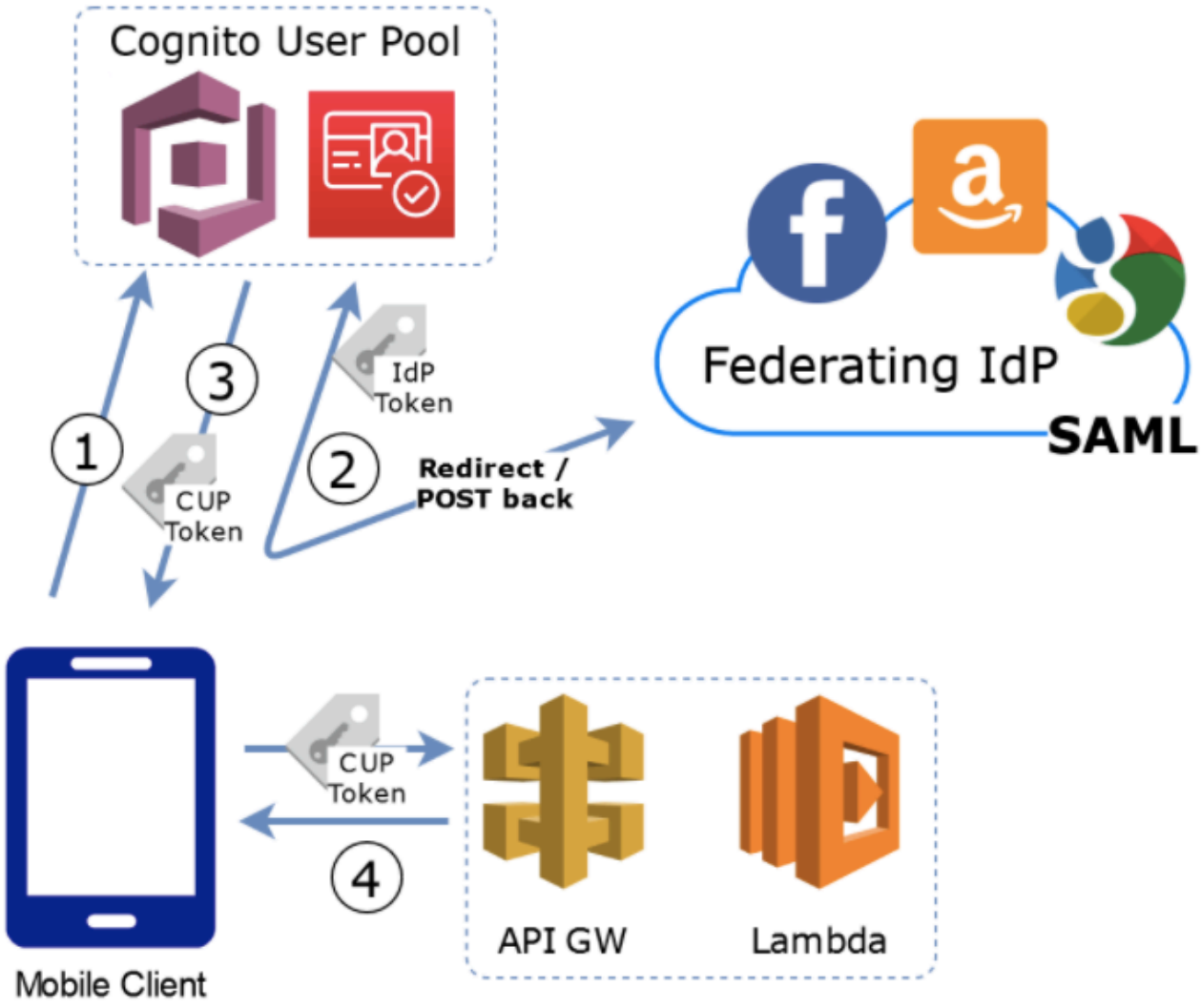
Amazon Cognito

User Pool vs. Identity Pool

Amazon Cognito User Pools are used to authenticate a user's identity using username/passwords. You can also authenticate users using external identity providers such as Amazon, Facebook, or Google or a SAML-supported authentication such as Microsoft Active Directory. Please note that Cognito User Pool is **only** responsible for obtaining the JSON Web tokens (JWT) supplied by these providers. You can have a validation logic in your application that verifies the JWT in exchange for access to your own resources. The JWT can also be used to exchange for temporary AWS credentials using AWS STS or Amazon Cognito Identity Pools.

With Cognito User Pools, you can provide sign-up and sign-in functionality for your mobile or web app users. You don't have to build or maintain any server infrastructure on which users will authenticate.

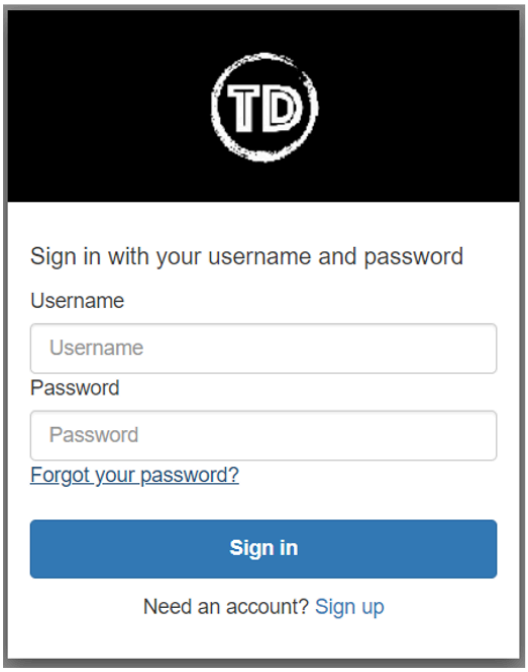
This diagram shows how authentication is handled with Cognito User Pools:



1. Users send authentication requests to Cognito User Pools.
2. The Cognito user pool verifies the identity of the user or sends the request to Identity Providers such as Facebook, Google, Amazon, or SAML authentication (with Microsoft AD).
3. The Cognito User Pool Token is sent back to the user.
4. The person can then use this token to access your backend APIs hosted on your EC2 clusters or in API Gateway and Lambda.

Features:

- Built-in User interface for sign-in/sign-up. This hosted UI is customizable; you can change how it looks, modify the size and color of buttons and input fields, and even upload your brand logo if you want.



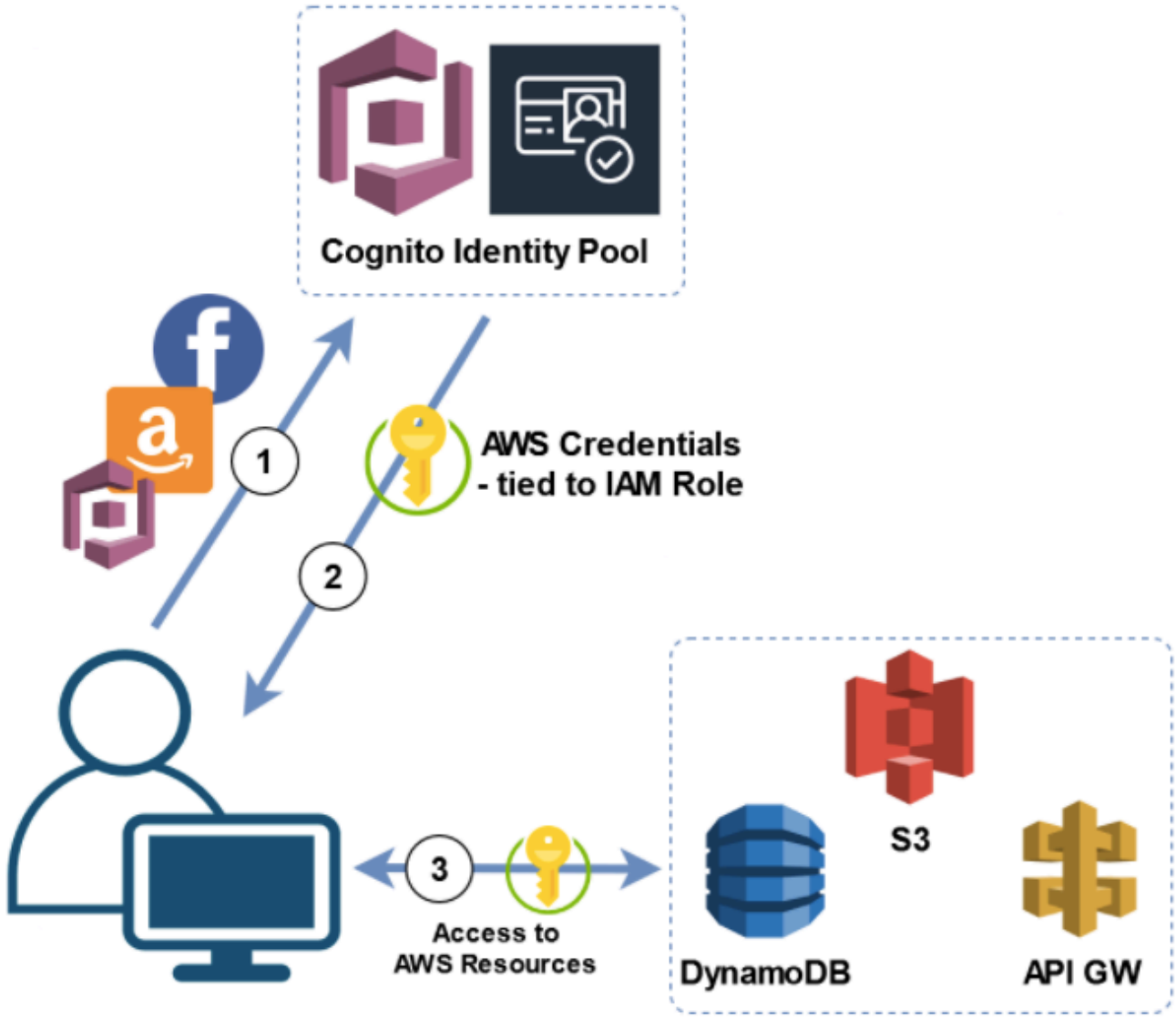
- Multi-factor authentication (MFA)
- Checking of compromised credentials
- Account takeover protection
- Phone and email verification
- Enforcement of strong password requirements

Amazon Cognito Identity Pools

Cognito Identity Pools (Federated Identities) provide different functionality compared to User Pools. Identity Pools are used for User Authorization. You can create unique identities for your users and federate them with your identity providers. Using identity pools, users can obtain temporary AWS credentials to access other AWS services.

Identity Pools can be thought of as the actual mechanism authorizing access to AWS resources. When you create Identity Pools, think of it as defining who is allowed to get AWS credentials and use those credentials to access AWS resources.

This diagram shows how authorization is handled with Cognito Identity Pools:



1. The web app or mobile app sends its authentication token to Cognito Identity Pools. The token can come from a valid Identity Provider, like Cognito User Pools, Amazon, or Facebook.
2. Cognito Identity Pool exchanges the user authentication token for temporary AWS credentials to access resources such as S3 or DynamoDB. AWS credentials are sent back to the user.
3. The temporary AWS credentials will be used to access AWS resources.

You can define rules in Cognito Identity Pools for mapping users to different IAM roles to provide fine-grain permissions.

1. Here's a table summary describing Cognito User Pool and Identity Pool:

Cognito User Pools	Cognito Identity Pools
Handles the IdP interactions for you	Provides AWS credentials for accessing resources on behalf of users
Provides profiles to manage users	Supports rules to map users to different IAM roles
Provides OpenID Connect and OAuth standard tokens	Free
Priced per monthly active user	

References:
https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html
https://aws.amazon.com/premiumsupport/knowledge-center/cognito-user-pools-identity-pools/

Granting access for unauthenticated Identities

If you have an application that serves media assets like images or videos from Amazon S3, you might want to have some control over what your users can and cannot do. You can, for example, allow registered users to view, upload, and share images, while guest users can only view photos posted by those with accounts. Obviously, the permissions granted to guest users should be more restrictive than those granted to authenticated identities.

You can enable unauthenticated identities upon the creation of an Identity Pool or by modifying the setting of an existing Identity Pool. Cognito Identity Pool with unauthenticated access works by providing a unique identifier and AWS credentials for your guest users. You can control their permissions by defining the policy associated with your unauthenticated identities' role. To restrict guest users to viewing images only, set S3 read-only permissions in the role's policy. As a result, guest users will be unable to delete, update, or upload images to your application.

References:
https://docs.aws.amazon.com/cognito/latest/developerguide/identity-pools.html

https://docs.aws.amazon.com/location/latest/developerguide/authenticating-using-cognito.html

AWS Amplify

AWS Amplify is one of the many development services in AWS that helps you build extensible, full-stack web and mobile apps faster. It allows you to easily start your application development and automatically scale your resources with less management overhead.

Put simply, AWS Amplify is a set of purpose-built tools and features. It consists of **Amplify Studio**, **Amplify Libraries**, **Amplify CLI**, and **Amplify Hosting**. Let's quickly cover its different modules one by one.

Amplify Studio

Amplify Studio is a visual development environment that simplifies the development of your web and mobile apps. You can easily build your frontend UI with its set of ready-to-use UI components, which you can directly connect to your app backend. Since your UI components are already pre-built, the number of your boilerplate code will be reduced, allowing you more time to focus on implementing the application business logic.

You can define your data models, implement user authentication, and add file storage using the Amplify Studio without any backend expertise. In addition, you can import Figma prototypes built by your application designers into the Amplify Studio for more seamless collaboration.

Amplify Hosting

Amplify Hosting is a fully managed CI/CD and hosting service that allows you to host) secure, reliable, fast web single-page applications (SPA) via the AWS content delivery network. Under the hood, it deploys your applications to Amazon CloudFront's content delivery network, which comes with hundreds of points of presence globally. Amplify Hosting also allows you to set up your own custom domain for your applications and add custom alarms that send notifications if a certain metric exceeds a threshold that you specify.

- Key features
    - Supports apps (built with Next.js 12 or later) that use server-side rendering (SSR)
    - Has deep integration with Cypress, which allows you to run end-to-end (E2E) tests. Cypress' configurations and commands must be included in the **amplify.yml** build settings of your application.
    - Supports external repositories like GitHub, Bitbucket, and GitLab, making it easy to set up a continuous deployment pipeline for your existing web applications.

- You can preview changes during code reviews
- Offers password protection for your web app so it can't be publicly viewable while you're developing new features.
- Provides instant cache invalidations to ensure that every code changes are immediately visible to users without purging the cache manually.
- Set up rewrites and redirects to maintain SEO rankings.

AWS Amplify Libraries

The AWS Amplify Libraries are open-source JavaScript libraries in the Amplify Framework. These are specifically designed to aid you in building AWS-powered mobile and web apps. The Amplify JavaScript libraries are supported in React, React Native, Angular, Ionic, Vue, and different web and mobile frameworks.

Amplify Command Line Interface

The Amplify Command Line Interface is a CLI toolchain that you can use to configure and maintain your app backend straight from your local desktop. The Amplify CLI has an interactive workflow and intuitive use cases that you can leverage, such as authentication, storage, and API. It allows you to test your features locally and deploy your app to multiple environments. This tool provides infrastructure-as-code templates, which are automatically loaded to CloudFormation or AWS SAM. With the Amplify CLI, you can have a much more effective team collaboration and easy integration with Amplify's CI/CD workflow.

References:
https://docs.aws.amazon.com/amplify/latest/userguide/welcome.html
https://docs.amplify.aws/

AWS CloudShell

AWS CloudShell is a powerful, managed command-line interface (CLI) provided by Amazon Web Services. It enables users to manage AWS resources directly from their web browser without installing or configuring the traditional AWS CLI locally. Integrated seamlessly with the AWS Management Console, CloudShell provides a convenient and immediate access point for AWS management tasks, with the benefit of being pre-authenticated using your AWS console credentials.

Key Features of AWS CloudShell:

1. CloudShell environment includes essential tools and programming languages such as Python, Node.js, Java, and the AWS CLI, all pre-installed and ready to use.

2. CloudShell offers 1 GB of free storage for each AWS Region, which is provided at no additional cost. This storage is allocated to your home directory and persists across sessions, enabling you to save and reuse scripts, configurations, and other essential data.
3. CloudShell utilizes an Amazon Linux 2-based compute environment, providing a stable and compatible setting for your scripts and tools.
4. CloudShell is provided at no additional cost; you only incur charges for the AWS resources you manage through the shell, ensuring a cost-effective way to access AWS command line management.
5. CloudShell supports a variety of programming languages and AWS SDKs, enhancing its adaptability for diverse development and operational requirements.

References:
https://docs.aws.amazon.com/cloudshell/latest/userguide/welcome.html
https://docs.aws.amazon.com/cloudshell/latest/userguide/limits.html

AWS CodeBuild

AWS Codebuild allows you to compile, build, run tests, and produce an artifact of your code. This can be integrated into your existing CI/CD process. For example, you are using GitHub or BitBucket as version control for your source code repository. After a developer pushes new code to the Development branch, you can have an automatic trigger for CodeBuild to run your project build, test your application, and then upload the output artifact to S3. The artifact file on S3 can then be used by deployment tools such as AWS CodeDeploy to have it deployed on your EC2 instances, ECS, or Lambda functions.

Buildspec file

The build specification file is a YAML file that defines the commands that CodeBuild will execute throughout each build step. The buildspec.yml file can be set when you're creating a build project in the AWS Console or it can also be included as a part of your source code. If you're including the buildspec file as part of your source code, it must be named *`buildspec.yml`* (all in lowercase) and should be placed at the root level of your project folder.

Reference:
https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html

Integrations with other AWS Services

Amazon Elastic Container Registry (ECR)

Amazon ECR is a managed Docker registry service in AWS. If you've used Docker Hub before then, you'll have an idea of what ECR is. In Docker, to access images stored in a private registry, we need to log in first using `docker login`. But how about in Amazon ECR? Amazon ECR has its own method of authentication. Since it is a service that manages a container registry under the hood, you need to be authenticated first at the ECR level. This means that you can't use the docker login command to log in to your private ECR repository directly. You can think of it as two-level access.

First, you need to issue the `aws ecr get-login-password` command. This returns an authentication token that you can use to authenticate against an Amazon ECR registry. You can then pass the authentication token on the `--password` flag of the docker login command. For better security, you can optionally pass the password via the `--password-stdin` flag to provide the token through the Standard Input stream. This prevents the token from ending up in the shell's history or log files.

```
aws ecr get-login-password \
    --region <region> \
    docker login \
    --username AWS \
   --password-stdin <aws_account_id>.dkr.ecr.<region>.amazonaws.com
```

Say you want to automate the image build of a docker file, you can take the command above and write it at the pre-build phase of your buildspec file and write a docker push command under the post-build phase.

Amazon EventBridge (formerly known as CloudWatch Events)

Amazon EventBridge is a service used for building event-driven applications in AWS. EventBridge can capture events produced by your build environment. With EventBridge or CloudWatch Events, you can create an event

rule that monitors the status of your build — see if it's in progress, succeeded, failed, or stopped. You can then hook an SNS topic to that event so you'll be alerted if something bad happens during the build.

AWS Systems Manager Parameter Store

AWS Systems Manager Parameter Store provides secure, hierarchical storage for configuration data management and secrets management. In simple terms, it's a service that you can use for storing data such as passwords, database credentials, environment variables, and any plaintext or encrypted data.

So why would you want to use this service with CodeBuild when you can store variables in your build environment? The reason is that CodeBuild has a limit to the length of all environment variables that you can use. The build will encounter an error if your environment variables (both key names and values) reach a combined maximum of around 5,500 characters.

One solution that AWS recommends is to store largely sized environment variables in Amazon Systems Manager Parameter Store. You can then execute the `GetParameters` API in your buildspec file to retrieve those variables at the pre-build phase.

References:
https://docs.aws.amazon.com/codebuild/latest/userguide/troubleshooting.html#troubleshooting-large-env-vars
https://docs.aws.amazon.com/codebuild/latest/userguide/sample-ecr.html

AWS CodeDeploy

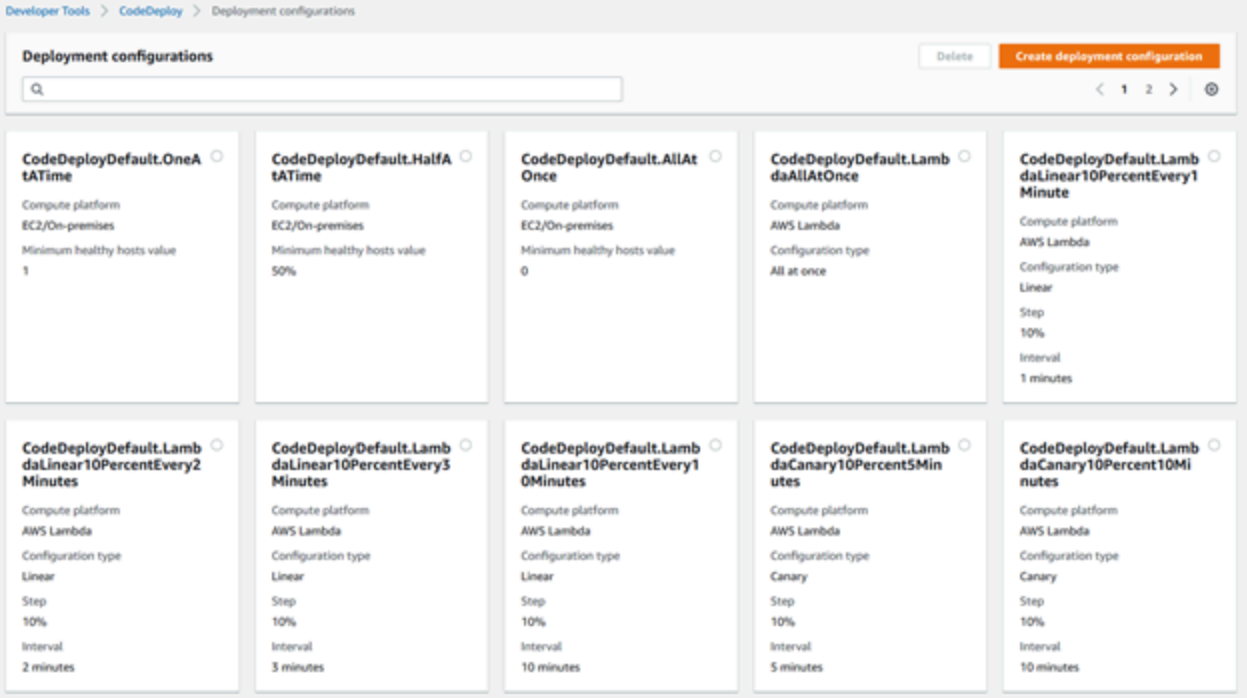
Deployment configurations

Deployment configuration is a set of rules and conditions that you can tweak to instruct CodeDeploy on how aggressive or conservative your deployment approach is going to be. When you deploy a new application version on your target compute platform (EC2 or ECS or Lambda), you can have several options for shifting network traffic to the newer version.

Using CodeDeploy, you can control how many instances are getting updated at any given time during the deployment. This is important because, during deployments, the application is stopped while the new version is deployed. You want to make sure that enough instances are online to serve the traffic as well as the ability to roll-back the changes when an error during the deployment occurs.

On the AWS Console, when you click on CodeDeploy > Deployment configurations, you will see the list of AWS-defined deployment strategies that you can use for your deployments. You can create your own

deployment configuration if you want to, but we'll discuss the most common ones here, which you will likely encounter in the exam.



CodeDeployDefault.AllAtOnce – this is the fastest deployment. The application will stop on all EC2 instances and CodeDeploy will install the newer version on all instances. The application will stop serving traffic during the deployment as all instances are offline.

CodeDeployDefault.OneAtATime – this is the slowest deployment. CodeDeploy will stop only one instance at a time. This will take time to deploy on all instances but the application will remain online since only one instance is offline at any given time.

CodeDeployDefault.HalfAtATime – half, or 50% of the instances will be offline during the deployment, while the other half are online to serve traffic. This is a good balance between a fast and safe deployment.

CodeDeployDefault.LambdaLinear10PercentEvery10Minutes – Deployment for Lambda functions. This deployment will use Aliases on the backend to shift traffic from the old version to the newer version. Ten percent of traffic will be shifted to the newer version every 10 minutes. Deployment will run at 10-minute intervals until 100% of the traffic is shifted to the newer version.

CodeDeployDefault.LambdaCanary10Percent10Minutes – Deployment for Lambda functions. This deployment will use Aliases on the backend to shift traffic from the old version to the newer version. Initially, 10% of the traffic will be shifted to the newer version. This will only last 10 minutes so you can have time to

check the application logs. After 10 minutes, the remaining 90% of the traffic will be shifted to the newer version.

Reference:
https://docs.aws.amazon.com/codedeploy/latest/userguide/deployment-configurations.html

AppSpec file

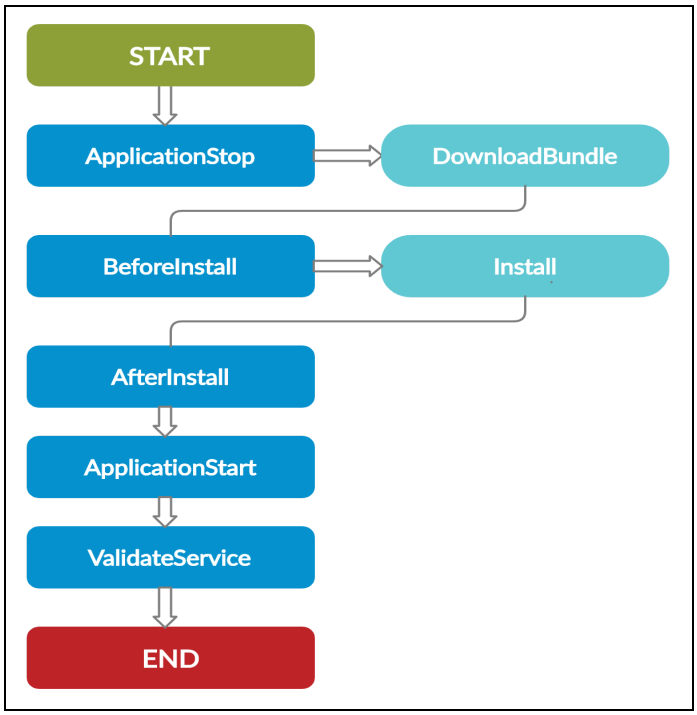
CodeDeploy manages each deployment stage as a collection of lifecycle event hooks through an `AppSpec` file. The `AppSpec` file allows you to instruct CodeDeploy on the deployment tasks that will be executed throughout your application's deployment process. For example, you may instruct CodeDeploy to install dependencies that your application requires before installing the new application version, or you can tell CodeDeploy to alter some file permissions before starting the deployment.

- The AppSpec file can be written in **YAML** or **JSON**.
- YAML is the **only available format** for configuring `AppSpec` in EC2/On-premise.
- You need to install the **CodeDeploy Agent** for configuring `AppSpec` in EC2/On-premise.
- The `AppSpec` file must be included in the **application's root folder**.

Reference:
https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file.html

Lifecycle Event Hooks

You can insert custom scripts or commands at each event hook to control the process of your deployment. The commands are executed in sequence based on the order of event hooks. Take note that each computer platform (Amazon ECS AWS Lambda, Amazon EC2, On-premises) has different hook events. In this section, we'll cover the event hooks available in an EC2/On-premises deployment.



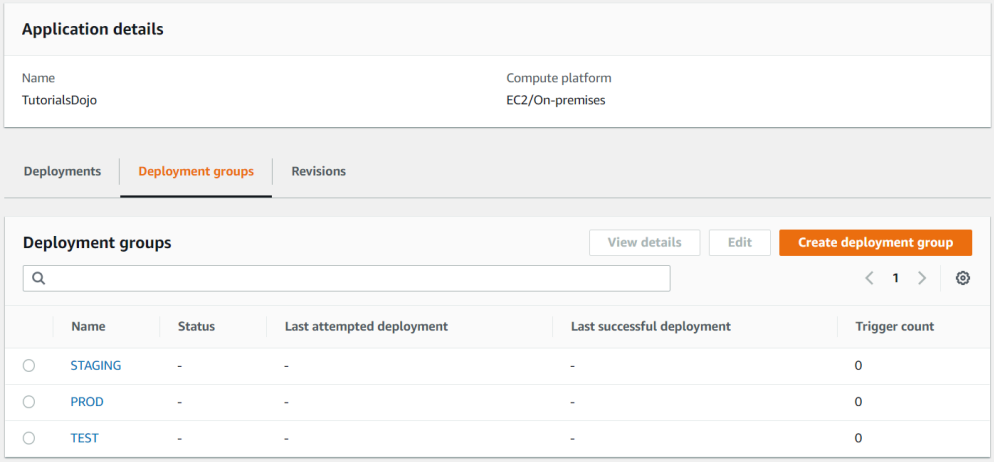
1. **`Start`** - The first lifecycle event is the Start event and this initiates the instance for deployment. The CodeDeploy Agent automatically executes this for you, meaning you don't have to include this event hook in your AppSpec file.
2. **`ApplicationStop`** - used for running any scripts that will prepare your instance in receiving the new application version. For example, If you're deploying a new application version, say v.2, you can include a script that will disable the previous version v.1 in this event hook.
3. **`DownloadBundle`** - this event is triggered automatically following the ApplicationStop event. You don't have to include this in your AppSpec file. During this stage, CodeDeploy will pull the new version v.2 from a source repository into the instance, as the name implies.
4. **`BeforeInstall`** - this event hook can be useful if you want to back up the configuration files of your old application or any old logs and store them elsewhere in your disk, so they are not overwritten when the new application is copied.
5. **`Install`** - this event is executed automatically. During the Install event, CodeDeploy copies your new application files to the file destination that you specified.
6. **`AfterInstall`** - this event can be used to decode database string information, application credentials, and perhaps other encrypted data that you've sent down as part of your application bundle and move everything to its proper destination.
7. **`ApplicationStart`** - use this event to enable your application back into service.
8. **`ValidateService`** - this event can be used to execute any validation script to see if the application is working as intended.

9. **End** - Like the Start event hook, this is executed automatically; no need to include this event hook in your AppSpec file. The End event would notify the central service if the deployment went successfully or not. If something goes wrong during the deployment, you'll be able to choose what measures to take next, such as rolling back to a known working version of your application.

References:
https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file-structure-hooks.html#appspec-hooks-server
https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file-example.html
https://docs.aws.amazon.com/codedeploy/latest/userguide/application-specification-files.html#appspec-files-on-ecs-compute-platform

Deployment groups

The collection of instances or Lambda functions to which you want to deploy an application revision is referred to as a *deployment group*. Your deployment group comprises information such as service role, which grants CodeDeploy permission to access the necessary AWS resources to accomplish its job.



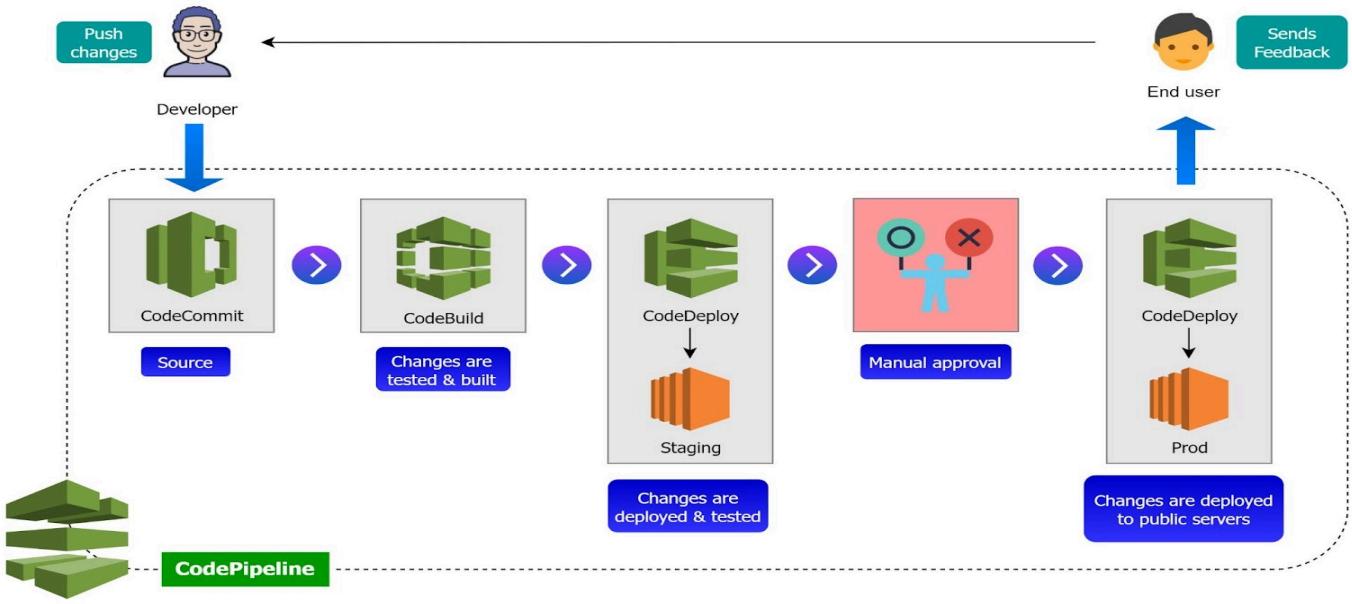
You can use deployment groups to create multiple stages in your pipeline before deploying to production. For example, to ensure code quality, you can first deploy a code update to a set of instances in a Test environment. After that, you can deploy it to a Staging environment for testing. Finally, if you're satisfied with the outcome of your deployment, you can move the code to your Production environment.

Reference:
https://docs.aws.amazon.com/codedeploy/latest/userguide/deployment-groups.html

AWS CodePipeline

Manual approval actions

Not every organization favors the idea of a fully automated deployment process — deploying codes from version control up to production with no human intervention. Most of the time, you assign someone to take over, test, and make the final decision on whether or not to push code changes to production. This is what *continuous delivery* is all about. The main difference between continuous delivery (AWS CodePipeline) and continuous deployment (AWS CodeDeploy) is the presence of manual approval.



CodeDeploy takes your code from a repository and deploys it to your servers with no human intervention. In AWS CodePipeline, you can insert a manual approval action stage before proceeding to the final deployment stage. At the point of the manual approval stage, CodePipeline will pause the deployment so that someone you trust can review the changes. If the application is not behaving the way it's supposed to prior to the change, the authorized user can reject the deployment. If this happens CodePipeline will stop the deployment. Conversely, if the user approves the change, CodePipeline resumes the deployment to production.

References:
https://docs.aws.amazon.com/codepipeline/latest/userguide/approvals-action-add.html
https://aws.amazon.com/blogs/devops/complete-ci-cd-with-aws-codecommit-aws-codebuild-aws-codedeploy-and-aws-codepipeline/

AWS CodeWhisperer

Amazon CodeWhisperer is a machine learning-enhanced tool that boosts your coding efficiency by offering real-time code suggestions within your favorite Integrated Development Environments (IDEs) such as VS Code, IntelliJ IDEA, and AWS Lambda console. This service intelligently analyzes your comments and existing code to provide personalized recommendations ranging from single lines to complete functions, enabling you to focus solely on strategic tasks rather than routine coding.

Amazon CodeWhisperer offers several key features designed to enhance software development productivity:

- Real-Time Code Suggestions: As you develop software, CodeWhisperer offers you immediate suggestions that align with the context of your work, ranging from snippets to entire functions, based on your existing code and comments.
- IDE Integration: You can enjoy seamless integration with your preferred Integrated Development Environments, such as VS Code, IntelliJ IDEA, and AWS Lambda console. This integration means you get coding assistance directly where you work.
- Security Scans: CodeWhisperer actively scans your code for potential security threats, helping you to identify and rectify vulnerabilities before they become issues..
- Customization Options: You have the ability to customize CodeWhisperer to recognize your organization's internal APIs and libraries. This customization makes the suggestions more relevant and tailored to your specific development environment.
- Support for Multiple Programming Languages: Whether you're working in Python, Java, JavaScript, C#, or TypeScript, CodeWhisperer supports these languages, making it a versatile tool across various projects.
- Enterprise Controls and Easy Sign-Up: With features like enterprise administrative controls and simple Single Sign-On integration, CodeWhisperer is easy to manage and deploy across your development teams, enhancing both productivity and governance.

References:
https://aws.amazon.com/codewhisperer/features/
https://docs.aws.amazon.com/codewhisperer/latest/userguide/what-is-cwspr.html

AWS CodeArtifact

Software packages play an important role in development as they often provide out-of-box solutions to solve common problems when working on projects. Sometimes, it may also be necessary to build a custom library on top of existing ones to fit specific use cases. This is not a big deal when you're working on a project alone, however, when working in a collaborative environment, some factors are needed to be considered. For example, maintaining consistency across your CI/CD process; you don't want your team to be installing

different versions of the same package as that might cause incompatibility issues down the line. Also, a developer might wildly pull open-source libraries from the internet. Without proper checking, you might end up with an application full of vulnerabilities, thus, it's critical to only allow the use of libraries that have been reviewed and approved by team leaders.

AWS CodeArtifact helps solve this problem. It is a fully managed artifact repository service that enables teams to securely store and share software packages used in application development. Since CodeArtifact is serverless, you save on the cost and effort of setting up traditional repository solutions. Managing packages using AWS CodeArtifact, allows teams to work independently while ensuring that every member is using similar approved packages and versions. This helps avoid dependency issues, improves code reuse, shortens delivery time, and imposes strict governance and security visibility over artifacts.

CodeArtifact uses AWS KMS to secure files and AWS IAM to control the level of access that different users can have. You can push and fetch artifacts from a repository using popular package managers such as *`NuGet`*, *`Maven, Gradle, npm, yarn, pip, and twine`*.

References:
https://docs.aws.amazon.com/codeartifact/latest/ug/getting-started.html
https://aws.amazon.com/blogs/aws/software-package-management-with-aws-codeartifact/

Amazon CodeGuru

Amazon CodeGuru is a service that can analyze Python and Java code using automated reasoning and machine learning to provide fixes, recommendations for improving code quality, and vulnerability detection.

Amazon CodeGuru has two components:

- Amazon CodeGuru Reviewer
    - Detects potential issues in code such as typos, missing API calls, inconsistent logging, missing exceptions, and so on, as well as security flaws such as hardcoded passwords and database connection strings.

    - You can integrate CodeGuru into existing pipelines to automatically evaluate incremental code changes when a push or pull request is issued.
    - CodeGuru supports GitHub and BitBucket repositories.
    - CodeGuru generates recommendations based on common scenarios. Take note that there will be times when you will not agree with a suggestion because it is not applicable to your situation. If this occurs, you may provide feedback to CodeGuru Reviewer. AWS uses user feedback to improve CodeGuru so that it can provide more accurate reports.

- Amazon CodeGuru Profiler
    - Collects performance data from your application and shows a visualization of how your code is executing.
    - CodeGuru Profiler suggests ways to improve your code in order to address issues such as CPU bottlenecks and high latency processes.
    - Supports environments hosted on EC2, EKS, ECS, Fargate, Lambda, or on-premises.

References:
https://docs.aws.amazon.com/codeguru/latest/reviewer-ug/welcome.html
https://aws.amazon.com/blogs/devops/detect-python-and-java-code-security-vulnerabilities-with-codeguru-reviewer/
https://aws.amazon.com/blogs/machine-learning/optimizing-application-performance-with-amazon-codeguru-profiler/

AWS Key Management Service (KMS)

AWS KMS Key

The KMS Key is the most basic resource in AWS KMS. A KMS key includes metadata, such as the key ID, creation date, description, and key state. The KMS key also contains the key material used to encrypt and decrypt data. AWS KMS has two types of keys:

1. **Symmetric** - a 256-bit key that is used for encryption and decryption.
2. **Asymmetric** - an RSA key pair that is used for encryption and decryption or signing and verification (but not both), or an elliptic curve (ECC) key pair that is used for signing and verification.

Three types of KMS Key:

1. **Customer-managed keys** - You have complete control over these KMS keys. You are in charge of key management, which includes key policy configuration, key material rotation, and key lifecycle management.
2. **AWS-managed keys**- These are KMS keys in your account that are created, managed, and used on your behalf by an AWS service. You cannot manage, rotate, and change the key policies of AWS-managed keys. You also cannot use them in cryptographic operations directly; the service that creates them uses them on your behalf.
3. **AWS-owned keys**- These are KMS keys that an AWS service creates, owns, and manages for use in multiple AWS accounts. You cannot view, use, track, or audit these KMS keys.

Type of KMS key	Can view metadata	Can manage KMS Key	Used only for your AWS account	Automatic rotation	Pricing
Customer managed key	Yes	Yes	Yes	Optional.	Monthly fee (pro-rated hourly), per use fee
AWS-managed key	Yes	No	Yes	Required. Every **365** days	No fixed monthly fee, per use fee
AWS-owned key	No	No	No	Varies	No fees

KMS generates key material for all KMS keys that it creates by default. Default key materials cannot be extracted, exported, viewed, or managed. A key material by itself cannot be deleted; if you want to get rid of it you must delete the KMS Key it's associated with. You can also import your own key material into a customer-managed KMS key. Storing KMS keys in AWS CloudHSM is also possible. Consider this option if you need to secure keys in a single-tenant HSM that is FIPS 140-2 level 3 validated.

References:
https://docs.aws.amazon.com/kms/latest/developerguide/concepts.html#master_keys
https://tutorialsdojo.com/aws-key-management-service-aws-kms/

Envelope Encryption

A KMS key is capable of encrypting data up to 4 KB in size, making it not suitable for encryption of large data sizes. To encrypt data more than 4KB, as you most likely will, you'll need to use a process called envelope encryption.

Envelope encryption is not specific to KMS. But when we talk about envelope encryption in the context of KMS, it refers to the process of generating a data key using a KMS key. Then, you use that data key outside of KMS to encrypt a file or another data key; the depth of encryption is up to you. Multiple layers of security can be achieved by generating a secondary data key under the top-level data key, then encrypting that key under another key, and so on. The catch is, there should always be a master key that remains in plaintext so you can eventually decrypt all data keys. In KMS, the master key is called the KMS key.

As a security best practice, always see to it that the plaintext data key is deleted from memory after use.

References:
https://docs.aws.amazon.com/wellarchitected/latest/financial-services-industry-lens/use-envelope-encryption-with-customer-master-keys.html
https://aws.amazon.com/blogs/security/importance-of-encryption-and-how-aws-can-help/

KMS API

Here's the list of the most commonly used KMS API commands that you'd most likely encounter in the exam:

1. `Encrypt`
    - converts plaintext data into ciphertext using a KMS key.
    - Keep in mind that the Encrypt operation can only encrypt data under 4 KB: useful for generating data keys and encryption of small data such as user ids, passwords, or database credentials.

2. `Decrypt`
    - converts ciphertext data back into its plaintext format using a KMS key.

- can **only** be used on data that have been encrypted with a KMS key.

3. `GenerateDataKey`
    - creates data key using a KMS key and returns two versions of it: a plaintext and a ciphertext data key.
    - the plaintext data key is what you use to encrypt files.
    - you cannot use the Encrypt API to encrypt data using the plaintext data key. Typically, you'll use the plaintext data key to perform cryptographic operations with the help of a client-side library like OpenSSL.

4. `GenerateDataKeyWithoutPlaintext`
    - creates data key using a KMS key and returns only the encrypted version of it.
    - to encrypt/decrypt files, convert the ciphertext data key first into plaintext using the Decrypt API.

References:
https://docs.aws.amazon.com/cli/latest/reference/kms/index.html
https://docs.aws.amazon.com/kms/latest/developerguide/programming-top.html

Amazon CloudFront

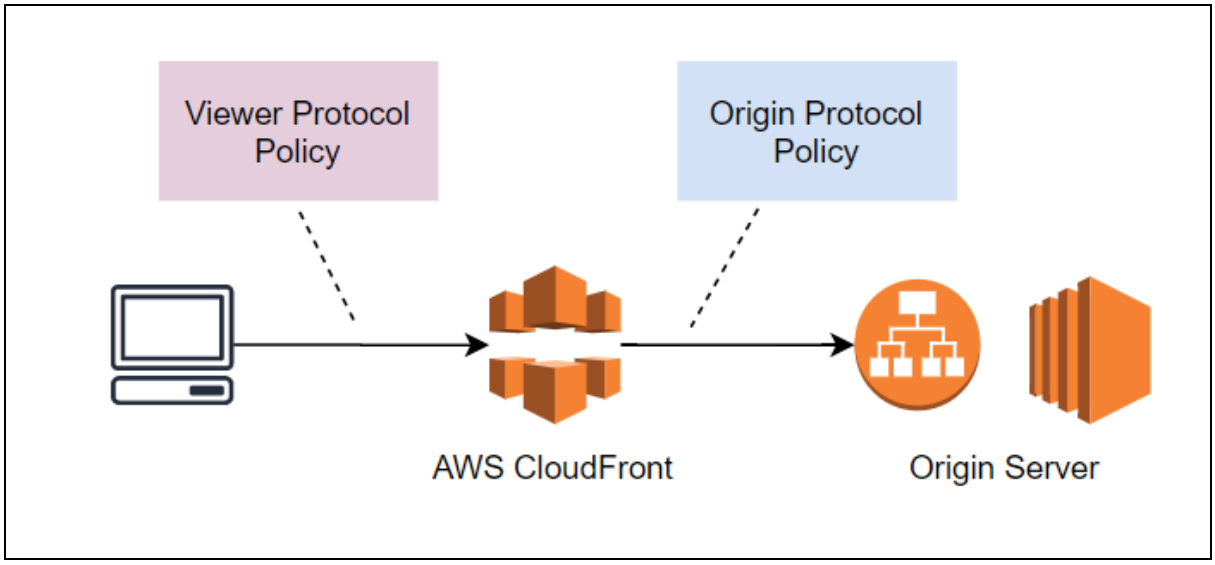
Amazon CloudFront has three basic components: the origin, the distribution, and the viewer. The origin refers to the source of the content, where it is generated or stored. The distribution is a CloudFront resource that controls how the content is delivered to your users, including caching and access controls. The viewer is the end-user or application that accesses and consumes your content.

CloudFront supports the following origin:

- Amazon S3 bucket
- Elastic Load Balancer
- AWS Elemental service
- HTTP server running in an Amazon EC2 instance
- On-premises server

Viewer Protocol Policy vs Origin Protocol Policy

In Amazon CloudFront, you can configure how CloudFront communicates with the viewer and origin server, respectively thru the Viewer Protocol Policy and Origin Protocol Policy.



Viewer Protocol Policy

- Describes the connection protocol and policy **between the client (viewer) and CloudFront**.
- It supports
    - HTTP and HTTPS
    - Redirects HTTP to HTTPS
    - HTTPS Only.

Origin Protocol Policy

- describes the connection protocol policy **between CloudFront and the Origin Server**.
- It supports
    - HTTP Only
    - HTTPS Only
    - Match Viewer
- Selecting **`Match viewer`** will prompt CloudFront tol use the same protocol as the viewer. This can be useful when you have a mix of HTTP and HTTPS content, and you want to avoid protocol mismatches that could affect the delivery of your content.
- Origin Protocol Policy **does not** apply at S3 website endpoints. This is because S3 website endpoints only support HTTP connections. In order to enforce HTTPS communication between CloudFront and S3, you must change Viewer Protocol Policy setting to "HTTP and HTTPS" or "Redirect HTTP to HTTPS."

Custom Domain Name

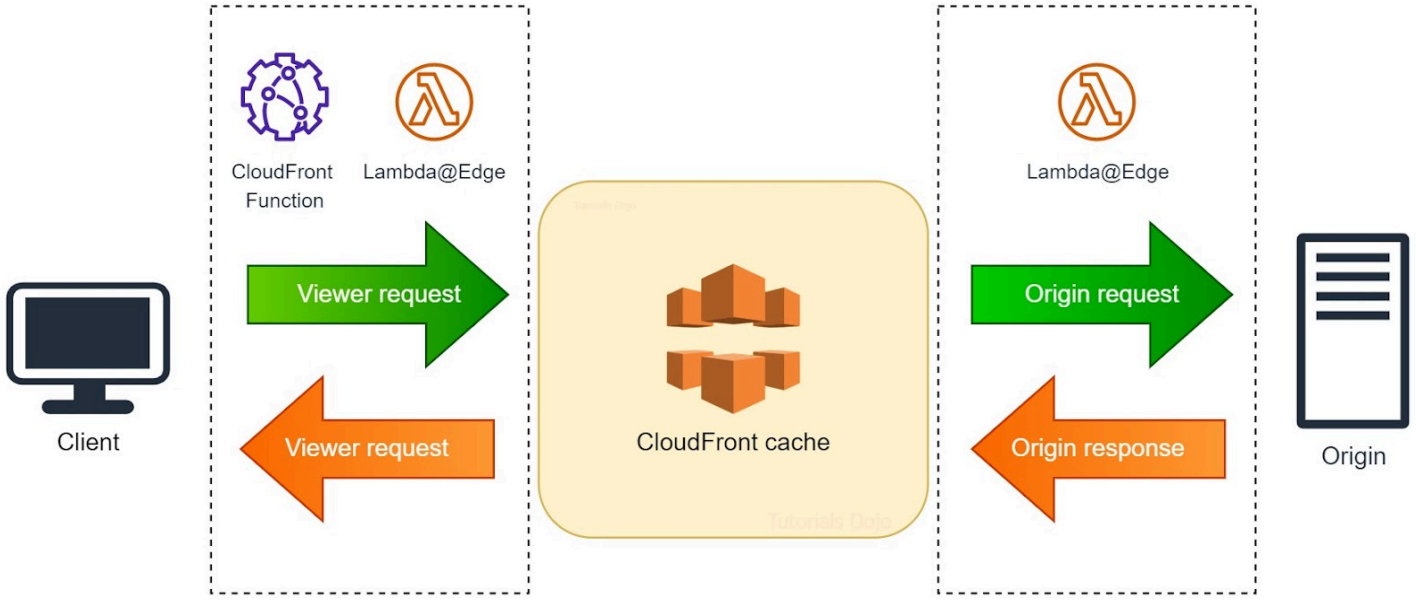
CloudFront uses its own SSL certificate to provide a default HTTPS CloudFront domain name in the <*cloudfront-distribution-id.cloudfront.net*> format. If you wish to use a custom domain name, you must use your own SSL certificate. Make sure that the domain listed on the certificate matches the alternate domain name

that you want to use. You can request one from the ACM in the N. Virginia Region or use an imported certificate stored on IAM.

Reference:
https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/using-https-cloudfront-to-custom-origin.html

CloudFront event triggers

CloudFront event triggers can be used to run certain actions to customize the behavior of your CloudFront distribution and create more personalized experiences for your viewers.



There are four trigger events in CloudFront:

1. Viewer request event - occurs when CloudFront receives a request from an end-user. You can use this event to run custom actions such as user authentication.

   Use when:
   - Modifying a cache key such as HTTP headers, cookies, query string parameters.
   - Generating custom responses that will not be cached.

2. Origin request event - occurs when there is a cache miss, and CloudFront needs to retrieve the requested content from the origin server. You may take actions such as modifying the request headers or requesting a specific version of the content.

Use when:

- Executing actions on a cache miss.
- Dynamically selecting the origin from which CloudFront will retrieve the requested content.
- Rewriting the URL to the origin
- Generating custom responses that you intend to cache.

3. Origin response event - occurs after CloudFront receives a response from the origin server but before it caches the response in the edge location. During this event, you can configure CloudFront to trigger certain actions, such as modifying the response headers or the response body before it is stored in the edge location.

   Use when:

   - Modifying the response before CloudFront caches it

4. Viewer response event - occurs before CloudFront returns the requested content to the user. You can use this event to trigger actions, such as logging the response or modifying the response headers before delivering it to the user.

   Use when:

   - Modifying the response without caching the result

References:
https://docs.amazonaws.cn/en_us/AmazonCloudFront/latest/DeveloperGuide/lambda-cloudfront-trigger-events.html
https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/lambda-how-to-choose-event.html

Lambda@Edge vs CloudFront functions

Lambda@Edge

Lambda@Edge lets you run Lambda functions closer to your end-users is the response to all CloudFront trigger events. When using Lambda@Edge, your Lambda functions are executed at CloudFront edge locations, which are distributed globally to provide low-latency access to your content. This means that you can process and modify requests and responses closer to the end users, resulting in faster content delivery and a better user experience.

Lambda@Edge functions **must** be created in the N.Virginia (us-east-1) region. Additionally, the runtime for Lambda@Edge functions is limited to Python and Node.js only.

Lambda@Edge is well-suited, particularly for dynamic content processing. For example, you can use Lambda@Edge to perform on-the-fly image processing jobs, such as creating thumbnails of images or converting an image format to another. This can help to reduce load times and improve the user experience, while also minimizing the need for pre-processing of all possible image sizes.

CloudFront function

CloudFront function enables you to run lightweight JavaScript functions at the edge of the CloudFront network. Unlike Lambda@Edge, CloudFront functions **can only be triggered at viewer requests and viewer response events**.

While CloudFront Functions offer faster startup and execution times compared to Lambda@Edge, its processing power is quite restricted. Also, it's limited in what it can do. For example, it doesn't have access to a a file system, meaning you can't perform certain tasks like reading or writing files, making network requests to external services, or directly accessing databases.

That being said, if you're looking to run simple tasks like header manipulation, URL redirects, or request authorization, you should consider using CloudFront Functions since it's much cheaper than Lambda@Edge.

References:
https://aws.amazon.com/cloudfront/features
https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/lambda-at-the-edge.html
https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/high_availability_origin_failover.html
https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/cloudfront-functions.html
https://aws.amazon.com/blogs/aws/introducing-cloudfront-functions-run-your-code-at-the-edge-with-low-latency-at-any-scale/

Amazon S3

Object Storage Classes

Amazon S3 Standard

This class is primarily used for storing frequently accessed data (hot data). It offers high durability, availability, and performance object storage for various workloads. It replicates data to three or more Availability Zones to achieve high availability of 99.99%. A common application for this class is the storage of static web assets such as Javascript files, CSS, or images for distribution. In terms of cost, the S3 Standard class is the most expensive among all other classes, so make sure that you properly review your storage requirements before using this type.

Amazon S3 Standard-Infrequent Access (IA)

As the name implies, this storage class is typically used for storing data that are retrieved less frequently but still require rapid access when needed. The S3 Standard-IA has the same durability and availability that the Standard Class offers, however, it is subjected to a minimum storage duration charge of 30 days.

Amazon S3 One Zone-Infrequent Access

This storage class is cheaper than Standard-IA because it stores data to a single availability zone only. As a best practice, you shouldn't be using this storage class for backup copies or any important data that cannot be easily reproduced.

Amazon S3 Intelligent-Tiering

This storage class is designed to optimize costs by automatically moving data to the most cost-effective class without any operational overhead on your part. The S3 Intelligent-Tiering storage class is suitable in cases where it's not clear how frequently your data is going to be accessed. Picking the right storage class to save money requires some insight into your object's access pattern. This means you must gather information about your data access patterns, as well as learn to interpret and analyze them to come up with the best decision possible. If data analysis is not your cup of tea, or if you want to save time, S3 Intelligent-Tiering is the best option for you.

Amazon S3 Express One Zone

This storage class is designed specifically for your applications that need ultra-low latency access, perfect for scenarios where rapid data retrieval is paramount. With Amazon S3 Express One Zone, your data is stored in a

single Availability Zone. This not only simplifies your data storage but also cuts your access costs by about 50% compared to the standard multi-zone S3 storage options. It's especially beneficial if your data has predictable access patterns and can be easily recreated if lost. If your top priority is the fastest possible access to your data without needing the resilience across multiple zones offered by other S3 storage classes, then S3 Express One Zone is an excellent choice for you. This storage class is ideal for your time-sensitive applications like real-time analytics, gaming, and machine learning, where every millisecond counts and can significantly impact performance.

Amazon S3 Glacier Flexible Retrieval

Amazon S3 Glacier Flexible Retrieval is a low-cost archiving solution optimized for backup, offsite data storage, and disaster recovery. It is suitable for rarely accessed data (typically once or twice a year) and does not require the fast retrieval speed of the Standard class. It also replicates your data to three or more Availability Zones and offers a data availability of 99.99%. S3 Glacier Flexible Retrieval offers three retrieval options, each with a different price and retrieval window ranging from minutes to hours.

Three types of Archive Retrieval Options:

1. *Standard retrievals (Default)*- allows you to access your glacier archives within **3–5 hours**.

2. *Expedited retrievals* - allows you to retrieve your archived data within just **1 - 5 minutes**, as long as its file size doesn't exceed 250 MB. This option has the highest data retrieval fee and is useful in situations where urgent requests for archive retrieval are required. You can also set up a provisioned capacity to ensure that the retrieval capacity of your Expedited retrievals is available as needed.

3. *Bulk retrievals* - Bulk retrievals are free; all you have to pay for is storage. This option is great for retrieving large amounts of data (petabytes) within **5–12 hours.**

Amazon S3 Glacier Instant Retrieval

Amazon S3 Glacier Instant Retrieval is an archiving solution that can deliver retrieval times in **milliseconds**. This option is recommended for long-lived user-generated data that needs immediate access, such as medical images, media assets, or genomics data. S3 Glacier Instant Retrieval offers the same throughput and access times of S3 Standard and S3 Standard-IA. For this reason, S3 Glacier Instant Retrieval is a much cheaper alternative over S3 Standard IA for data that is accessed only a few times a year, such as once every quarter.

Amazon S3 Glacier Deep Archive

This is Amazon S3's cheapest storage class. It is designed for storing data that requires long retention periods (typically 10 years or more) to meet regulatory compliance requirements. S3 Deep Archive replicates data across three or more Availability Zones and guarantees data availability of 99.99%. However, it's significantly less expensive and has a longer retrieval time. It has a minimum storage duration charge of 180 days, which is nearly equivalent to 6 months (the longest among other storage classes). For this reason, the data that you store here should be rarely accessed and does not have a strict retrieval time. S3 Glacier Deep Archive offers two retrieval options ranging from 12 hours (Standard) to 48 hours (Extended) (Bulk).

Minimum Storage Duration

Amazon S3 imposes a minimum storage duration to objects stored in all classes except for S3 Standard. If you delete, overwrite, or transition objects to a different storage class before the end of the minimum duration, you'd still be charged for the entire duration.

For example, uploading an object to S3 Standard-IA will cost you the entire minimum duration of 30 days, regardless of what you do with the object. Let's say that you deleted it after 5 days in the hopes of saving storage costs, the effective duration for which you're billed would still be 30 days, even if you didn't store your data for that long. On top of this, you pay a prorated charge equal to the storage charge for the remaining days.

Amazon S3 Encryption

1. Server-side encryption-KMS (SSE-KMS):
    - Amazon S3 facilitates the encryption for you
    - You can use the AWS-Managed KMS key for S3 or a customer-managed KMS key to encrypt S3 objects.
    - To encrypt with a customer-managed KMS key, make sure to add the **`kms:GenerateDataKey`** and **`kms:Decrypt`** permissions to its key policy.
    - To enforce the encryption of objects as they get uploaded, include a condition in your bucket policy that denies requests that do not include the **`x-amz-server-side-encryption`** header with **`aws:kms`** as the value.

2. Server-side encryption-S3 (SSE-S3)
    - Amazon S3 facilitates the encryption for you
    - Amazon S3 uses an AES-256 KMS key that it owns and manages for encryption.
    - To enforce the encryption of objects, include a condition in your bucket policy that denies requests that do not include the **`x-amz-server-side-encryption`** header with **`AES256`** as the value.

3. Server-side encryption-Customer-provided encryption keys (SSE-C)
    - Amazon S3 facilitates the encryption but you have to provide the encryption key along with the data as part of your request.
    - To retrieve encrypted data, you must include the same encryption key as part of your request so Amazon S3 can decrypt the data.
    - Amazon S3 does not store your encryption keys. You're responsible for managing it, meaning, if you lose your encryption keys, your data is as good as gone.
    - You must include the following headers in your request:
        - `x-amz-server-side-encryption-customer-algorithm`
        - `x-amz-server-side-encryption-customer-key`
        - `x-amz-server-side-encryption-customer-key-MD5`
    - SSE-C only accepts AES-256 keys and rejects requests made over HTTP. It only accepts HTTPS requests, thus, it enforces both encryptions in transit and encryption at rest.

4. Client-side encryption
    - Data is encrypted by the sender before it reaches Amazon S3
    - The management of keys and encryption/decryption of data are the sender's responsibility.
    - You may use a client-side cryptographic library like OpenSSL or AWS Encryption SDK to help you with the encryption and decryption of data as you send and retrieve them.

S3 Event Notifications

S3 Event notification allows you to fire up 'triggers' when certain events occur in your S3 bucket. You can configure S3 to send notifications to three different destinations:

- **Lambda function** - when an object is uploaded to your S3 bucket, you can use S3 Event Notifications to publish its event data to a Lambda function. This allows you to automate processes on the object, such as image resizing or creating thumbnails.
- **SNS topic** - you can opt to subscribe to an SNS topic and get notified when an S3 object is created or deleted from your bucket.
- **SQS queue** - you may also send events to an SQS queue and have a Lambda function to process them. This is useful when you want to process events in batches rather than running the function at every event. By using an SQS queue as a buffer for a consumer, like the Lambda function, you can ensure that events are processed efficiently and at the desired rate.

You can create a notification for the following events:

- Object creation
- Object deletion

- Restore object events
- Reduced Redundancy Storage (RRS) object lost events
- Replication events
- S3 Lifecycle expiration events
- S3 Lifecycle transition events
- S3 Intelligent-Tiering automatic archival events
- Object tagging events
- Object ACL PUT events

If none of these events match your use case, you can also enable the sending of notifications to Amazon EventBridge. This will cause S3 to send **all events** to EventBridge, allowing you to process them using custom code or third-party integrations.

Filtering

Optionally, you can use prefixes and suffixes to filter the objects for which you want to receive notifications. For example, let's say you have an S3 bucket containing static assets for a social media app, and you want to receive notifications only when avatar images are uploaded. You could create an S3 Event Notification with a filter that matches the prefix **`avatar/`**. This means that an object with the key **`avatar/user1.jpg`** or **`avatar/cat.png`** will trigger a notification, but an object with the key **`icons/icon123.jpg`** will not.

Note that you can't have overlapping suffixes in two rules if the prefixes are overlapping for the same event type. Suppose you have two S3 Event Notification rules configured. The first rule is set up to trigger a Lambda function whenever a new object is created with a prefix of **`docs/`** and a suffix of **.pdf**. The second rule is triggers a Lambda function whenever a new object is created with a prefix of **`docs/payslip/`** and a suffix of **.pdf**. Now, suppose a new PDF file is uploaded to the **`docs/payslip/`** folder in the S3 bucket. This object key matches both rules, because it has a prefix of documents/ and a suffix of .pdf. As a result, S3 will throw a **`Configuration is ambiguously defined`** error because it cannot determine which Lambda function should be triggered for this event.

References:
https://docs.aws.amazon.com/AmazonS3/latest/userguide/NotificationHowTo.html
https://aws.amazon.com/premiumsupport/knowledge-center/lambda-s3-event-configuration-error/

S3 Object Lambda

S3 Object Lambda enables you to add your own code to process data as it is being retrieved from S3 objects using Lambda functions. You can transform or augment data on-the-fly without having to modify your source data. One popular use case for S3 Object Lambda is redacting Personally Identifiable Information (PII) from

objects stored in S3. PII is any information that can be used to identify an individual, such as their name, email address, phone number, or social security number. With S3 Object Lambda, you no longer have to modify the original object or create a separate copy of it with the PII removed.

Aside from removing PIIs, S3 Object Lambda is also suited for applications such as:

- Converting across data formats, such as converting XML to JSON.
- Augmenting data with information from other services or databases.
- Compressing or decompressing files as they are being downloaded.
- Resizing and watermarking images on the fly using caller-specific details, such as the user who requested the object.
- Implementing custom authorization rules to access data.

How is S3 Object Lambda different from S3 Event notification + Lambda function?

In the S3 Event Notifications + Lambda function pattern, the Lambda function is triggered after an object event occurs. On the other hand, in S3 Object Lambda, the Lambda function is executed as part of the retrieval process, which means it happens in real time and is applied to the data as it is being retrieved.

References:
https://aws.amazon.com/s3/features/object-lambda/
https://aws.amazon.com/blogs/aws/introducing-amazon-s3-object-lambda-use-your-code-to-process-data-as-it-is-being-retrieved-from-s3/

Other key S3 bucket features

- Lifecycle policies — used to automatically move objects from one storage class to another in an effort to reduce storage cost or to archive an object. Lifecycle policies can also be used to expire versioned objects and permanently delete them from your bucket. When creating a lifecycle policy, you configure two parameters for each transition or deletion action:

    1. Whether the policy should apply to all objects in the bucket or only a group of objects with matching prefix

    2. The number of days after object creation before the action is applied. To migrate objects from Standard to Standard-IA, One-Zone-IA, or S3 Intelligent-Tiering, a minimum of 30 days for transition is required.

- Multipart Upload — When using the aws s3 cp to copy/upload large files, S3 implicitly uses the multipart upload feature. The multipart upload works by splitting up data into individual parts that are uploaded in parallel, making the upload speed faster. Amazon S3 then reconstructs the uploaded parts into the original data. When uploading or retrieving objects encrypted using SSE-KMS, you must grant your KMS key the **`kms:Decrypt`** and **`kms:GenerateDataKey*`** permissions since parts that get uploaded to S3 are server-side encrypted. Amazon S3 needs to decrypt the parts first before they can be reassembled into the original object.

- Static Web Hosting — An S3 bucket can be configured to host a simple static website. You simply have to upload your web page's assets (Javascript, CSS, image, etc.) to the bucket and enable web hosting on the S3 console. Make sure to set your objects as publicly available as well. Amazon S3 will issue a website endpoint for your bucket. This endpoint, however, does not support HTTPS. A workaround for this is to put the S3 bucket behind a CloudFront distribution and encrypt the connection between client and CloudFront using HTTPS. The connection between CloudFront and S3 stays unencrypted. You may also give your website a custom domain name by setting up a CNAME record in Route 53 pointing to your S3 bucket URL. For this matter, the domain name and the name of the S3 bucket must be an exact match.

- Versioning — Versioning lets you keep a copy of an object whenever it is overwritten as its *versions*. You can preserve and restore to a specific version of an object if you need to. This feature also protects you from accidental deletions since versioning places deletion markers on an object version to mark it as removed rather than permanently deleting it from your S3 bucket. By default, versioning is disabled on buckets, and you must explicitly enable it. Once it has been enabled, it cannot be disabled, but it can be suspended. When you suspend versioning, any future updates on your objects will not create a new version, but existing versions will still be retained. Since a version of an object also takes up storage space, versioning will incur additional S3 costs, so only use this feature if you need it.

- Cross-origin Resource Sharing (CORS) — CORS is a way for client applications that are loaded in one domain to interact with resources in a different domain. When this feature is disabled, requests directed to a different domain will not work properly. If your S3 bucket is used for web hosting, verify if you need to enable CORS. To configure your bucket to allow cross-origin requests, you create a CORS configuration document. This is a document with rules that identify the origins that you will allow to access your bucket, the operations (HTTP methods) that will support for each origin, and other operation-specific information.

- Presigned URLs - By default, all S3 buckets and objects are private and can only be accessed by the object owner. Object owners can share objects with other users or enable users to upload objects to their S3 buckets using a presigned URL. A presigned URL grants others time-limited permission to download or upload objects from the owner's S3 buckets. When object owners create presigned URLs,

they need to specify their security credentials, the bucket name and object key, the HTTP method (GET to download the object), and the expiration date and time. The bucket owner then shares these URLs with those who need access to the objects or the buckets. A presigned URL can be used many times, as long as it has not expired.

References:
https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html
https://tutorialsdojo.com/amazon-s3/

Amazon Elastic Block Store (EBS)

Amazon Elastic Block Store (EBS) is a high-performance block storage, and it offers scalable resources to work with Amazon EC2 for both throughput and transaction-intensive workloads at any scale. A basic EBS volume functions like a raw, unformatted block device, which you can attach to one or more EC2 instances.

Integration with AWS Services

You can manage Amazon EBS using various tools, including the Amazon EC2 Console for direct web-based management of volumes and snapshots, the AWS Command Line Interface for command-line operations, and AWS Tools for PowerShell for scripting. Additionally, AWS CloudFormation allows for the automation of EBS resources through templates. At the same time, the Amazon EC2 Query API and AWS SDKs provide programmatic access to manage and integrate EBS with other AWS services.

Key Features of Amazon EBS:

- Multiple Volume Type - You can choose from different types of EBS volumes based on your specific needs: SSD-backed volumes are ideal for interactive applications that demand fast data access, while HDD-backed volumes are suited for applications that require extensive sequential read and write operations.
- Scalability and Performance Tuning - You can scale your EBS volumes up or down, and adjust performance as your requirements change, without any downtime.
- Backup and Recovery - With EBS, creating backups is straightforward using snapshots. These snapshots capture a point-in-time state of a volume and can be used to restore volumes or even copy data across different AWS accounts or geographical regions.
- High Durability and Availability - To protect against data loss, EBS automatically replicates data within its Availability Zone. This replication ensures that your data is highly durable and available.

- Encryption and Security - EBS provides encryption options for both the volumes and their snapshots, using keys managed through AWS KMS. This encryption secures your data at rest and in transit from EC2 instances to EBS storage.
- Data Archiving - For long-term data retention, especially for regulatory compliance, EBS offers a Snapshot Archive service that is cost-effective and retains data for extended periods.

Reference
https://docs.aws.amazon.com/ebs/latest/userguide/what-is-ebs.html
https://docs.aws.amazon.com/ebs/latest/userguide/ebs-volume-types.html

AWS X-Ray

AWS X-Ray is a distributed tracing service that can help you analyze and debug potential issues in a distributed application, such as those built using a microservices architecture. Since microservices communicate over APIs, one of its implementation challenges is dealing with the additional network latency between each service. So, in addition to monitoring performance metrics and application logs, as you're used to with monolithic applications, devising a way to track latency across your application's components to spot disruptions, as they occur, is needed as well.

With AWS X-ray, you can easily determine which part of your application is causing an issue or a performance bottleneck. The service map feature of AWS X-ray allows you to see user requests as they travel through the components of your application. The service map is a visual representation of the connections between your application's services which allows you to see information such as the average latency and failure rates for all services that are part of your application.

End-to-end tracing is also supported by AWS X-ray. This is useful if you want to trace the path of a single request across your application nodes. AWS X-ray can be used on applications that run on Amazon EC2, Amazon Elastic Container Service, ElasticBeanstalk, or even AWS Lambda.

References:
https://aws.amazon.com/xray/
https://aws.amazon.com/blogs/aws/aws-x-ray-see-inside-of-your-distributed-application/

Key concepts

Segment

The applications that you're tracing send data to AWS X-Ray in the form of segments. A *segment* is a JSON document that contains information about a request, such as how quickly the work was completed, the request method, the client address, the user agent, and so on. Segments generated from common requests are group together in a *trace*, and are given a trace id. The trace data is what X-Ray uses to generate a service graph that allows you to visually interpret end-to-end requests across your application's components.

Subsegment

Subsegments allow you to trace specific functions in your code. This can provide more granular timing information as well as details about the downstream calls made by your application.

Below is an example of how subsegments are used with the X-Ray SDK for Python:

```
from aws_xray_sdk.core import xray_recorder

@xray_recorder.capture('register user')
def register_user():
    # Do something here

register_user()
```

In the example, we imported the X-Ray SDK for Python into the application code. Then we placed the `@xray_recorder.capture` decorator before the `register_user` function, which is the function the we want to instrument. You can either pass a name to the capture function or leave it blank to use the function name. The process of instrumenting codes varies by the programming language that you're using.

X-Ray Daemon

The X-Ray Daemon is an agent that acts as a proxy between the application being traced and AWS X-Ray. The X-Ray SDK does not send segments directly to AWS X-Ray. Behind the scenes, the X-Ray daemon buffers the segments in a queue and uploads them to AWS X-Ray in batches. Please note that the process for enabling X-Ray Daemon varies from one compute platform to another. The table below summarizes how to enable X-Ray daemon on various platforms.

Compute platform	**How to enable X-Ray daemon**
EC2/on-premises	Download and install X-Ray daemon
Lambda Function	Enable the `Active Tracing` option

Elastic Beanstalk	Enable the `X-Ray Daemon` option
Amazon ECS	Use a Docker image that has X-Ray daemon

In addition to enabling X-Ray daemon, you must also grant it the necessary IAM permissions for it to call X-Ray APIs on your behalf. It must have the **`PutTraceSegments`** and **`PutTelemetryRecords`** permissions at the very least to publish segments. These permissions can be granted in different ways. For example, you can attach these permissions to an IAM role that your application can assume. If the application is hosted on EC2, you can attach the role to EC2 instance's instance profile. For Lambda functions, the permissions must be attached to its execution role.

Other features:

Filter expressions - used to search traces related to specific paths or user

Groups - a collection of traces that are defined by a filter expression. Groups are identified by their name or an Amazon Resource Name and contain a filter expression.

Annotations - simple key-value pairs that are indexed for searching traces. Use annotations to record data that you want to use to group traces.

Metadata - key-value pairs with values of any type. Metadata is not indexed. They are meant for recording data that you wish to store in the trace but will not be used for searching traces.

References:
https://docs.aws.amazon.com/xray/latest/devguide/xray-services.html
https://docs.aws.amazon.com/xray/latest/devguide/xray-api.html

Amazon EventBridge

EventBridge acts as a central hub for events, where different sources (e.g., AWS services, custom applications, and third-party) can publish events and subscribe to events of interest. When an event is sent to an EventBridge event bus, it is matched against pre-defined rules and routed to the appropriate subscribers.

These rules enable you to filter events based on attributes such as the source of the event, the content of the event, or a combination of both. Once a rule matches an incoming event, it can trigger one or more targets, such as Lambda functions to take custom actions, SNS topics to get notified, or AWS Step Functions workflows to execute a state machine.

Cross-Account Access to Event Bus

In a multi-account strategy where components of an application are isolated from each other in separate accounts, a central event bus can be used to facilitate communication between these components. By setting up a central event bus in a separate account, you can ensure that communication between the different components of your application is secure and reliable. Each component can publish events to the central event bus, and other components can subscribe to those events as needed.

Suppose you have three AWS accounts: Account A, Account B, and Account C. Account A is the central hub for your event bus, containing the necessary configurations and permissions for the other accounts to communicate with it. Account B contains the backend components of your application, and Account C contains your data analytics components.

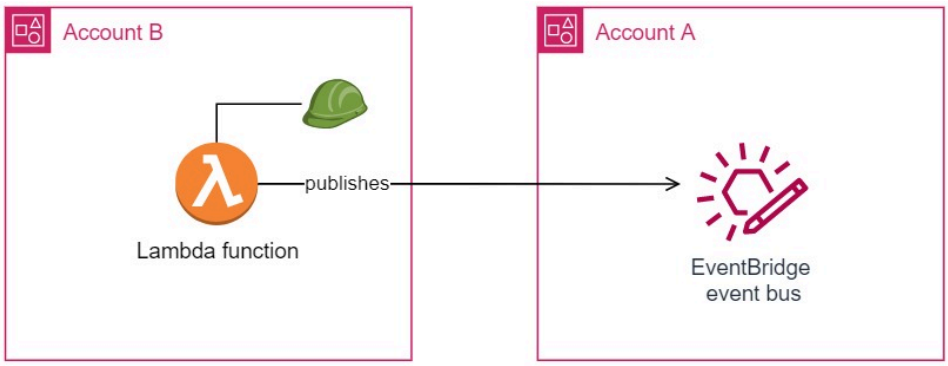
To allow Account B and Account C to publish events to the event bus in Account A, you must first create an event bus in Account A with the following resource-based policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccountBAccess",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::ACCOUNT_B_ID:root"
            },
            "Action": [
                "events:PutEvents",
```

```
                "events:PutRule",
                "events:PutTargets"
            ],
            "Resource": "arn:aws:events:REGION:ACCOUNT_A_ID:event-bus/central-eb"
        },
        {
            "Sid": "AllowAccountCAccess",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::ACCOUNT_C_ID:root"
            },
            "Action": [
                "events:PutEvents",
                "events:PutRule",
                "events:PutTargets"
            ],
            "Resource": "arn:aws:events:REGION:ACCOUNT_A_ID:event-bus/central-eb"
        }
    ]
}
```

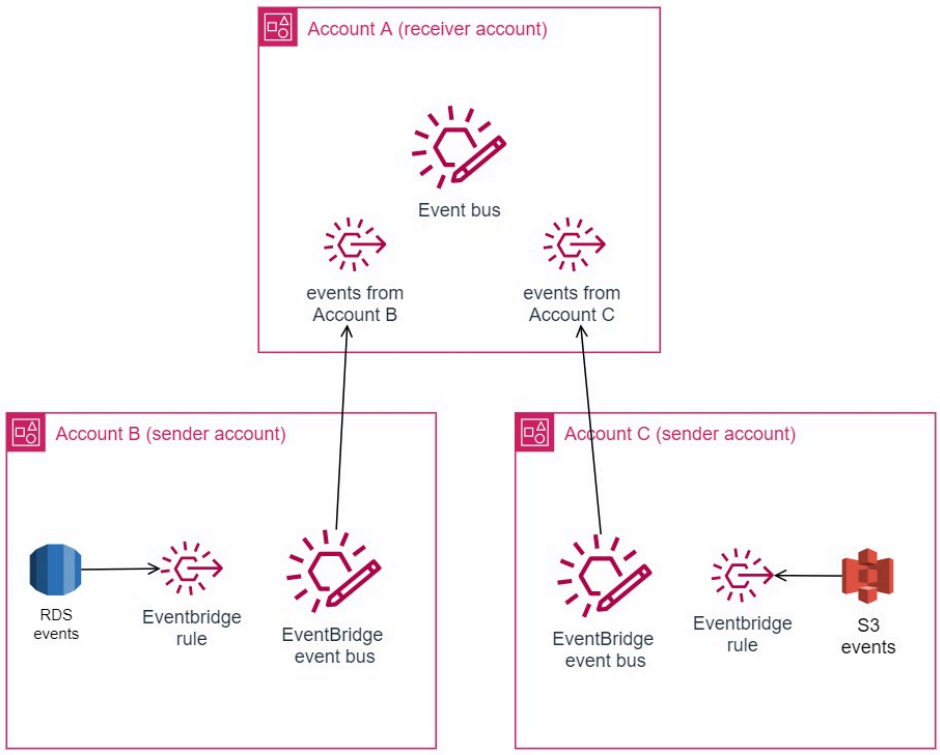
Publishing events directly to the event bus

To send events directly to the event bus, you can use the `PutEvents` API. When using this method, you must have the necessary permissions to publish events to the central event bus in the Account A. For example, a Lambda function in Account B must be given its execution role the necessary permissions to publish events to Account A's event bus.



Using the EventBridge rule to send events to the central event bus

Instead of sending events straight to the central event bus, you can set up EventBridge rules in each sender account (Account B and Account C) that target the event bus in Account A. On the receiver's side, set up one or more rules that match events coming from the sender accounts. Just make sure the sender accounts have the right permissions to send events to Account A's event bus. This strategy is helpful if you want to collect events related to each account and brings them together in one central place.



References:
https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-cross-account.html
https://aws.amazon.com/blogs/compute/simplifying-cross-account-access-with-amazon-eventbridge-resource-policies/

Amazon CloudWatch

Publishing Custom Metrics

There are two ways by which you can push custom metrics to Cloudwatch logs.

1. Using CloudWatch agent

    - The Unified CloudWatch agent is a software application for collecting system-level metrics, custom metrics, and logs from your EC2 instances and on-premises servers. Cloudwatch Metrics can collect different types of data from your AWS resources, however, it does not capture everything. Important internal system-level metrics like memory usage are not directly monitored by Cloudwatch for example. For such cases, installing the Cloudwatch agent into your servers makes monitoring easier.

    - You need to have permission to use the `PutMetricData` operation to publish metrics to Amazon CloudWatch.

    - If you're hosting the agent on an on-premises server, download the AWS CLI and set up your credentials (access key & secret access) using `aws configure`. Make sure your IAM user has permission to use the `PutMetricData`. For EC2 instances, it's recommended to use instance profiles instead of programmatic credentials.

2. Using the `PutMetricData` API

    - Under the hood, the CloudWatch agent uses the `PutMetricData` API to send metrics to AWS CloudWatch. If you don't want to go through the process of installing and configuring the CloudWatch agent, or perhaps your use case simply does not require it, an alternative solution is to just call the `PutMetricData` API directly. You could, for example, write your own monitoring script that periodically sends custom metrics using `PutMetricData`.

Amazon CloudWatch Logs

Amazon CloudWatch Logs is a powerful AWS service designed to monitor, store, and access your log files from various sources, including Amazon EC2 instances, AWS CloudTrail, Route 53, and more. It acts as a centralized repository for all your system, application, and AWS service logs, providing a scalable solution to view, search, and filter your logs for specific error codes, patterns, or fields. Additionally, it also enables for the secure storage of logs for future the review.

Key features of CloudWatch Logs include:

- Two Log Classes: this is for flexibitlity that offers both a cost-effective option for logs accessed infrequently and a full-featured option for logs requiring real-time monitoring or other advanced features.
- Log Data Querying:With CloudWatch Logs Insights, you can perform interactive searches and analyses of your log data using a purpose-built query language, that includes command descriptions, query autocompletion, and log field discovery, which can be useful for your project.
- Live Tail for Debugging: This feature enables real-time troubleshooting by allowing users to view and filter new log events as they are ingested. You can customize filters to meet specific requirements, facilitating efficient troubleshooting and debugging processes.
- Monitoring Log Events: This tool tracks application errors and other important log events, sending notifications when specific criteria are met.
- Auditing and Masking Sensitive Data: Provides data protection policies to audit and mask sensitive data within your logs.
- Flexible Log Retention allows you to adjust the retention policy for each log group from indefinite retention to a specified period, e.g., up to 5 or more years. This feature gives users greater control over the storage duration of log data, accommodating various compliance requirements and storage needs.

CloudWatch Logs integrates with other AWS services such as AWS CloudTrail for monitoring API calls, AWS IAM for secure access control, Amazon Kinesis Data Streams for real-time data intake and aggregation, and AWS Lambda for building responsive applications.

Analyzing log data with CloudWatch Logs Insights

When it comes to analyzing log data, CloudWatch Log Insights stands out as a powerful tool within Amazon CloudWatch, offering the ability to interactively search and analyze log data stored in Amazon CloudWatch Logs through efficient and effective queries. This feature is particularly valuable for addressing operational challenges, as it enables you to identify potential causes of issues and validate deployed fixes seamlessly. With its purpose-built query language, CloudWatch Log Insights allows for the swift analysis of log data, supporting a broad range of log formats, including JSON and automatically discovering fields in logs from AWS services and applications. It is designed for cloud-scale use, requiring no setup or maintenance, and facilitates fast, interactive query execution and visualization, making it an indispensable resource for managing operational health and troubleshooting in the AWS environment.

CloudWatch Embedded Metric Format

The CloudWatch Embedded Metric Format (EMF) is a JSON-based specification that enables the automatic extraction of metric values from logs sent to CloudWatch Logs. This feature allows you to generate custom metrics from your logs without needing separate instrumentation. By structuring your log data according to the

EMF specification, CloudWatch can identify and extract metric values, which you can then visualize and set alarms on.

References:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Install-CloudWatch-Agent.html
https://aws.amazon.com/premiumsupport/knowledge-center/cloudwatch-push-custom-metrics/
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/WhatIsCloudWatchLogs.html
https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch_Embedded_Metric_Format.html

Amazon CloudTrail

AWS CloudTrail is a managed service that you can use to monitor and record the activities that were done by an IAM user, IAM role, or an AWS Service on your AWS account. All actions made in the AWS Management Console, API, or CLI calls are logged as events by CloudTrail. CloudTrail can help you facilitate governance, compliance, operational auditing, and risk auditing of your AWS account.

Say, for example, a member of your organization has been excessively calling the StartInstances API, which launches EC2 instances. With CloudTrail, you can exactly pinpoint who's responsible for launching those instances, determine the time of action, as well as the source IP address where the action originated. You can also integrate CloudTrail with AWS CloudWatch and Amazon SNS for alerting. For instance, you could create a CloudWatch alarm that triggers an SNS topic whenever the number of a certain API event exceeds the threshold that you set.

References:
https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-user-guide.html
https://aws.amazon.com/blogs/mt/category/management-tools/aws-cloudtrail/

Concepts

Events are the record of activity in an AWS account. This activity can be an action taken by a user, role, or service that is monitorable by CloudTrail.

There are three types of events that you can record:

- **Management events**- events that include management operations that have occurred within your AWS account, such as user logins. Management events are enabled by default
- **Data events** - events that include resource operations performed on or within the resource itself, such as *S3 object-level API* activity or *Lambda function execution activity.* Logging data events are charged and, therefore, disabled by default.
- **Insights events** - events that include unusual activity and user behavior in your account.

Event History

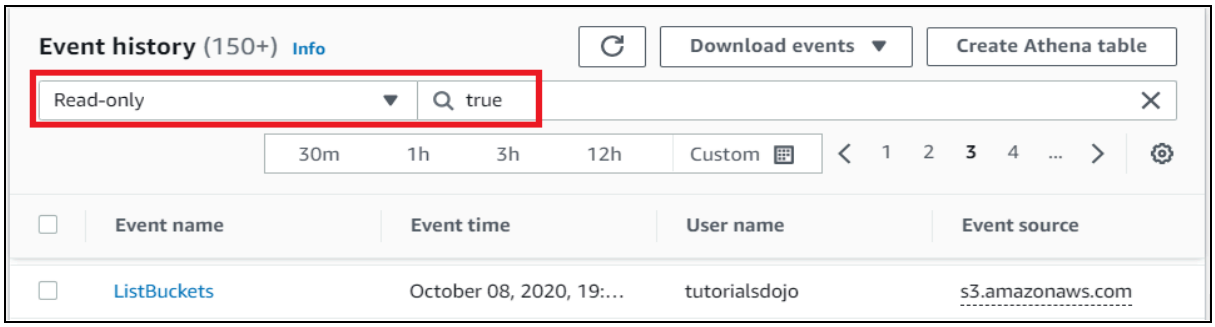
Even without creating a trail, you can still view the history of API activities in your account. You can view events in **Event History** to gain immediate visibility to API activities that occur in your AWS account in the past 90 days. Searching and downloading of event records are also possible.

If you have a high API activity in a day, the number of CloudTrail events can be overwhelming. It would be difficult to look for specific events that may be helpful in troubleshooting a security incident.

In that case, you may use the event filter definitions to only view particular events (in Event history) that you're interested in:

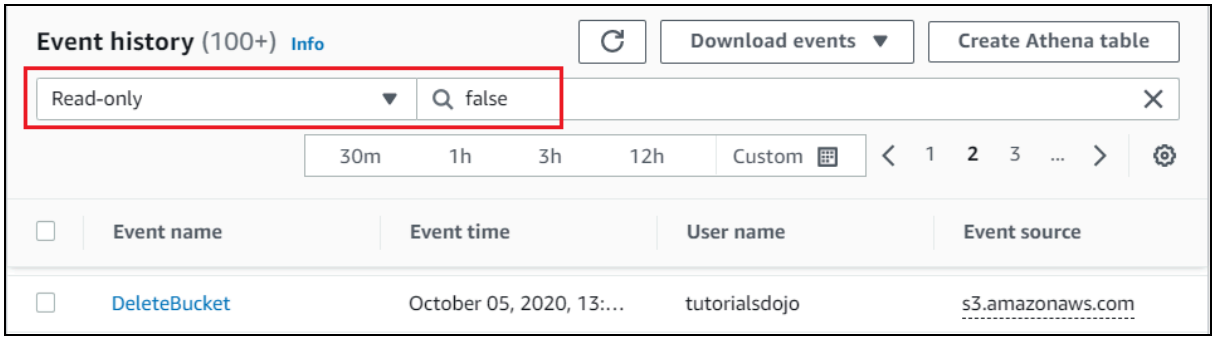
- **Read-only events** refer to API operations that read resources. Read operations are operations that don't cause modifications to a resource. Examples of these events are Amazon S3 *ListBuckets* and Amazon CloudWatch Logs *DescribeMetricsFilters* API actions.

In the CloudTrail console, set the value of Read-only events to "true" to view Read-only events.



- **Write-only events** refer to API operations that modify or may modify resources in your account. Examples are Amazon S3 *DeleteBucket* and Amazon EC2 *TerminateInstances* API operations.

To enable write-only events, set the value of Read-only to "false".



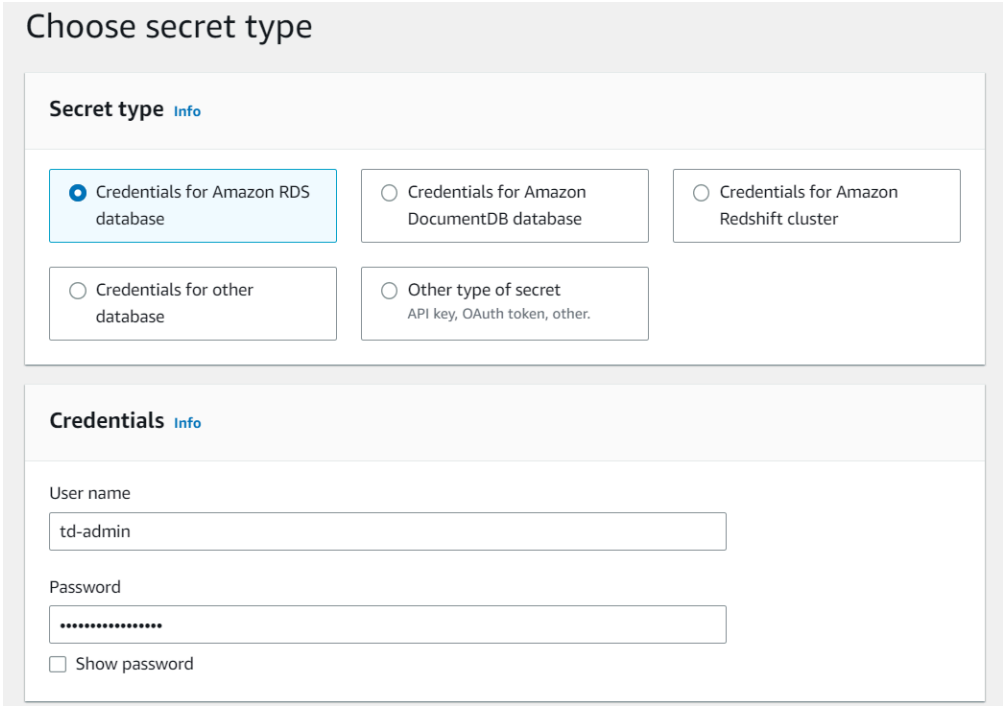
References:
https://aws.amazon.com/cloudtrail/faqs/
https://docs.aws.amazon.com/awscloudtrail/latest/userguide/logging-management-events-with-cloudtrail.html

AWS Secrets Manager

As a best practice, confidential data (e.g., database credentials, API keys, passwords) shouldn't be hardcoded in your application code because it can lead to security breaches. It's often recommended to rotate passwords regularly as well, so in case your code gets leaked online, passwords will not remain valid indefinitely. These tasks can be challenging to implement, more so, when dealing with different applications. Traditionally, developers store app credentials in environment variables or separate config files. While this approach can work well for a single application, it doesn't scale well when dealing with larger or more complex applications. This is especially true for microservices that have multiple components, each with its own set of credentials. This is where AWS Secrets Manager can help you.

Secrets Manager allows you to store and manage application credentials in a centralized location. This eliminates the need for keeping credentials in multiple locations, such as configuration files or environment variables. Data stored in Secrets Manager are called 'secrets', which can be a set of database credentials, a key-value pair (e.g., API keys, tokens), a JSON document, or even a certificate with a long chain of trust.

The main selling point of Secrets Manager is its ability to automate the process of rotating credentials for Amazon RDS, Amazon Redshift, and Amazon DocumentDB. If you're using a non-AWS database service or need to rotate other types of secrets, such as Oauth tokens, you have to implement the logic for rotating credentials using a Lambda function.



Secrets Manager uses a KMS that it manages by default to enforce encryption at rest, thus, storing data in plaintext is not possible. However, you may also specify a custom KMS key.

Retrieving secrets

When you create a secret, you associate a secret name with it. This secret name simply acts as a pointer to the secret that you can refer to in your application code. So, for example, instead of hardcoding database credentials, you can replace those with codes that utilize the `GetSecretValue` API to pull the secrets at runtime. Repeatedly calling this API for authentication can be inefficient because of network overhead and may potentially lead to hitting your API limit faster. As a workaround, Secrets Manager provides a client-side library that you can work with to cache secrets locally instead of retrieving them every time from Secrets Manager.

References:
https://aws.amazon.com/secrets-manager/
https://docs.aws.amazon.com/secretsmanager/latest/userguide/rotate-secrets_turn-on-for-db.html

AWS Systems Manager Parameter Store

Parameter Store allows you to create key-value parameters where you can store application configurations, custom environment variables, product keys, and credentials in a single location. Unlike Secrets Manager, the Parameter Store is designed to cater to a wider range of use cases, not only for passwords or database credentials. If you want to store parameters that don't require encryption, such as AMI IDs, security group IDs, or URLs, then Parameter Store is the better tool for the job since it is a lot cheaper.

Three types of parameters:

1. `String` - can be any block of text that you wish to store unencrypted

2. `StringList` - a type of parameter where strings are separated by commas.

3. `SecureString` - a type of parameter for storing sensitive information, such as passwords or license keys. You can either specify a customer-managed KMS key or an AWS-managed KMS key to encrypt parameters. Unlike Secrets Manager, Parameter Store does not have a built-in password rotation feature so this is something that you have to build on your own.

There are also two tiers that you can choose from when creating a parameter:

1. Standard tier

- Can store up to 10,000 parameters
- Maximum parameter value size can be up to 4KB
- No additional fee

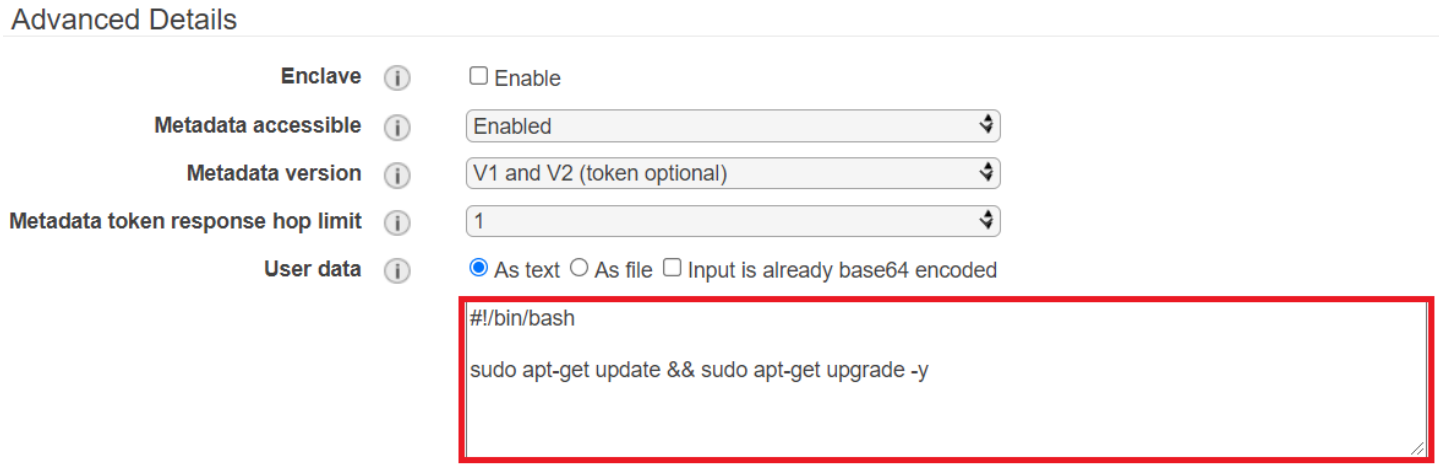
2. Advanced tier
    - Can store up to 10,000 parameters
    - Supports **Parameter policies,** which allows you to set an expiration date or time-to-live to a parameter. You can select from three Parameter policies:
        - `Expiration` - deletes a parameter on a specific date and time
        - `ExpirationNotification` - allows you to receive a notification from Amazon EventBridge when a parameter is about to expire within a specified number of days or hours.
        - `NoChangeNotification` - allows you to receive a notification from Amazon EventBridge when a parameter hasn't been modified for a specified period of time. This is helpful in cases where you have to proactively identify passwords that may need to be updated.
    - Maximum parameter value size can be up to 8KB
    - Comes with fixed monthly charge

References:
https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-parameter-store.html
https://aws.amazon.com/blogs/mt/the-right-way-to-store-secrets-using-parameter-store/

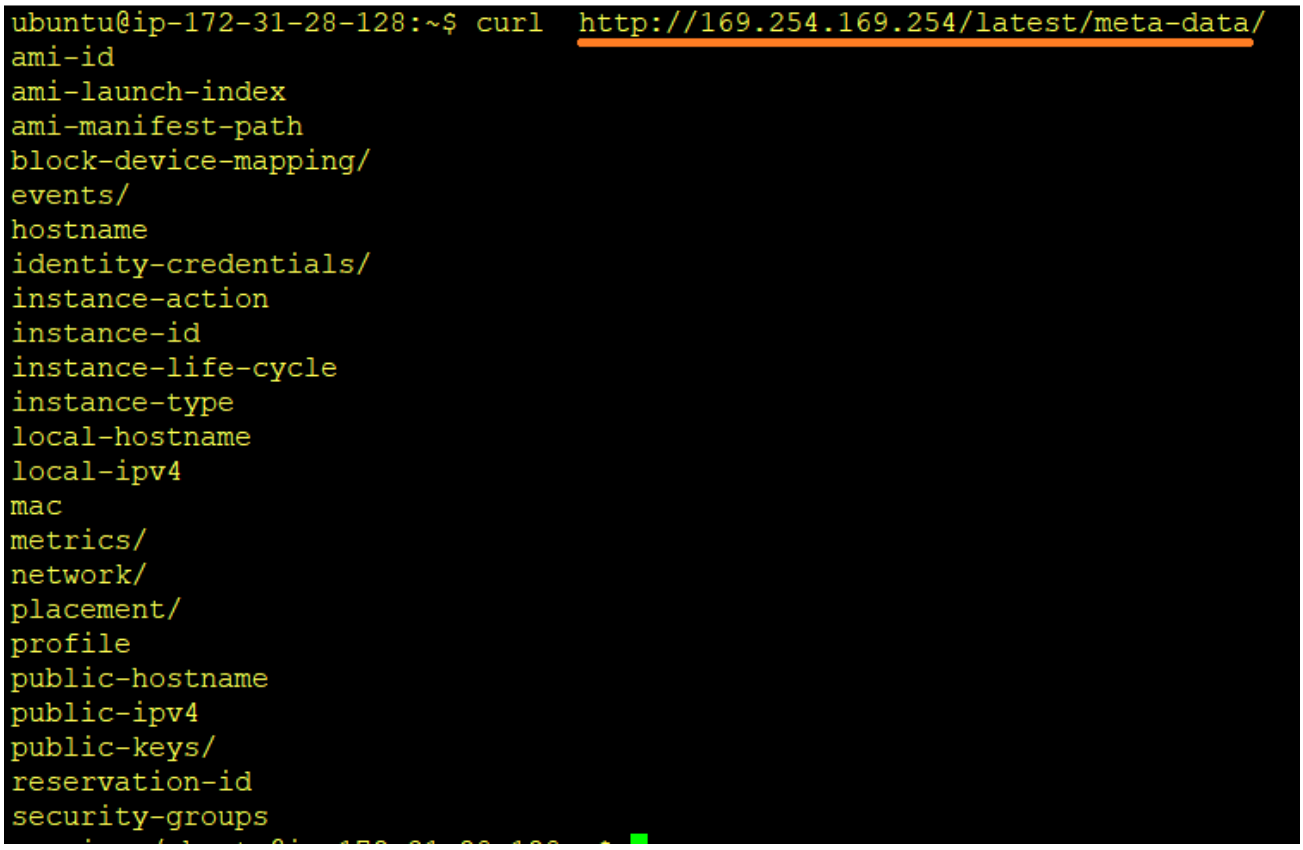
Amazon EC2

Instance User Data vs Instance Metadata

An *instance user data* pertains to a set of commands that are run once an instance is created. You provide the user data during instance creation. You can use this feature to bootstrap mundane configuration tasks that are commonly done while configuring a new environment, such as downloading software updates, installing software packages, or editing server configuration files.



An *instance metadata* is a set of available information that describes your EC2 instance. Metadata is a general term for things that describes a data. For example, in a JPG file, aside from the actual bits that make up the image, other properties like dimension, resolution and timestamp are embedded into the image file as well. The concept is similar to what an instance metadata is. It is a collection of relevant EC2 instance properties, such as its AMI, hostname, public IP address, private IP address, instance type, MAC address, and many more.



Instance metadata allows you to view the associated security groups, security credentials, and IAM Roles of your EC2 instance as well. Keep in mind that you can only fetch the instance metadata on a running instance from the Instance Metadata Service (IMDS) which can be accessed through the **http://169.254.169.254/latest/metadata** link-local address. This address is not publicly accessible and it's exclusive for EC2 instances only.

References:
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instancedata-data-retrieval.html

Auto Scaling Group

Vertical scaling vs Horizontal scaling

If a person is struggling to complete a taxing job, you have two options: replace that person with someone who's more capable of handling the task or add more people to help out. This example illustrates the concept of scaling. The first option represents vertical scaling, while the second one is known as horizontal scaling. Vertical scaling involves replacing your current server with a more powerful one. In AWS, if the load on your instance frequently exceeds its CPU capacity, a solution could be to simply upgrade it to an EC2 instance type with more cores. On the other hand, horizontal scaling is the process of adding more servers to meet the demands of your application.

Horizontal scaling in AWS

The process of scaling horizontally is made easier with the help of the Auto Scaling service, which enables compute capacity to adapt dynamically based on load requirements. An Auto Scaling Group (ASG) is a collection of EC2 instances that are treated as a single unit for scaling and management purposes.

When creating an ASG, you define a launch template. A launch template serves as a blueprint for the instances that are created within an Auto Scaling group, outlining the AMI id, instance type, security settings, and other instance-related parameters. With the launch template in place, you then establish an Auto-scaling policy that sets the maximum and minimum capacity, as well as its scaling behavior. For example, you can set the ASG to scale out whenever its CPU utilization exceeds 60% and scale in if it goes under 30%. Scaling can also be based on a schedule. For instance, you can maintain a specific capacity for normal hours and only add capacity during peak hours.

References:
https://aws.amazon.com/ec2/autoscaling
https://docs.aws.amazon.com/autoscaling/ec2/userguide/AutoScalingGroup.html

EC2 Image Builder

EC2 Image simplifies the process of creating, managing, and deploying custom Amazon Machine Images (AMIs) with specific software configurations, updates, and patches. It automates the entire image creation process, allowing you to maintain consistent and up-to-date AMIs for use across various environments, accounts, and regions.

Common use cases

1. Automated AMI Updates for Auto Scaling Groups

Supposed you have an Auto Scaling group with a Launch Template that references a custom AMI containing your application's software stack. Since EC2 instances in the group are only temporary, they don't receive security patches or software updates during their runtime. If want to apply these updates, you'd have to launch an instance from the current AMI, install the updates there, take an AMI snapshot of that instance, then reconfigure your ASG's launch template to reference the new AMI. Such tasks can be error-prone and difficult to replicate across accounts or regions. To simplify this process, you can use an EC2 Image Builder pipeline that creates a new launch template that references the latest AMI of your application.

2. Reduce boot time of instances launched in an Auto Scaling Group

EC2 Image Builder enables you to reduce or even eliminate the need for bootstrap scripts in your instances. Instead, you can use EC2 Image Builder to define a recipe that includes all the necessary software packages, configurations, and customizations for your AMI. By using a pre-configured AMI for the instances before they are launched, you can minimize the time it takes for an instance to become available and start processing requests.

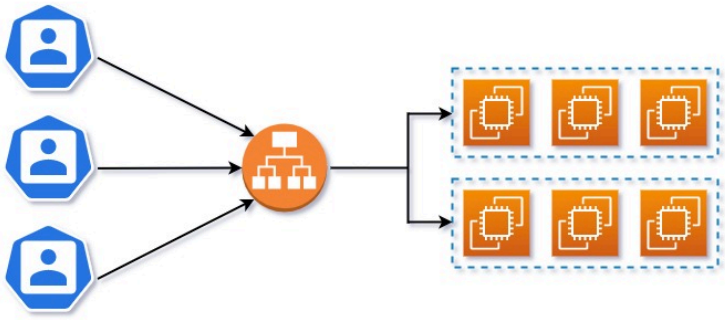
3. Standardizing Security-Compliant AMIs Across Accounts and Regions

EC2 Image Builder enables you to enforce consistent security controls and configurations on your AMIs, which can be distributed across different regions and AWS accounts.

References:
https://docs.aws.amazon.com/imagebuilder/latest/userguide/what-is-image-builder.html
https://aws.amazon.com/blogs/security/how-to-set-up-continuous-golden-ami-vulnerability-assessments-with-amazon-inspector

Amazon Elastic Load Balancing (ELB)

Elastic Load Balancing (ELB) distributes incoming traffic across multiple targets such as Amazon EC2 instances, AWS Lambda functions, Amazon EKS containers, Amazon ECS tasks, and other components. ELB scales as traffic changes allowing it to handle millions of requests per second. ELBs are typically used in conjunction with ASG to provide a single point of access for users to the underlying EC2 instances. Another advantage of ELB is that it allows you to create a highly available environment for your application. When you create an ELB, AWS assigns a load balancer node in the availability zone that you specify. You can route traffic to servers running in different availability zones. So assuming you have servers in three availability zones, in case one of them suffers an outage, you'd still end up with two unaffected zones that can receive traffic.



Load Balancer Types

ELB offers different types of load balancers. The type of load balancer depends on the traffic that you want to distribute. Another thing to check when setting up an ELB is whether the load balancer would be internet-facing or internal.

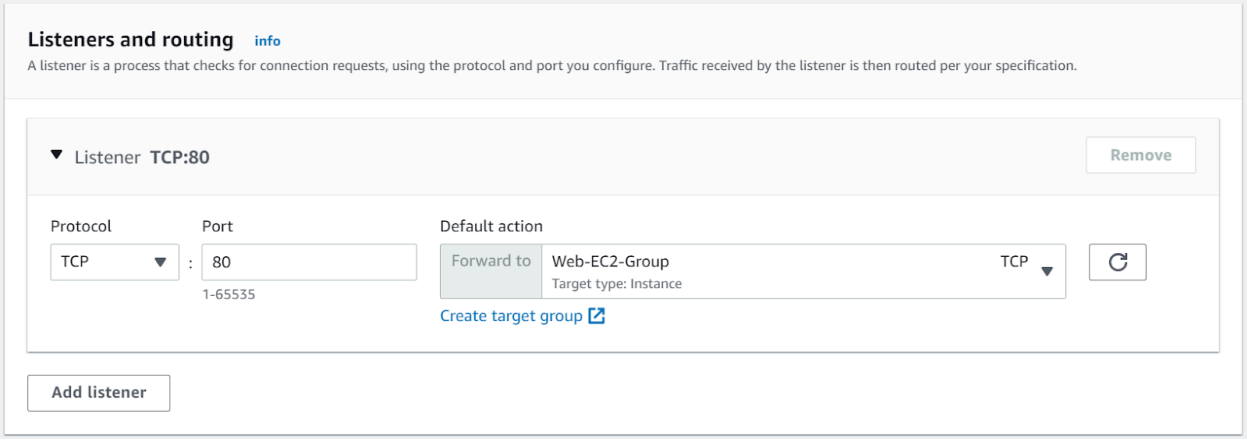
Application Load Balancer (ALB)

For routing HTTP and HTTPS requests, an Application Load Balancer is a good choice. Since this load balancer works on Layer 7, it can distribute your web application's HTTP and HTTPS requests. An ALB can work as an internet-facing and internal load balancer and work for IPv4 and dualstack IP addresses. It requires a minimum of two different subnets on different Availability Zones. Note that a server certificate is required if you have a listener configured for HTTPS protocol. You can either use a certificate from AWS Certificate Manager or IAM.

Network Load Balancer (NLB)

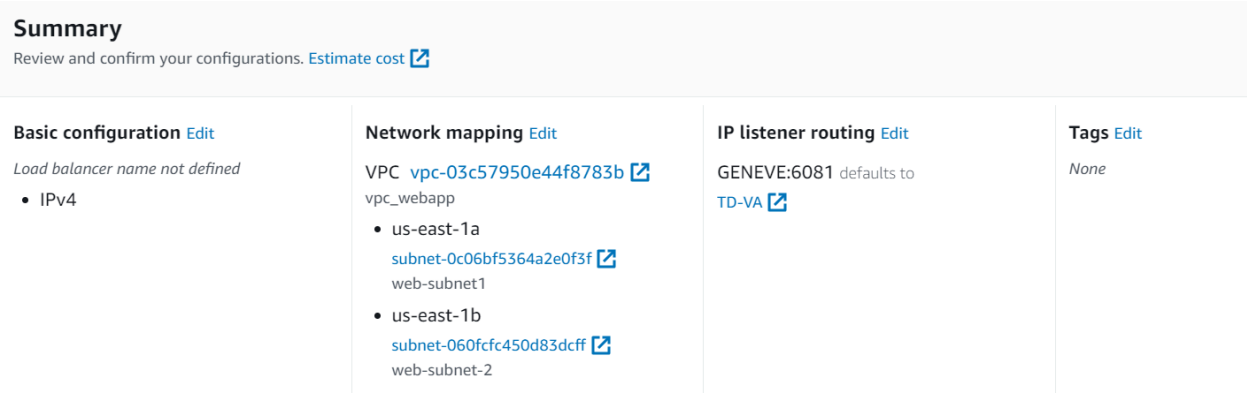
A Network Load Balancer operates at the transport layer (Layer 4) and is capable of handling millions of requests per second at ultra-low latencies. The Network Load Balancer can also be either internal or internet-facing and also supports both IPv4 and dualstack IP addresses. NLBs only see traffic information such

as the source/destination ports, IP addresses, and packet sizes. It does not consider the additional data that an HTTP request has, like its headers, body, cookies, etc. You can set up a Network Load Balancer using only one subnet, but two subnets are ideal for improving your application's fault tolerance. Make sure as well that the target group has at least one valid target to accept the requests. If multiple AZs are specified, you can enable Cross-Zone Load Balancing to distribute the traffic in all targets equally. The direction of the network traffic will depend on the rules defined by the listener and routing.



Gateway Load Balancer (GWLB)

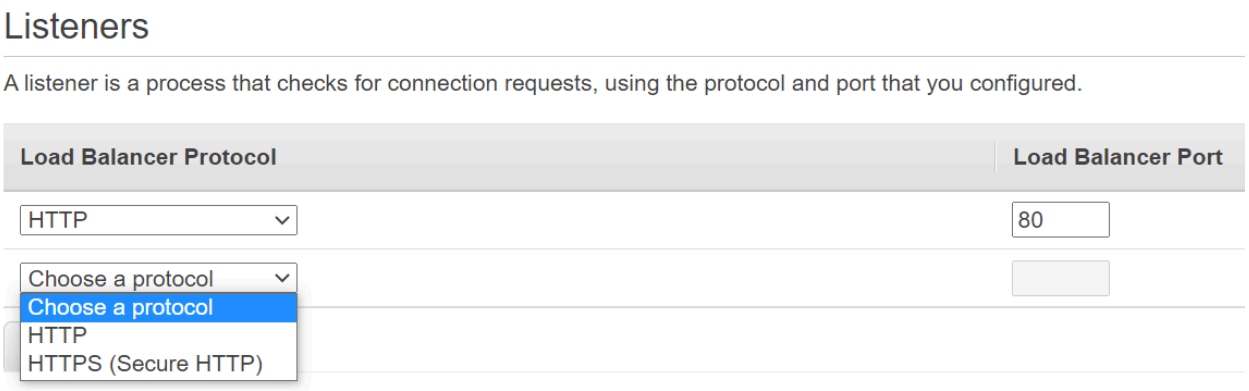
A Gateway Load Balancer eases the management of multiple virtual appliances. A GWLB can distribute traffic across multiple virtual appliances that support Generic Network Virtualization Encapsulation (GENEVE). You can enable GWLB on a single subnet, but the target group should have at least one target that accepts the GENEVE protocol. Take note that Gateway Load Balancer only supports IPv4 addresses; it doesn't support IPv6 and dualstack.



References:
https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/what-is-load-balancing.html
https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/how-elastic-load-balancing-works.html

ELB components

1. Listener - a process that continuously checks for connection requests in your load balancer using the protocol, port, or any other setting that you configured. For example, you can create a listener for HTTP requests that checks for any inbound traffic that is using port 80. So if the user enters http://tutorialsdojo.com into a web browser, that request will be picked up by the listener. However, if another user sends an HTTPS request to your application, then the traffic might not be routed properly since the HTTPS protocol uses port 443.



2. Target - it can be an EC2 instance, an IP address, a Lambda function, or an ECS task. These targets are organized in a resource called Target Group. All the traffic that is captured by the listener is directed to a particular target group that contains the compute resources. For example, you can create a target group that contains several EC2 instances and a secured listener that listens to all HTTPS requests on port 443.

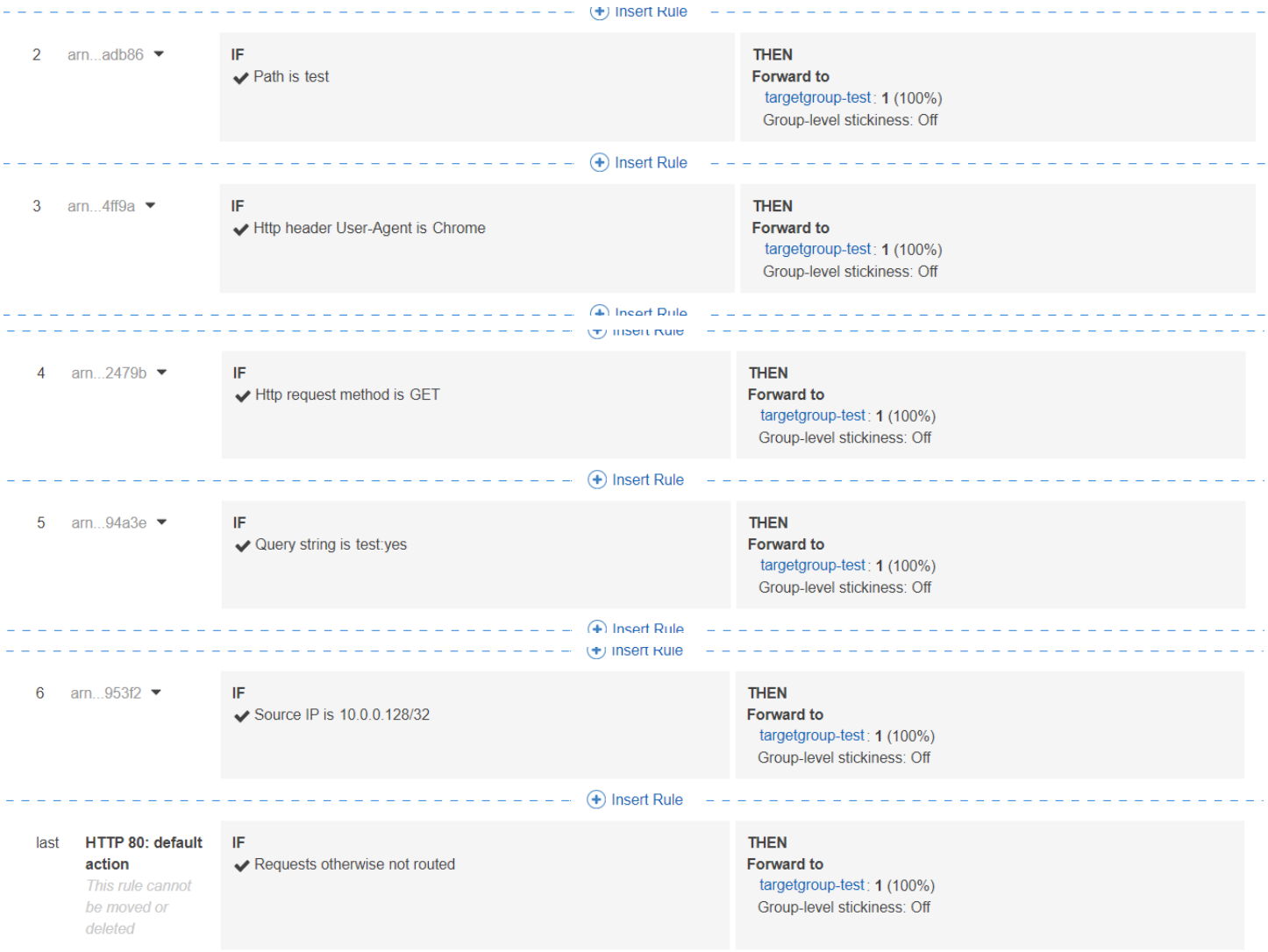
Application Load Balancer Listener Rule Conditions

The AWS ELB Application Load Balancer is rich in routing features that cannot be found in other types of elastic load balancers. A listener rule condition determines how the load balancer routes request to its registered targets.

You can add the following conditions to a listener rule to create multiple routing paths under a single load balancer:

- **host-header** — Route based on the hostname of each request. Also known as host-based routing. This condition enables you to support multiple subdomains and different top-level domains using a single load balancer. Hostnames and match evaluations are not case-sensitive.
- **http-header** — Route based on the HTTP headers for each request. Standard and custom headers are supported. Header name and match evaluation are not case-sensitive.
- **http-request-method** — Route based on the HTTP request method of each request. You can specify standard or custom HTTP methods for the value. The match evaluation is case-sensitive, so to properly route requests to this condition, the request method must exactly match the value you've entered.
- **path-pattern** — Route based on path patterns in the request URLs. Also known as path-based routing. This condition allows you to route to multiple targets depending on the URL path supplied in the request. URL path does not include the query parameters. Path evaluation is case-sensitive.
- **query-string** — Route based on key/value pairs or values in the query strings. Match evaluation is not case-sensitive. This condition does not include the URL path in the evaluation.
- **source-ip** — Route based on the source IP address of each request. The IP address must be specified in CIDR format. Both IPv4 and IPv6 addresses are supported as values for this condition. If a client is behind a proxy, the condition evaluates the IP address of the proxy, not the IP address of the client.

A listener rule can include up to one of each of the following conditions: host-header, http-request-method, path-pattern, and source-ip; and include one or more of each of the following conditions: http-header and query-string. You can also specify up to three match evaluations per condition, but only up to five-match evaluations per rule. This gives you more value to work with for each condition you create.



References:

https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-listeners.html#rule-condition-types
https://tutorialsdojo.com/aws-elastic-load-balancing-elb
https://aws.amazon.com/elasticloadbalancing/
https://aws.amazon.com/blogs/aws/category/elastic-load-balancing/

Multi-Value Headers

In some cases, you may want to build a hybrid set up for your application in which some requests are routed to EC2 instances while others are handled by a Lambda function. You can adopt this model to offload some of the tasks normally performed by your EC2 instances to AWS Lambda. This enables you to deploy smaller and cheaper instances which can save cost in the long run.

Processing requests with Lambda functions as an ALB target can be a bit clunky. By default, if a request or response from a Lambda function includes headers/query parameters with multiple values or multiple values for the same query parameter key, ALB will only pick up the last value.

Suppose a client sends the sample request:

`https://tutorialsdojo.com?id=123&id=789`

Upon receiving the request, ALB will include the `queryStringParameters` to the event data sent to the Lambda function. This parameter contains the last query parameter in the client request,.

```
"queryStringParameters": { "id": "789"}
```

To include all query parameter values, simply enable the multi-value setting in ALB. Once enabled, the headers and query parameters exchanged between the ALB and the Lambda function will be constructed as an array instead of string. This time instead of `queryStringParameters`, ALB uses `multiValueQueryStringParameters` to represent the values associate to a particular query parameter.

```
"multiValueQueryStringParameters": { "id": ["123", "789"] }
```

References:
https://docs.aws.amazon.com/elasticloadbalancing/latest/application/lambda-functions.html#multi-value-headers
https://aws.amazon.com/blogs/networking-and-content-delivery/lambda-functions-as-targets-for-application-load-balancers/

Preserving Client's IP address using the X-Forwarded-For header

If your application is fronted by an Application Load Balancer (ALB), your server logs will only contain the IP address of the ALB as the source of the request. This is because the ALB intercepts the traffic before it is

passed down to your server. If you need to determine the actual IP addresses of users who are hitting your server for logging purposes, you can retrieve them from the `X-Forwarded-For` header.

The snippet below represents the headers typically included in an HTTP request that your application might see from ALB.

```
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/97.0.4692.99 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,
*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Connection: keep-alive
X-Forwarded-For: 203.0.113.195
X-Forwarded-Proto: http
X-Forwarded-Port: 80
X-Amzn-Trace-Id: Root=1-61f1d2ba-6ea61b6a65f6c0975439a9a3
```

In the ALB's configuration, you can set the `routing.http.xff_header_processing.mode` attribute to append to tell ALB to include the `X-Forwarded-For` header in the HTTP request. When a request is made through a series of proxies, the X-Forwarded-For request header may contain multiple IP addresses separated by commas. In this case, the left-most IP address is typically the original client IP address that made the request, and any subsequent addresses represent the IP addresses of intermediate proxies in the request chain

References:
https://docs.aws.amazon.com/elasticloadbalancing/latest/application/x-forwarded-headers.html

Amazon Elasticache

Memcached vs. Redis

Amazon ElastiCache is a caching service that allows you to configure, execute, and scale open-source in-memory databases like Memcached and Redis. Because they store data in RAM (Random-Access Memory), they can read data faster than disk-based databases. Large volumes of reads are often the cause of application performance slowdowns. A caching layer is a common pattern for alleviating this issue. Instead of continually retrieving the most frequently accessed data from your traditional disk-based storage, you may redesign your application to store and get data from an Amazon ElastiCache cache instance. This service, in addition to caching, can be utilized in real-time analytics, distributed session management, geographic services, and a variety of other applications.

ElastiCache is classified into two types:

1. **Amazon ElastiCache for Memcached** is ideal for caching simple data structures likekey-value pairs . Memcached is multithreaded, meaning that it can take advantage of several processing cores enabling you to handle additional processes by increasing computational capacity. The disadvantage of using Memcached is its lack of data replication functionality which can impact your application's availability.

2. **Amazon ElastiCache for Redis** supports more advanced data structures such as lists, sets, hashes, sorted sets. It also offers advanced functionality for sorting and ranking data, which can be useful for applications like leaderboards and rankings. In addition, ElastiCache for Redis provides several other features, such as pub/sub messaging, geospatial indexing, and point-in-time snapshot support. Data replication is available as well, making it a reliable and high-availability storage solution. By enabling Cluster Mode, developers can create a Redis cluster with multiple primary nodes and replicas across two or more Availability Zones, providing redundancy and failover capabilities.

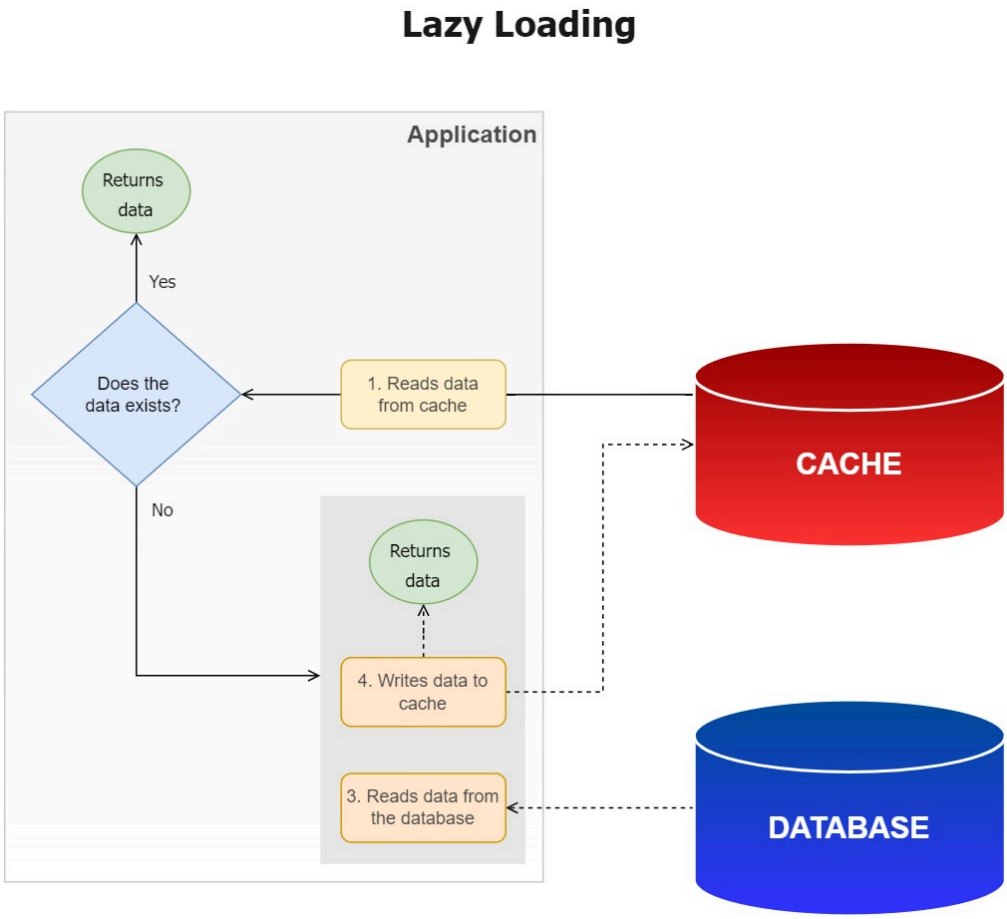
Reference:
https://aws.amazon.com/elasticache/redis-vs-memcached/

Caching Strategies

There are various design patterns for caching data, each with a different purpose that is dependent on the needs of your application. There are two primary caching patterns that you should be aware of in the exam.

Lazy Loading

Lazy loading is perhaps the most widely used caching strategy for web applications. It reduces loading times and improves responsiveness by loading data into the cache only when necessary. As a result, only the requested data is cached. Lazy loading prevents the cache from storing data that are rarely accessed. The downside, however, is the extra latency that occurs during a cache miss since. Also, the requested data might become outdated because the cache only refreshes with new updates when there is a cache miss. The diagram below illustrates how Lazy loading works.



1. The application tries to retrieve data from the cache. If the data is found in the cache (cache hit), the calling function returns it to the application.
2. If the cache does not have the requested data (cache miss), the cache sends a `null` response to the application.
3. The application's caching logic handles the `null` response by retrieving the requested data from the database. The requested data is then written to the cache before being returned to the application by the calling function.

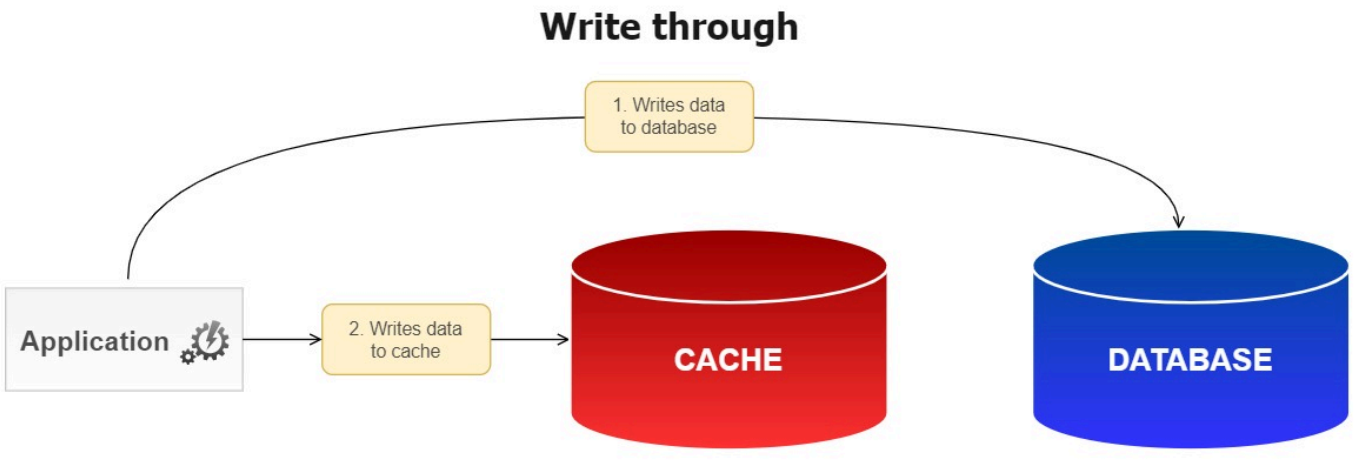
The pseudocode below is an example of how you might implement a lazy loading logic.

```
get_user(user_id)

    record = cache.get(user_id)

    if (record == null)
        query_statement = "SELECT * FROM Users WHERE id = {user_id}"
        record = db.query(query_statement)
        cache.set(user_id, record)

    return record
```

Write through

The write-through strategy adds or updates data in the cache whenever new data is written to the database. As a result, the data that the application retrieves from the cache is always the latest. One of the disadvantages of a write-through strategy is missing data when adding new nodes. This due to the fact that the cache is only updated once a write operation is performed against the database. You can implement a combination of write-through and lazy loading to solve the problem of missing data.

The diagram below illustrates how write-through works.



1. The application writes data to the database.
2. The application writes data to the cache.

The pseudocode below is an example of how you might implement a write-through logic.

```
update_username(user_id, username)
    query_statement = "UPDATE Users SET username = {username} WHERE id = {user_id}"
    record = db.query(query_statement)
    cache.set(user_id, record)
    return ok
```

Reference:
https://docs.aws.amazon.com/AmazonElastiCache/latest/mem-ug/Strategies.html

Amazon Kinesis

Amazon Kinesis is a suite of services that helps you work with streaming data. It has four services under it namely, *Amazon Kinesis Data Streams*, *Amazon Data Firehose*, *Amazon Managed Service for Apache Flink* and *Amazon Kinesis Video Streams*. For the purposes of the exam, we'll only focus on **Amazon Kinesis Data Streams** and **Amazon Data Firehose**.

Kinesis Data Stream

Kinesis Data Stream allows you to ingest data in real-time from various sources, such as IoT devices, web apps, system logs, and then process it using the tools of your choice.

Shards

When creating a Kinesis stream, you set the number of shards that you want to allocate. A shard is a unit of capacity that determines how much data can be written to or read from a stream at any given time. When doing capacity planning, it's important to consider the average record size and the number of consumers that you'll be dealing with.

If your records are large, you may need to provision more shards to handle the volume of data. Likewise, having multiple consumers read data from the same shard can cause contention and limit the throughput capacity of the shard.

Provisioned vs On-demand capacity mode

You have two options for capacity provisioning: **Provisioned** or **On-demand**.

1. Provisioned mode
    - A single shard supports
        - A maximum of 1,000 records/second and 1 MB/second for writes.
        - A maximum of 2 MB/second for reads
    - Can retain data in the stream for 24 hours (default) up to 365 days.
    - Supports server-side encryption using a KMS key

2. On-demand mode
    - Kinesis automatically scales throughput capacity based on traffic
    - By default, a data stream starts with 4 MB/s of write and 8 MB/s of read throughput but can go as high as 200MB/s and 400 MB/s for writes and reads, respectively.

- If you require higher throughput, you may submit a quota increase from AWS Support.
- Suitable for unpredictable and varying throughput requirements.
- Can retain data in the stream for 24 hours (default) up to 365 days.
- Supports server-side encryption using a KMS key

References:
https://docs.aws.amazon.com/streams/latest/dev/key-concepts.html
https://aws.amazon.com/kinesis/data-streams/
https://docs.aws.amazon.com/streams/latest/dev/service-sizes-and-limits.html

Writing and Reading data in shards

Writing to a shard

You can write to a shard using either the **`PutRecord`** or **`PutRecords`** API. **`PutRecord`** allows you to write a single record at a time, while **`PutRecords`** enables you to write records in batches. When using **`PutRecords`**, Kinesis will attempt to process all records in the request, even if there is a failure in processing one of the records, which means the ordering of records is not guaranteed. If you need to ensure that records are processed in the same order they were written in the shard, use **`PutRecord`**.

Note that each piece of data written to a stream is assigned a unique sequence number. These sequence numbers increase over time, so newer data will have a higher sequence number than older data. However, it's important to note that these sequence numbers are specific to a particular shard within the stream. If you're writing data to multiple shards within the same stream, the sequence numbers will increase independently for each shard. To ensure that your data is strictly ordered based, you need to write to each shard serially, one at a time, and add the **`SequenceNumberForOrdering`** parameter in your PutRecord request.

Optionally, you may use the Kinesis Producer Library (KPL) framework for writing multiple records to multiple shards in a single request. Using KPL over the native **`PutRecords`** API can help reduce the complexity of your code and make it easier to write robust, scalable applications. Take note that KPL uses buffering methods to achieve higher throughput, which means some processing delays are to be expected. If your application cannot tolerate delays in writing data to Kinesis streams, KPL may not be the best choice.

Reading from a shard

Reading from a shard typically involves two steps:

1. Obtaining an iterator for the shard from which you want to read. This can be done by calling the **`GetShardIterator`** API and specifying the shard ID and the type of iterator you want to use. An **iterator** serves like a bookmark that pinpoints the position of the last record that was read.

2. Use the **`GetRecords`** API and provide the iterator along with the maximum number of records to read. The **`GetRecords`** response includes a new iterator that can be used to read the next set of records. In some cases, you may need to obtain a new iterator separately, such as when the current iterator has expired, or you need to read from a different position in the shard.

Iterators expire after a certain period of time. When you're using a Lambda function to process data from a stream, it's important that you monitor the **`IteratorAge`** metric. **`IteratorAge`** measures how much time has passed since the Lambda function last read a piece of data from a stream. If the Iterator age gets too high, it means the Lambda function is falling behind and not keeping up with new data being added to the stream. This can cause problems like slow processing of new data, duplicate processing of old data, or even missing data altogether.

The following methods can help mitigate an increasing **`IteratorAge`:**

- Increase the amount of memory allocated to your Lambda function so record processing time becomes faster. You may also try optimizing your function's code to improve execution time without additional cost.
- Increase shard count.
- Increase your Lambda function's batch size.
- Using a parallelization factor to increase the number of concurrent Lambda invocations for each shard of a stream.
- Enabling the enhanced fan-out feature on the stream.

Enhance fan-out

Imagine that there are multiple Lambda functions that read data from a stream at the same time. Which means they might have to compete for access to the stream. Normally, these functions would have to compete for access to the stream, which could slow down data retrieval since they would be sharing the stream's capacity. However, with enhanced fan-out, each function gets its own dedicated portion of the stream's capacity. So if the stream can handle 10 MB of data per second, and there are three consumers using enhanced fan-out, each function can receive up to 2 MB of data per second without having to compete with the others. This can reduce latency and increase read-throughput.

Just like KPL is for publishing data to a stream (Kinesis Consumer Library), KCL is a framework for consuming data. KCL provides a simple way to read and process data from a Kinesis stream using an application-level

consumer. It takes care of many of the low-level details involved in consuming data, such as managing the stream's shards, coordinating with other consumers, and handling failures.

References:
https://docs.aws.amazon.com/kinesis/latest/APIReference/API_PutRecord.html
https://docs.aws.amazon.com/kinesis/latest/APIReference/API_PutRecord.html
https://docs.aws.amazon.com/streams/latest/dev/troubleshooting-consumers.html
https://aws.amazon.com/premiumsupport/knowledge-center/lambda-iterator-age/

Scaling, Resharding and Parallel Processing

- **Kinesis Resharding** enables you to increase (**split shard**) or decrease the number of shards (**merge shards**) in a stream to adapt to changes in the rate of data flowing through the stream.
- Resharding is always pairwise. You cannot split into more than two shards in a single operation, and you cannot merge more than two shards in a single operation.
- The Kinesis Client Library (KCL) tracks the shards in the stream using an Amazon DynamoDB table, and adapts to changes in the number of shards that result from resharding. When new shards are created as a result of resharding, the KCL discovers the new shards and populates new rows in the table.
- The workers automatically discover the new shards and create processors to handle the data from them. The KCL also distributes the shards in the stream across all the available workers and record processors.
- When you use the KCL, you should ensure that **the number of instances does not exceed the number of shards** (except for failure standby purposes).
    - **Each shard is processed by exactly one KCL worker and has exactly one corresponding record processor**.
    - **One worker can process any number of shards**.
- You can scale your application to use more than one EC2 instance when processing a stream. By doing so, you allow the record processors in each instance to work in parallel. When the KCL worker starts up on the scaled instance, it load-balances with the existing instances, so now each instance handles the same amount of shards.
- To scale up processing in your application:
    - Increase the instance size (because all record processors run in parallel within a process)
    - Increase the number of instances up to the maximum number of open shards (because shards can be processed independently)
    - Increase the number of shards (which increases the level of parallelism)

Reference:
https://docs.aws.amazon.com/streams/latest/dev/kinesis-record-processor-scaling.html

Amazon Data Firehose

Unlike Kinesis Data Stream that can store data, Kinesis Firehose is a service that can deliver streaming data in near-real time to a destination that you specify such as Amazon S3, Amazon Redshift, Amazon OpenSearch Service, or a custom service via HTTP. Kinesis Firehose buffers data before it is delivered to a destination. By default, Firehose buffers incoming data for up to 300 seconds (5 minutes) or 128 MB, whichever comes first.

You don't have to provision capacity to a Kinesis Firehose stream, as it automatically adjusts the resources it uses based on the incoming data volume. You may also transform data prior to delivery using a Lambda function. This allows you to easily modify and enrich your data as needed without having to manage separate processing infrastructure.

References:
https://docs.aws.amazon.com/firehose/latest/dev/what-is-this-service.html

Amazon Copilot

AWS Copilot is a command-line interface (CLI) designed to help you, as developers, easily launch and manage containerized applications on AWS services like Amazon Elastic Container Service (ECS), AWS Fargate, and AWS App Runner. Copilot simplifies many of the tasks involved in running your applications in containers, including setting up infrastructure, deploying applications, and configuring delivery pipelines directly from your code repository to your application's environment. It offers scalable, production-ready, and secure infrastructure-as-code (IaC) templates to speed up your development process, allowing you to focus more on building your applications rather than managing infrastructure.

Using the AWS Copilot command line interface

The AWS Copilot CLI is compatible with developer workflows that facilitate the best practices for today's applications, such as building a CI/CD pipeline that is provided on the user's behalf and employing infrastructure as code. Instead of using the AWS Management Console for your regular development and testing cycle, use the AWS Copilot CLI.

There are two ways to install the AWS Copilot;

1. Installing the AWS Copilot CLI using Homebrew

You need to have the Homebrew installed first, before proceeding to install the AWS Copilot. This command can be used to your MacOS or linux system.

```
brew install aws/tap/copilot-cli
```

2. Manually installing the AWS Copilot CLI

- For MacOS

```
$ sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releas
es/latest/download/copilot-darwin \
>    && sudo chmod +x /usr/local/bin/copilot \
>    && copilot --help
```

- For Linux

```
$ sudo curl -Lo /usr/local/bin/copilot https://github.com/aws/copilot-cli/releas
es/latest/download/copilot-linux \
>    && sudo chmod +x /usr/local/bin/copilot \
>    && copilot --help
```

- For Windows

```
$ New-Item -Path 'C:\copilot' -ItemType directory; `
>   Invoke-WebRequest -OutFile 'C:\copilot\copilot.exe' https://github.com/aws/copilot
-cli/releases/latest/download/copilot-windows.exe
```

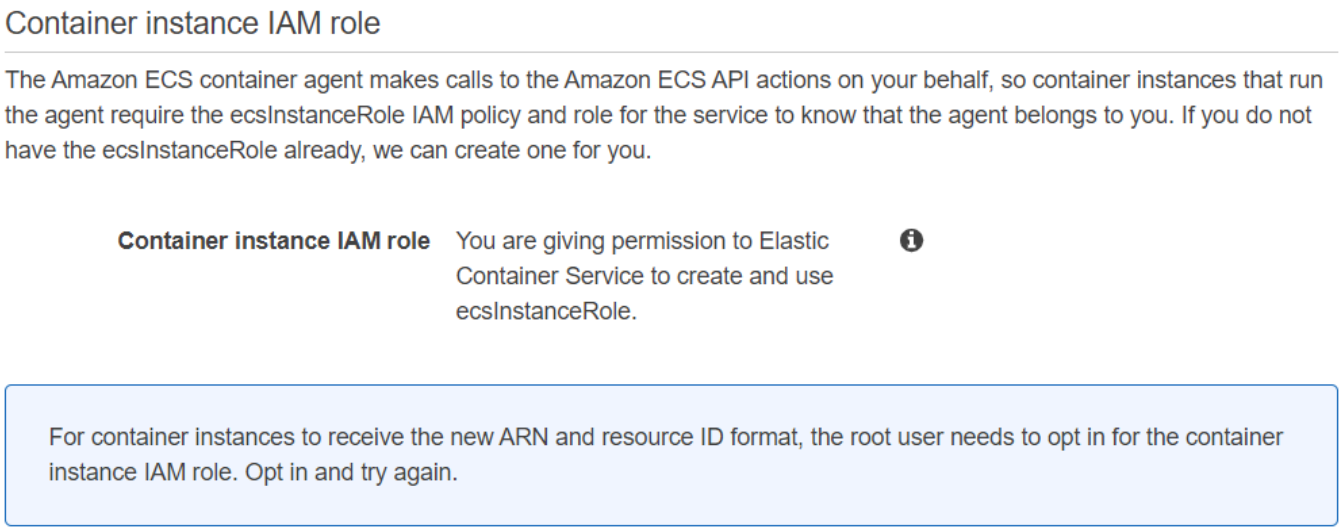
References:
https://aws.amazon.com/containers/copilot/
https://docs.aws.amazon.com/AmazonECS/latest/developerguide/AWS_Copilot.html
https://docs.aws.amazon.com/AmazonECS/latest/developerguide/copilot-install.html

Amazon Elastic Container Service (ECS)

Amazon ECS Container Instance Role vs Task Execution Role vs Task Role

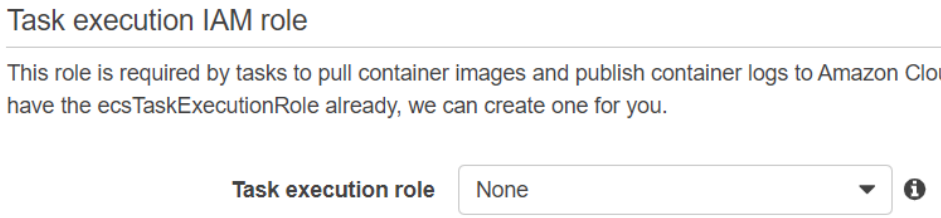
An ECS cluster is the very first resource you create in Amazon ECS. You define your cluster's underlying infrastructure, instance provisioning model (on-demand or spot), instance configuration (AMI, type, size,

volumes, key pair, number of instances to launch), cluster network, and container instance role. The container instance role allows the Amazon ECS container agent running in your container instances to call ECS API actions on your behalf. This role attaches the `ecsInstanceRole` IAM policy.

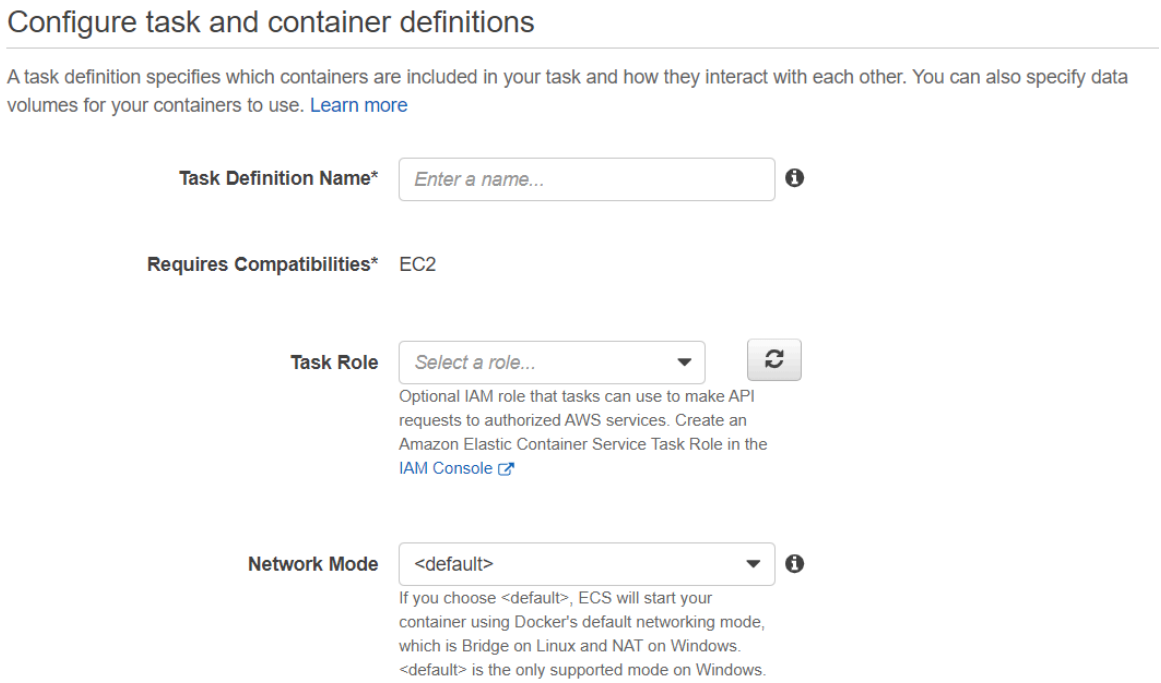


After creating your ECS cluster, one of the very first things you'll do next is create your task definition. A task definition is like a spec sheet for the Docker containers that will be running in your ECS instances or *tasks*. The following are the parameters that are defined in a task definition:

- The Docker image to use with each container in your task
- CPU and memory allocation for each task or each container within a task
- The launch type to use (EC2 or Fargate)
- The Docker networking mode to use for the containers in your task
- The logging configuration to use (`bridge`, `host`, `awsvpc`, or `none`)
- Whether the task should continue to run if the container finishes or fails
- The command the container executes when it is started
- Volumes that should be mounted on the containers in a task
- The Task Execution IAM role provides your tasks permissions to pull Docker images and publish container logs.



Lastly, since the containers running in your ECS tasks might need to make some AWS API calls themselves, they will need the appropriate permissions to do so. The task role provides your container's permissions to make API requests to authorized AWS services. In addition to the standard ECS permissions required to run tasks and services, IAM users also require iam:PassRole permissions to use IAM roles for tasks. Assigning a task role is optional.



Container Images

Container images are lightweight, standalone executable software packages that include everything you need to run a piece of software, such as the code, runtime, system libraries, and settings. They are crucial for ensuring consistent application deployment across different environments. By packaging all dependencies within the container image, you can avoid the common "it works on my machine" problem, ensuring that your applications run seamlessly, whether deployed in development, testing, or production environments. This

portability makes container images a cornerstone of modern DevOps practices and cloud-native application development.

In Amazon ECS, you use container images to define task definitions, which act as blueprints for running containers. These images can be sourced from public repositories like Docker Hub or private repositories such as Amazon Elastic Container Registry (ECR). When setting up task definitions, you specify various parameters, including the Docker image to be used, resource allocation (CPU and memory), networking modes, and logging configurations. By leveraging ECS and container images, you can achieve efficient, scalable, and reliable application deployments, maintaining consistent operational environments and enabling the flexible execution of microservices architectures

References:
https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_execution_IAM_role.html
https://docs.aws.amazon.com/AmazonECS/latest/developerguide/container-considerations.html
https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-iam-roles.html
https://tutorialsdojo.com/amazon-elastic-container-service-amazon-ecs/
https://docs.aws.amazon.com/AmazonECS/latest/developerguide/AWS_Copilot.html#copilot-install

Amazon OpenSearch Service (successor to Amazon Elasticsearch Service)

Amazon OpenSearch Service is a managed service built on top of OpenSearch, a distributed, open-source analytics platform suited for applications such as log analytics, real-time application monitoring, and search engines. OpenSearch enables developers to easily run a complex full-text search and filter queries over documents more efficiently than traditional SQL and NoSQL databases.

An easy way to stream data to an OpenSearch cluster is through Amazon Kinesis Firehose. If you're sending unstructured data, such as Apache logs, you can use Kinesis Firehose's data transformation feature, which leverages AWS Lambda to convert the logs to a format required by your OpenSearch cluster.

References:
https://docs.aws.amazon.com/opensearch-service/
https://aws.amazon.com/opensearch-service/the-elk-stack/data-ingestion/

Amazon Elastic Kubernetes Services (Amazon EKS)

Amazon EKS lets you easily run and scale Kubernetes applications in the AWS cloud or on-premises. Kubernetes is not an AWS native service. Kubernetes is an open-source container-orchestration tool used for

deployment and management of containerized applications. Amazon EKS just builds additional features on top of this platform, so you can run Kubernetes in AWS much more easily.
If you have containerized applications running on-premises that you would like to move into AWS, but you wish to keep your applications as cloud agnostic as possible, then EKS is a great choice for your workload. All the Kubernetes-supported tools and plugins you use on-premises will also work in EKS. You do not need to make any code changes when re-platforming your applications.

An EKS cluster consists of two components:

- The Amazon EKS control plane
- And the Amazon EKS nodes that are registered with the control plane

The Amazon EKS control plane consists of control plane nodes that run the Kubernetes software, such as *etcd* and the Kubernetes API server. The control plane runs in an account managed by AWS, and the Kubernetes API is exposed via the cluster's EKS endpoint. Amazon EKS nodes run in your AWS account and connect to your cluster's control plane via the API server endpoint and a certificate file that is created for your cluster.

References:
https://docs.aws.amazon.com/eks/latest/userguide/clusters.html
https://aws.amazon.com/premiumsupport/knowledge-center/eks-worker-nodes-cluster/

Amazon Athena

Amazon Athena is a serverless, interactive query service that makes it easy for developers to analyze data in Amazon S3 using SQL without the need for provisioning or managing any infrastructure.

To get started with Athena, your first need to create a data catalog in AWS Glue, which stores the metadata for the data sets stored in S3. This data catalog can be created and managed through the AWS Management Console or by using the Glue API.

Optimizing queries

Athena supports various data formats, including CSV, JSON, Parquet, and ORC. The performance of your queries can be improved by converting your data into columnar formats such as Parquet and ORC. You may also implement data partitioning to limit the amount of data that needs to be scanned by a query. This can lead to Athena returning query results faster at a reduced cost.

Here are a few examples of scenarios where Amazon Athena could be used by developers:

1. Log analysis: A developer needs to analyze log data stored in S3 to understand user behavior and identify any performance issues. They can use Athena to run SQL queries against the log data to extract useful information, such as user activity, error rates, and response times.
2. Business intelligence: A developer working on a business intelligence project needs to analyze large data sets stored in S3 to identify trends and insights that can inform business decisions. They can use Athena to run queries on the data, such as calculating metrics like sales by region or identifying patterns in customer behavior.
3. Data migration: A developer is working on migrating data from an older system to a new one and needs to validate the data before loading it into the new system. They can use Athena to run SQL queries to check for data integrity and consistency without having to load the data into a separate database.

References:
https://docs.aws.amazon.com/athena/latest/ug/what-is.html
https://aws.amazon.com/blogs/big-data/analyzing-data-in-s3-using-amazon-athena/

AWS Cloud Development Kit (CDK)

AWS Cloud Development Kit (CDK) is an open-source software development framework for creating and managing cloud infrastructure using programming languages. It allows you to define cloud infrastructure in a familiar programming language, such as TypeScript, JavaScript, C#, .NET, Python or Go, rather than writing CloudFormation templates in JSON or YAML.

AWS CDK's main selling point is it lets you apply the programming skills and constructs you already know, like loops, conditionals, and functions, to build infrastructure as code. By doing so, you can significantly reduce the amount of code required and create more concise and readable IaC templates. In contrast, the CloudFormation template tends to be more verbose and requires explicit definition of resource properties and relationship mappings.

Constructs

A construct is your building block for defining cloud infrastructure. It is a class that represents a specific resource or group of AWS resources. For example, a construct can be as simple as an S3 bucket or something more complicated like a group of resources: an Amazon Elastic Container Service (ECS) cluster running a containerized application with a load balancer and an auto-scaling group.

AWS CDK constructs comes in three levels:

1. L1 constructs (a.k.a *CFN resources*) - these are the low-level constructs that map directly to components that are available in CloudFormation. For example, a security group, S3 bucket, or a DynamoDB table.

2. L2 constructs - these are higher-level constructs that are built using L1 constructs. These constructs follows best practices and design patterns and comes with resources with proper default settings, enabling you to write resources faster without worrying too much about the details. For example, you can create a VPC with public and private subnets in all Availability Zones in just two lines of code:

```
import ec2 = require('@aws-cdk/aws-ec2');

const vpc = new ec2.VpcNetwork(this, 'VPC');
```

3. L3 constructs (a.k.a *Patterns*)- these are the highest-level constructs, they are built on top of L2 constructs and provide a higher level of abstraction and encapsulation. They represent a complete solution or a group of resources that work together to achieve a specific goal. Architectures built on top

of L3 constructs follow best practices and conventions, such as the least privilege of access, which can help you improve the structure and maintainability of your CDK apps

References:
https://aws.amazon.com/blogs/aws/boost-your-infrastructure-with-cdk/
https://docs.aws.amazon.com/cdk/v2/guide.html

Amazon Route 53

Amazon Route 53 is a highly available and scalable Domain Name System (DNS) web service. DNS translates human-friendly domain names (like "tutorialsdojo.com") into IP addresses (like "1.2.3.4"). When you enter a domain name into your web browser, DNS helps your computer find the correct IP address so it can connect to the right server and load the website you want to visit.

Route 53 works similarly to other DNS providers out there such as CloudFlare and GoDaddy, with a few extra functionalities. You aren't required to use Route 53 as your DNS provider if you are using the AWS cloud, but since Route 53 is tightly integrated with other AWS services, you can always move from your current provider to enjoy these benefits. Route 53's primary functions can be summarized into four sections:

1. Domain registration
2. DNS management
3. Traffic management
4. Availability monitoring

DNS Management

You may use Route 53 as your DNS service even if your domains are registered with a different domain registrar. It is able to resolve DNS queries to targets that are running inside and outside of AWS. In DNS management, everything starts at your hosted zones. A hosted zone is a container for DNS records, and these records contain information about how you want to route traffic for a specific domain. Hosted zones should have the same name as its associated domain. There are two types of hosted zones that you can create — **public hosted zone** and **private hosted zone**. The main difference between the two is, with public hosted zones, the records stored in them are publicly resolvable. On the other hand, private hosted zones contain records that are only resolvable within a VPC you associate, like if you want a record to resolve to a private EC2 instance for example.

In each public hosted zone, Route 53 automatically creates a name server (NS) record and a start of authority (SOA) record. Afterwards, you can create additional records in this hosted zone to point your domain and subdomains to their endpoints. If you are moving from an existing DNS service, you can also import a zone file

instead to automatically populate your hosted zone. Be sure to modify the NS records of the DNS service to use the name servers of AWS. Once you've performed the actions above, just wait for DNS queries to come in (and wait for the DNS cache TTL to expire if the records were existing beforehand), and they should resolve to your designated targets.

Alias vs Non-alias records

There are multiple types of records that you can create in Route 53, but the most common ones you'll encounter are A record, AAAA record, and CNAME record. Furthermore, each of these records can be alias or non-alias records.

Alias record

- a special type of record that maps a domain name to an Amazon resource.
- Supported resources include:
    - An A record in your hosted zone
    - API Gateway endpoint
    - CloudFront distribution
    - Elastic Load Balancer
    - Elastic Beanstalk environment
    - Global Accelerator
    - S3 website endpoint
    - VPC endpoint
- For example, you can make your S3 website available at your domain name by creating an Alias record pointing to it. Note that the domain name you use in the record (e.g., "**shop.customtoys.com**") should exactly match the name of your S3 bucket (e.g., "**shop.customtoys.com.s3-website-us-west-2.amazonaws.com**"), otherwise your website may not be accessible via your domain name.

Non-alias records

- These are your standard record that maps domains to IP addresses (A record for IPv4 addresses and AAAA record for IPv6) or another domain (CNAME).
- You cannot create a CNAME record for the root domain or zone apex. For example, if you have an apex zone for "coolsite.com", you can create CNAME records for subdomains such as "www.coolsite.com" or "blog.coolsite.com", but you cannot create a CNAME record for "example.com" itself.

Traffic Management

Each Route 53 DNS record also has its own routing policy. A routing policy determines how Route 53 responds to DNS queries. Different routing policies achieve different results:

- **Simple routing policy** – Resolves your DNS to a resource as is.
- **Failover routing policy** – Use for configuring active-passive routing failover. You can specify two DNS records with the same DNS name and have them point to two different targets. If your primary target becomes unavailable, Route 53 automatically routes succeeding incoming requests to your secondary target.
- **Geolocation routing policy** – Use when you want to route traffic based on the location of your users. This policy helps you serve geolocation-specific content to your users.
- **Geoproximity routing policy** – Use when you want to route traffic based on the location of your resources and, optionally, shift traffic from resources in one location to resources in another.
- **Latency routing policy** – Use when you have resources in multiple AWS Regions, and you want to route traffic to the region that provides the best latency.
- **Weighted routing policy** – Use to route traffic to multiple resources in proportion to the weights you assign for each target. The greater the weight, the greater the traffic portion it receives. This policy can be used when you've deployed a new version of an application, and you only want to route a percentage of your user traffic to it.
- **IP-based routing policy** – use when you want to route traffic based on your users' locations, and know where the IP address or traffic is coming from.
- **Multivalue answer routing policy** – Use when you want Route 53 to respond to DNS queries with up to eight healthy records selected at random. Users who query this type of record can choose a target from the DNS response to connect to.

Some of these routing policies can actually be used together, such as latency and weighted records, to produce a more complex routing system.

References:
https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/resource-record-sets-choosing-alias-non-alias.html
https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/dns-configuring.html
https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/dns-failover.html

Amazon Virtual Private Cloud (VPC)

Amazon VPC allows you to create a logically isolated virtual network of the AWS Cloud, where you can deploy AWS resources within a self-defined virtual network. This network is designed to reflect a conventional network that one might manage in their own data center but with the scalability and flexibility provided by AWS.

A default virtual private cloud (VPC) that comes with your AWS account is configured with subnets and an internet gateway, so you can use it right away. You possess the adaptability to generate further, customized VPCs that are suited to particular architectural requirements. To meet your specific needs, these configurable virtual private clouds (VPCs) can be set up with specific IP ranges, subnets, route tables, gateways, and security configurations.

Public subnets in a VPC offer direct internet access to instances via an internet gateway. In contrast, private subnets use NAT devices to enable outbound internet traffic while preventing unwanted inbound traffic.

Features such as VPC Flow Logs give you insights into traffic flow, helping you monitor and troubleshoot network issues. Additionally, comprehensive security configurations, including security groups and network ACLs, provide stringent protections for your network resources.

Key Components of Amazon VPC:

- IP Addressing - Both IPv4 and IPv6 are supported within a VPC. You can import your own public and private IP addresses and allocate them to various services such as EC2 instances and NAT gateways.
- Subnets - represent specific IP address ranges within a VPC and are confined to single Availability Zones. They host AWS resources like EC2 instances.
- Routing - Route tables within VPCs control the routing of network traffic from subnets or gateways.
- Gateways and Endpoints - An internet gateway allows connectivity between the VPC and the internet. VPC endpoints enable private connections to AWS services without an internet gateway.
- Security - Security groups and network ACLs control the inbound and outbound traffic at the instance and subnet level, respectively.

References:
https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html
https://docs.aws.amazon.com/vpc/latest/userguide/how-it-works.html

AWS Web Application Firewall (WAF)

AWS Web Application Firewall (AWS WAF) helps safeguard your web applications from common web exploits that could cause application unavailability, jeopardize security, or consume excessive resources.

With AWS WAF, you can save yourself the pain and effort of having to write WAF rules from scratch. AWS WAF offers a broad range of managed WAF rules that have been thoughtfully designed to identify and block a variety of common attack patterns. This includes but is not limited to SQL injection and cross-site scripting attacks.

AWS WAF can be integrated with the following AWS service:
- Amazon CloudFront
- Application Load Balancer (ALB)
- Amazon API Gateway

AWS WAF Rule Statements To Filter Web Traffic

AWS WAF uses rule statements to filter web requests and determine the appropriate action (allow, block, or count) to apply to a web request that matches a specific rule. Rule statements may be simple with only one criterion for a match, or they can be complex and include multiple statements combined using AND, OR, and NOT operators.

In AWS WAF, you can choose from a variety of match statements to create both simple and complex rule statements:

Match Statement	Use Case
Geographic match	Allows you to allow or block web requests based on country of origin by creating one or more geographical, or geo, match statements. If you use the CloudFront geo restriction feature to block a country, requests from that country are blocked and are not forwarded to WAF.
IP set match	Inspects the IP address of a request against a set of IP addresses and address ranges that you want to allow through or block with your WAF.
Label match rule statement	Inspects the request for labels that have been added by other rules in the same web ACL.
Regex pattern set	Lets you compare regex patterns against a specified component of a web request.

Size constraint	Compares the size of a request component against a size constraint in bytes.
SQLi attack	Inspects for malicious SQL code in a web request.
String match	Searches for a matching string in a web request component. If a matching string is found, WAF allows/blocks the request.
XSS scripting attack	Inspects for cross-site scripting attacks in a web request.
Rate-based	Tracks the rate of requests of each originating IP addresses, and triggers a rule action on IPs with rates that go over a limit. You can use this type of rule to put a temporary block on requests from an IP address that's sending excessive requests.

AWS WAF's IP Match condition match statement is highly valuable, as it enables you to effectively block malicious requests coming from known abusive IP addresses. Another essential feature of AWS WAF is its rate-limiting rule. The rate-limiting rule allows you to limit the number of requests that can be sent to your application or service from a particular IP address or group of IP addresses.

Say you're running an application on a group of EC2 instances behind an ALB, you could create a rate-based rule that limits the number of requests within a specified time period. For example, you might set the limit at 100 requests per 5 minutes. Next, you would create a web ACL that includes the rate-based rule and associate it with the ALB. AWS WAF intercepts incoming requests and evaluates them based on the configured rule before routing them to ALB. If the request count of an IP address exceeds the limit, AWS WAF can block subsequent requests from that IP address.

Note that AWS WAF is capable of blocking up to 10,000 IP addresses based on a rate-based rule. However, in some situations where there are more than 10,000 IP addresses sending high rates of requests simultaneously, you may need to narrow the scope of the requests that AWS WAF tracks and counts. To achieve this, you can nest a scope-down statement inside the rate-based statement. This enables you to define a specific set of conditions that must be met before the rate-based rule is applied. For example, you could define a scope-down statement to only count requests that come from specific IP address ranges or that contain certain query string parameters. By using a scope-down statement, AWS WAF will only count requests that match the defined criteria, which can help to reduce the number of IP addresses that are blocked based on the rate-based rule.

References:
https://docs.aws.amazon.com/waf/latest/developerguide/waf-rules.html
https://tutorialsdojo.com/aws-waf

Amazon MemoryDB for Redis

A three-tier application architecture typically includes a presentation layer, an application layer, and a database layer, with a caching layer often added in front of the database to improve application performance. The caching layer stores data in memory, making data retrieval a lot faster as opposed to slower disk-based storage, such as in the case of a database like MySQL.

By using Amazon MemoryDB for Redis as your primary database, you can simplify this architecture by eliminating the need for a separate caching layer. With MemoryDB, you store data directly in memory, providing microsecond read latency, single-digit millisecond write latency and high throughput for your application.

'But memory is volatile by nature so how does Amazon MemoryDB ensure data is not lost in case of failures?', you might ask. Amazon MemoryDB for Redis provides data durability, consistency, and recoverability by maintaining multiple copies of your data across multiple availability zones. MemoryDB for Redis also provides features like automatic failover, backups, and replication. Additionally, MemoryDB for Redis is fully compatible with the Redis protocol, so it can be used with existing Redis clients and applications with minimal changes.

References:
https://aws.amazon.com/memorydb/faqs
https://aws.amazon.com/blogs/aws/introducing-amazon-memorydb-for-redis-a-redis-compatible-durable-in-memory-database-service/

AWS AppConfig

AWS AppConfig is a managed service that helps developers manage and deploy application configurations. It provides a centralized location for managing application settings, which can be accessed by multiple applications and services.

AppConfig can be integrated with the following AWS services:

- AWS Lambda extension
- AWS Secrets Manager
- Amazon Elastic Container Service (Amazon ECS)
- Elastic Kubernetes Service (Amazon EKS)
- AWS CodePipeline

To use AWS AppConfig, developers first need to define the configurations that they want to manage. This could include application settings, feature flags, or any other configuration data. Developers can use the AppConfig console or the AWS CLI to create a configuration profile, which defines the configuration data and the applications or services that can access it.

Once the configuration profile has been created, developers can create a deployment strategy, which specifies how and when to roll out changes to the configuration data. Deployment strategies can be based on a range of factors, such as the percentage of users that should receive the changes, the time of day when changes should be rolled out, or the location of the users.

References:
https://aws.amazon.com/blogs/mt/how-cyberark-implements-feature-flags-with-aws-appconfig/
https://docs.aws.amazon.com/appconfig/latest/userguide/what-is-appconfig.html

AWS CLI (Command Line Interface)

The AWS CLI (Command Line Interface) is a powerful tool designed to streamline your interactions with AWS services. By using the CLI, you can manage multiple AWS services and automate them through scripts, enhancing both efficiency and productivity.

The AWS CLI provides a unified interface that allows you to control virtually all aspects of AWS. Its design is aimed at simplifying the way you manage AWS services from your terminal or command prompt. The AWS CLI v2, the latest version, includes significant enhancements over its predecessor, like improved installation processes that eliminate the need for Python on your machine, making it even more accessible and easier to set up

Installation and Configuration

Installing the AWS CLI is straightforward. For Windows, macOS, and Linux, pre-built binaries are provided, ensuring compatibility across different operating systems. Once installed, setting up your AWS environment is facilitated by commands that help import credentials, configure settings, and even guide you through initial setup using interactive prompts.

Enhanced Interactivity

AWS CLI v2 emphasizes interactivity, aiming to assist both new and experienced users. Features such as server-side auto-completion and interactive prompts enhance usability. These features help you navigate the CLI more effectively by suggesting command completions and guiding you through the command parameters, making the CLI accessible even to those who are not deeply familiar with AWS' extensive command set.

References:
https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html
https://aws.amazon.com/blogs/developer/aws-cli-v2-is-now-generally-available/

AWS AppSync

AWS AppSync is a fully managed service that simplifies the process of building and deploying GraphQL APIs. With AWS AppSync, you can connect and combine data from a variety of data sources such as DynamoDB tables, Lambda functions, Amazon OpenSearch Service, Aurora databases, and even non-AWS sources through HTTP APIS. You can write business logic using pre-built code templates for common GraphQL API patterns.

Why GraphQL Over Rest?

Unlike traditional REST APIs, with GraphQL, clients can specify the exact data they need and how it should be formatted, reducing over-fetching or under-fetching of data. GraphQL also allows you to define a single schema that describes the entire API, making it easier to manage and evolve the API over time.

Key features of AppSync:

- Powerful GraphQL schema editing through the AWS AppSync console, including automatic GraphQL schema generation from
- DynamoDB Support for JavaScript and TypeScript to write business logic that accesses data sources
- Efficient data caching to optimize API performance
- Integration with Amazon Cognito user pools for fine-grained access control at a per-field level

References:
https://docs.aws.amazon.com/appsync/latest/devguide/what-is-appsync.html
https://aws.amazon.com/blogs/architecture/what-to-consider-when-modernizing-apis-with-graphql-on-aws/

AWS Systems Manager

AWS Systems Manager, or SSM is a suite of services that provides you visibility and control of your cloud and on-premises infrastructure. SSM has several features that you can leverage namely:

- AWS Systems Manager Explorer
- AWS Systems Manager OpsCenter
- AWS Systems Manager Incident Manager
- AWS Systems Manager Application Manager
- AWS Systems Manager AppConfig
- AWS Systems Manager Parameter Store
- AWS Systems Manager Change Manager
- AWS Systems Manager Automation
- AWS Systems Manager Maintenance Windows
- AWS Systems Manager Fleet Manager
- AWS Systems Manager Compliance
- AWS Systems Manager Inventory
- AWS Systems Manager Session Manager
- AWS Systems Manager Run Command
- AWS Systems Manager State Manager
- AWS Systems Manager Patch Manager
- AWS Systems Manager Distributor
- ...and many other sub-modules.

AWS Systems Manager also has an SSM agent, just like a CloudWatch or DataSync agent. Through this SSM agent, the Systems Manager service can manage both Amazon EC2 instances and on-premises servers.

The AWS Systems Manager Explorer is basically a customizable operations dashboard which allows you to explore and view the various technical reports regarding your AWS resources. The Explorer module shows an aggregated view of operations data for your AWS accounts and across AWS Regions. This data is referred to as "OpsData" in the AWS Systems Manager Explorer. The OpsData contains the relevant metadata about your Edge devices, on-premises servers, virtual machines and Amazon EC2 instances in your hybrid environment. It also includes information provided by other Systems Manager capabilities, such as the State Manager association compliance and Patch Manager patch compliance details. The Systems Manager Explorer contains information from supporting AWS services as well like AWS Compute Optimizer, AWS Support AWS Config and AWS Trusted Advisor.

The Patch Manager automates the process of patching the OS of your EC2 instances and on-premises servers using predefined and custom patch baselines. You can set a scheduled maintenance window to execute the patching activities to reduce any operational impact via its Maintenance Window module. With the State

Manager, you can control the configuration details or the "state" of your resources, such as server configurations, virtualized hardware, and firewall settings. You can even associate Ansible playbooks, Chef recipes, PowerShell modules, and other SSM Documents with your resources.

The Systems Manager Parameter Store provides centralized storage and management of your "parameters" such as passwords, database strings, Amazon Machine Image IDs, license codes, environment variables, et cetera. You can store the parameter as a SecureString datatype to instruct SSM to encrypt it automatically using a KMS key in AWS KMS. Its OpsCenter module provides a central location where you can view, investigate, and resolve operational issues related to your AWS resources. This module also aggregates and standardizes operational issues while providing contextually relevant data to assist you in diagnosing and remediating your systems.

AWS Certificate Manager

AWS Certificate Manager (AWS ACM) is a service that lets you easily provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with AWS services and your internal connected resources. The AWS ACM service manages the lifecycle of certificates; from creating, storing, deploying, and managing renewals for AWS services like Amazon CloudFront, Amazon API Gateway and Elastic Load Balancing to name a few. It also enables you to create private certificates for your internal resources and manage the certificate lifecycle centrally. Public and private certificates provisioned through AWS Certificate Manager for use with ACM-integrated services are free.

There is also a separate service called AWS Private Certificate Authority (AWS Private CA) that is related with the AWS Certificate Manager. ACM and AWS Private CA have specific roles in the process of creating and managing the digital certificates that are used to identify resources and secure network communications that passes through over the public internet, on private networks and in the cloud. The AWS Private CA service enables you to create customizable private certificates for a broad range of scenarios, including for AWS services like ACM, Amazon Managed Streaming for Apache Kafka (MSK), IAM Roles Anywhere and Amazon Elastic Kubernetes Service (EKS). These services can all leverage private certificates from Private CA which is something that the AWS ACM service can’t fully provide. The AWS Private CA also supports creating private certificates for Internet of Things (IoT) devices, external systems, enterprise users and other services.

COMPARISON OF AWS SERVICES

Amazon S3 vs EBS vs EFS

Type of storage	Object storage. You can store virtually any kind of data in any format.	Persistent block level storage for EC2 instances.	POSIX-compliant file storage for EC2 instances.
Features	Accessible to anyone or any service with the right permissions	Deliver performance for workloads that require the lowest-latency access to data from a single EC2 instance	Has a file system interface, file system access semantics (such as strong consistency and file locking), and concurrently-accessible storage for multiple EC2 instances
Max Storage Style	Virtually unlimited	16 TiB for one volume	Unlimited system size
Max File Size	Individual Amazon S3 objects can range in size to a maximum of 5 terabytes.	Equivalent to the maximum size of your volumes	47.9 TiB for a single file
Performance (Latency)	Low, for mixed request types, and integration with CloudFront	Lowest, consistent; SSD-backed storages include the highest performance Provisioned OPS SSD and General Purpose SSD that balance price and performance.	Low, consistent; use Max I/O mode for higher performance
Performance (Throughput)	Multiple GBs per second; supports multi-part upload	Up to 2 GB per second. HDD-backed volumes include throughput intensive workloads and Cold HDD for less frequently accessed data.	10+ GB per second. Bursting Throughput mode scales with the scales with the size of the file system. Provisioned throughput mode

			offers higher dedicated throughput than bustring throughput.
Durability	Stored redundantly across multiple AZs; has 99.999999999% durability	Stored redundantly in a single AZ	Stored redundantly across multiple AZs
Availability	S3 Standard – 99.99% availability S3 Standard-IA – 99.9% availability S3 One Zone-IA – 99.5% availability. S3 Intelligent Tiering – 99.9%	Has 99.999% availability	99.9% SLA. Runs in multi – AZ
Scalability	Highly scalable	Manually increase/decrease your memory size. Attach and detach additional volumes to and from your EC2 instance to scale	EFS file systems are elastic, and automatically grow and shrink as you add and remove files.
Data Accessing	One to millions of connections over the wed; S3 provides a REST web services interface	Single EC2 instance in a single AZ Amazon EBS Multi-Attach a single Provisioned IOPS SSD (io1 or io2) volume to up to 16 Nitro-based instances that are in the same Availability Zone.	One to thousands of EC2 instances or on-premises servers, from multiple AZs, regions, VPCs, and accounts concurrently
Access Control	Uses bucket policies and IAM user policies. Has Block Public Access settings to help	IAM Policies, Roles, and Security Groups	Only resources that can access endpoints in your VPC, called a mount target, can access your file

	manage public access to resources.		system; POSIX-compliant user and group-level permissions.
Encryption Methods	Supports SSL endpoints using the HTTPS protocol, Client-Side and Server-Side Encryption (SSE-S3, SSE-C, SSE – KMS)	Encrypts both data-at-rest and data-in-transit through EBS encryption that uses AWS KMS key	Encrypt data at rest and in transit. Data at rest encryption uses AWS KMS. Data in transit uses TLS.
Backup and Restoration	Use versioning or cross-region replication	All EBS volume types offer durable snapshot capabilities.	EFS to EFS replication through third party tools or AWS DataSynch
Pricing	Billing prices are based on the location of your bucket. Lower costs equals lower prices. You get cheaper prices the more you use S3 storage.	You pay Gb-month of provisioned storage, provisioned IOPS-month, GB-month of snapshot data stored in S3	You pay more the amount of file system storage used per month. When using the Provisioned Throughput mode you pay for the throughput you provision per month.
Use cases	Web serving and content management, media and entertainment, backups, big data analytics, data lake	Boot volumes, transactional and NoSQL databases, data warehousing & ETL	Web serving and content management,enterprise applications, media and entertainment, home directories, database backups, developer tools, container storage, big data analytics
Service endpoint	Can be accessed within and outside a VPC (via S3 bucket URL)	Accessed within one's VPC	Accessed within one's VPC

S3 Standard vs S3 Standard-IA vs S3 One Zone-IA vs S3 Intelligent Tiering

Features	General-purpose storage of frequently accessed data.	For long-lived, rapid but less frequently accessed data; data is stored redundantly in multiple AZs.	For long-lived, rapid but less frequently accessed data; data is stored redundantly in only one AZ of your choice.	For long-lived data that have unpredictable access patterns.
Durability	99.999999999% (11 9's)			
Availability	99.99%	99.9%	99.5%	99.9%
Availability SLA	99.9%	99%	99%	99%
Number of Availability Zones	At least 3	At least 3	Only 1	At least 3
Minimum capacity charge per object	N/A	128KB	128KB	N/A
Minimum storage duration charge	N/A	30 days	30 Days	30 Days
Inserting data	Directly PUT into S3 Standard	Directly PUT into S3 Standard-IA or set Lifecycle policies to transition objects from the S3 Standard to the S3 Standard-IA storage class.	Directly PUT into S3 One Zone-IA or set Lifecycle policies to transition objects from the S3 Standard to the S3 One Zone-IA storage class.	Directly PUT into S3 Intelligent-Tiering or set Lifecycle policies to transition objects from the S3 Standard to the S3 Intelligent-Tiering storage class.
Retrieval fee	N/A	per GB retrieved	per GB retrieved	N/A
First byte latency	milliseconds			

Storage transition	S3 Standard to all other S3 storage types, including Glacier	S3 Standard-IA to S3 One Zone-IA or S3 Glacier	S3 One Zone-IA to S3 Glacier	S3 Intelligent to S3 One Zone-IA or S3 Glacier
Use cases	Cloud applications, dynamic websites, content distribution, mobile and gaming applications, and big data analytics.	Ideally suited for long-term file storage, older sync and share storage, and other aging data.	For infrequently-accessed storage, like backup copies, disaster recovery copies, or other easily recreatable data.	Data with unknown or changing access patterns, optimize storage costs automatically, and unpredictable workloads.

Additional Notes:

- Data stored in the S3 One Zone-IA storage class will be lost in the event of AZ destruction.
- S3 Standard-IA costs less than S3 Standard in terms of storage price, while still providing the same high durability, throughput, and low latency of S3 Standard.
- S3 One Zone-IA has 20% less cost than Standard-IA.
- It is recommended to use multipart upload for objects larger than 100MB.

Provisioned IOPS SSD (io1, io2) volumes are designed to meet the needs of I/O-intensive workloads, particularly database workloads, that are sensitive to storage performance and consistency. Unlike gp2, which uses a bucket and credit model to calculate performance, an io1 volume allows you to specify a consistent IOPS rate when you create the volume, and Amazon EBS delivers within 10 percent of the provisioned IOPS performance 99.9 percent of the time over a given year. Provisioned IOPS SSD io2 is an upgrade of Provisioned IOPS SSD io1. It offers higher 99.999% durability and higher IOPS per GiB ratio with 500 IOPS per GiB, all at the same cost as io1 volumes.

Volume type	gp3	gp2	io2	io1
Description	General Purpose SSD volume that balances price performance for a wide variety of transactional workloads	General Purpose SSD volume that balances price performance for a wide variety of transactional workloads	High-performance SSD volume designed for **business-critical** latency-sensitive applications	High-performance SSD volume designed for latency-sensitive transactional workloads
Use Cases	virtual desktops, medium sized single instance databases	Boot volumes, low-latency interactive apps, dev	Workloads that require sub-millisecond	Workloads that require sustained IOPS performance or

	such as MSFT SQL Server and Oracle DB, low-latency interactive apps, dev & test, boot volumes	& test	latency, and sustained IOPS performance or more than 64,000 IOPS or 1,000 MiB/s of throughput	more than 16,000 IOPS and I/O-intensive database workloads
Volume Size	1 GB – 16 TB	1 GB – 16 TB	4 GB – 16 TB	4 GB – 16 TB
Durability	99.8% - 99.9% durability	99.8% - 99.9% durability	99.999%	99.8% - 99.9%
Max IOPS / Volume	16,000	16,000	64,000	64,000
Max Throughput / Volume	1000 MB/s	250 MB/s	1,000 MB/s	1,000 MB/s
Max IOPS / Instance	260,000	260,000	160,000	260,000
Max IOPS / GB	N/A	N/A	500 IOPS/GB	50 IOPS/GB
Max Throughput / Instance	7,500 MB/s	7,500 MB/s	4,750 MB/s	7,500 MB/s
Latency	single digit millisecond	single digit millisecond	single digit millisecond	single digit millisecond
Multi-Attach	No	No	Yes	Yes

Volume type	st1	sc1
Description	Low cost HDD volume designed for frequently accessed, throughput-intensive workloads	Throughput-oriented storage for data that is infrequently accessed Scenarios where the lowest storage cost is important
Use Cases	Big data, data warehouses, log processing	Colder data requiring fewer scans per day
Volume Size	125 GB – 16 TB	125 GB – 16 TB
Durability	99.8% - 99.9% durability	99.8% - 99.9% durability
Max IOPS / Volume	500	250

Max Throughput / Volume	500 MB/s	250 MB/s
Max IOPS / Instance	260,000	260,000
Max IOPS / GB	N/A	N/A
Max Throughput / Instance	7,500 MB/s	7,500 MB/s
Multi-Attach	No	No

RDS vs DynamoDB

	RDS	DynamoDB
Type of databases	Managed relational (SQL) database	Fully managed key-value and document (NoSQL) database
Features	Has several database instance types for different kinds of workloads and supports six database engines – Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, and Mircrodoft SQL Server.	Delivers single-digit millisecond performance at any scale.
Storage Size	-128 TB for Aurora engine. -64 TB for MySQL, MariaDB, Oracle and PostgreSQL engines. -16 TB for SQL Server engine.	Supports tables of virtually any size.
Number of tables per unit	Depends on the database engine	256
Performance	General Purpose Storage is an SSD-backed storage option that delivers at consistent baseline of 3 IOPS per provisioned GB	Single-digit millisecond read and write performance. Can handle more than 10 trillion requests per day with peaks greater than 20 million request per second, over petabytes ofstorage.

	with the ability to burst up to 3,000 IOPS. Provisioned IOPS Storage is an SSD-backed storage option designed to deliver a consistent IOPS rate that you specify when creating a database instance, up to 40,000 IOPS per database Instance. Amazon RDS provisions that IOPS rate for the lifetime of the database instance. Optimized for OLTP database workloads. Magnetic – Amazon RDS also supports magnetic storage for backward compatibility.	DynamoDB Accelerator (DAX) is an in-memory cache that can improve the read performance of your DynamoDB tables by up to 10 times – taking the time required for reads from milliseconds to microseconds, even at millions of requests per second. You specify the read and write throughput for each of your tables.
Availability and durability	Amazon RDS Multi-AZ deployments synchronously replicates your data to a standby instance in a different Availability Zone Amazon RDS will automatically replace the compute instance powering your deployment in the event of a hardware failure.	DynamoDB global tables replicate your data automatically across 3 Availability Zones of your choice of AWS Regions and automatically scale capacity to accommodate your workloads.
Backups	The automated backup feature enables point-in-time recovery for your database instance. Database snapshots are user-initiated backups of your instance stored in Amazon S3 that are kept until you explicitly delete them.	Point-in-time recovery (PITR) provides continuous backups of your DynamoDB table data, and you can restore that table to any point in time up to the second during the preceding 35 days. On-demand backups and restore allows you to create full backups of your DynamoDB tables' data for data archiving.
Scalability	The Amazon Aurora engine will automatically grow the size of your database volume. The	Support tables of virtually any size with horizontal scaling.

	MySQL, MariaDB, SQL Server, Oracle, and PostgreSQL engines allow you to scale on-the-fly with zero downtime. RDS also supports storage auto scaling Reads replicas are available in Amazon RDS for MySQL, MariaDB, and PostgreSQL as well as Amazon Aurora.	For tables using on-demand capacity mode, DynamoDB instantly accommodates your workloads a they ramp up or down to any previously reached traffic level. For tables using provisioned capacity, DynamoDB delivers automatic scaling of throughput and storage based on your previously set capacity.
Security	Isolate your database in your own virtual network. Connect to your on-premises IT infrastructure using industry-standard encrypted IPsec VPNs. You can configure firewall settings and control network access to your database instances. Integrates with IAM.	Integrates with IAM.
Encryption	Encrypt your databases using keys you manage through AWS KMS. With encryption enabled, data stored at rest is encrypted, as are its automated backups, read replicas, and snapshots. Supports Transparent Data Encryption in SQL Server and Oracle. Supports the use of SSL to secure data in transit.	DynamoDB encrypts data at rest by default using encryption keys stored in AWS KMS.
Maintenance	Amazon RDS will update databases with the latest patches. You can exert optional	No maintenance since DynamoDB is serverless.

	control over when and if your database instance is patched.	
Pricing	A monthly charge for each database instance that you launch. Option to reserve a DB instance for a One or three year term and receive discounts in pricing, compared to On-Demand instance pricing.	Charges for reading, writing, and storing data in your DynamoDb tables, along with any optional features you choose to enable. There are specific billing options for each of DynamoDB's capacity modes.
Use cases	Traditional applications, ERP, CRM, and e-commerce.	Internet-scale applications, real-time bidding, shopping carts, and customer Preferences, content management, Personalization, and mobile applications.

Additional notes:

- DynamoDB has built-in support for ACID transactions.
- DynamoDB uses filter expressions because it does not support complex queries.
- Multi-AZ deployments for the MySQL, MariaDB, Oracle, and PostgreSQL engines utilize synchronous physical replication. Multi-AZ deployments for the SQL Server engine use synchronous logical replication.

Global Secondary Index vs Local Secondary Index

A *secondary index* is a data structure that contains a subset of attributes from a table, along with an alternate key to support Query operations. An Amazon DynamoDB table can have multiple secondary indexes.

	Global secondary index	Local secondary index
Definition	An index with a partition key and a sort key that can be different from those on the base table.	An index that has the same partition key as the base table, but a different sort key.
Span of query	Queries on the index can span all of the data in the base table, across all partitions.	Every partition of a local secondary index is scoped to a base table partition that has the same partition key value.
Primary Key Schema	The partition key and, optionally, the sort key	Must be both partition key and sort key
Partition Key Attributes	Any base table attribute of type string, number, or binary.	Must have the same attribute as the partition key of the base table.
Sort Key Attributes	Any base table attribute of type string, number, or binary.	Any base table attribute of type string, number, or binary.
Key Values	Do not need to be unique	The sort key value does not need to be unique for a given partition key value.
Size Restrictions Per Partition Key Value	No restriction.	For each partition key value, the total size of all indexed items must be 10 GB or less.
Index Operations	• Can be created during creation of a table. • Can be created on an existing table. • Can be deleted from an existing table.	• Can be created during creation of a table.
Read Consistency for Queries	Supports eventual consistency only from asynchronous updates and deletes	You can choose either eventual consistency or strong consistency.
Provisioned	Every global secondary index has its own	Queries or scans on a local

Throughput Consumption	provisioned throughput settings for read and write activity that you need to specify. Queries or scans on a global secondary index consume capacity units from the index, not from the base table. This is also the case for global secondary index updates due to table writes.	secondary index consume read capacity units from the base table. When you write to a table and its local secondary indexes are updated, these updates consume write capacity units from the base table.
Projected Attributes	With global secondary index queries or scans, you can only request the attributes that are projected into the index.	If you query or scan a local secondary index, you can request attributes that are not projected into the index. DynamoDB will automatically fetch those attributes from the table.
Index limit (default)	20 indexes	5 indexes per table

Global Secondary Index Read/Write Capacity Calculation (Provisioned Throughput Mode)

- **Eventually consistent reads consume ½ read capacity unit.** Therefore, each query can retrieve up to 8KB of data per capacity unit (4KB x 2). The maximum size of the results returned by a Query operation is 1 MB.
- **The total provisioned throughput cost for a write consists of the sum of write capacity units consumed by writing to the base table and those consumed by updating the global secondary indexes.** If a write to a table does not require a global secondary index update, then no write capacity is consumed from the index.
- Cost of a write operation depends on the ff factors (assuming up to 1 KB item size):
    - If you write a new item to the table that defines an indexed attribute, or you update an existing item to define a previously undefined indexed attribute, one write operation is required to put the item into the index.
    - If an update to the table changes the value of an indexed key attribute, two writes are required, one to delete the previous item from the index and another write to put the new item into the index.
    - If an item was present in the index, but a write to the table caused the indexed attribute to be deleted, one write is required to delete the old item projection from the index.
    - If an item is not present in the index before or after the item is updated, there is no additional write cost for the index.
    - If an update to the table only changes the value of projected attributes in the index key schema but does not change the value of any indexed key attribute, then one write is required to update the values of the projected attributes into the index.

Local Secondary Index Read/Write Capacity Calculation (Provisioned Throughput Mode)

- An index query can use either eventually consistent or strongly consistent reads. **One strongly consistent read consumes one read capacity unit; an eventually consistent read consumes only half of that.** The number of read capacity units is the sum of all projected attribute sizes across all of the items returned (from index and base table), and the result is then rounded up to the next 4 KB multiple. The maximum size of the results returned by a Query operation is 1 MB.
- **The total provisioned throughput cost for a write is the sum of write capacity units consumed by writing to the table and those consumed by updating the local secondary indexes.**
- Cost of a write operation depends on the ff factors (assuming up to 1 KB item size):
    - If you write a new item to the table that defines an indexed attribute, or you update an existing item to define a previously undefined indexed attribute, one write operation is required to put the item into the index.
    - If an update to the table changes the value of an indexed key attribute, two writes are required, one to delete the previous item from the index and another write to put the new item into the index.
    - If an item was present in the index, but a write to the table caused the indexed attribute to be deleted, one write is required to delete the old item projection from the index.
    - If an item is not present in the index before or after the item is updated, there is no additional write cost for the index.

EC2 Container Services ECS vs Lambda

Amazon ECS is a highly scalable, high performance container management service that supports Docker containers and allows you to easily run applications on a managed cluster of Amazon EC2 instances. ECS eliminates the need for you to install, operate, and scale your own cluster management infrastructure.	AWS Lambda is a function-as-a-service offering that runs your code in response to events and automatically manages the compute resources for you, since Lambda is a serverless compute service. With Lambda, you do not have to worry about managing servers, and directly focus on your application code.
With ECS, deploying containerized applications is easily accomplished. This service fits well in running batch jobs or in a microservice architecture.	Lambda automatically scales your function to meet demands. It is noteworthy, however, that Lambda has a maximum execution duration per request of 900 seconds or 15 minutes.
You have a central repository where you can upload your Docker Images from ECS container for safekeeping called Amazon ECR.	To allow your Lambda function to access other services such as Cloudwatch Logs, you would need to create an execution role that has the necessary permissions to do so.
Applications in ECS can be written in a stateful or stateless matter.	You can easily integrate your function with different services such as API Gateway, DynamoDB, CloudFront, etc. using the Lambda console.
The Amazon ECS CLI supports Docker Compose, which allows you to simplify your local development experience as well as easily set up and run your containers on Amazon ECS.	You can test your function code locally in the Lambda console before launching it into production.
Since your applications still run on EC2 instances, server management is your responsibility. This gives you more granular control over your system.	Currently, Lambda supports only a number of programming languages such as Java, Go, PowerShell, Node.js, C#, Python, and Ruby. ECS is not limited by programming languages since it mainly caters to Docker.
It is up to you to manage scaling and load balancing of your EC2 instances as well, unlike in AWS Lambda where functions scale automatically.	Lambda functions must be stateless since you do not have volumes for data storage.
You are charged for the costs incurred by your EC2 instances in your clusters. Most of the time,	You are charged based on the number of requests for your functions and the duration, the time it takes

Amazon ECS costs more than using AWS Lambda since your active EC2 instances will be charged by the hour.	for your code to execute. To minimize costs, you can throttle the number of concurrent executions running at a time, and the execution time limit of the function.
One version of Amazon ECS, know as AWS Fargate, will fully manage your infrastructure so you can just focus on deploying containers. AWS Fargate has a different pricing model from the standard EC2 cluster.	With Lambda@Edge, AWS Lambda can run your code across AWS locations globally in response to Amazon CloudFront events, such as requests for content to or from origin servers and viewers. This makes it easier to deliver content to end users with lower latency.
ECS will automatically recover unhealthy containers to ensure that you have the desired number of containers supporting your application.	

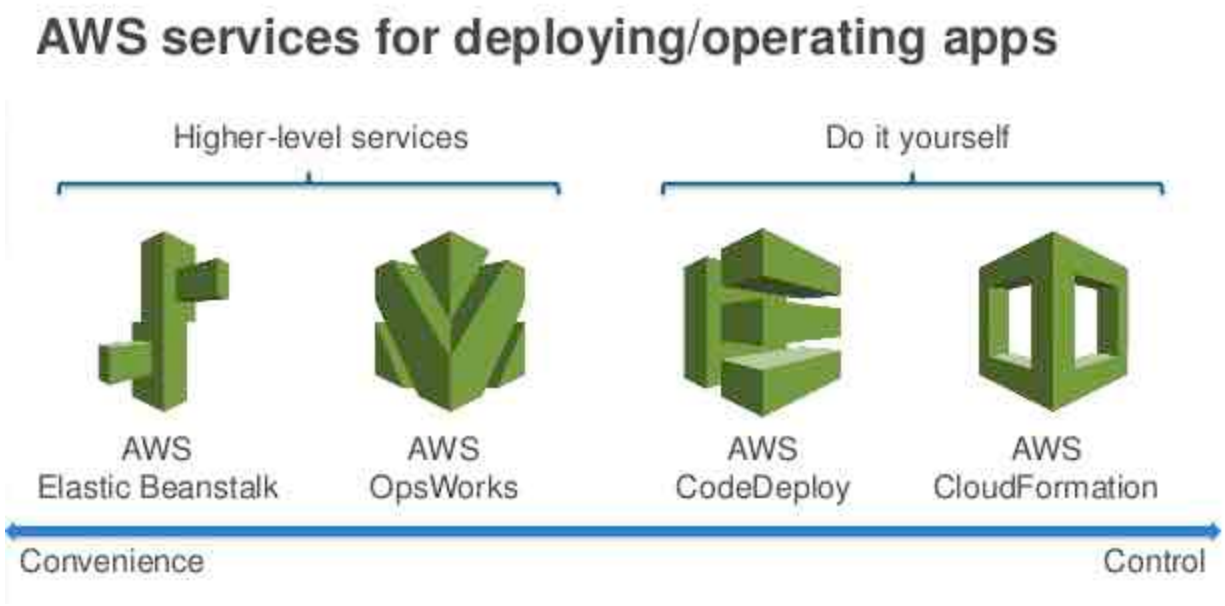
Elastic Beanstalk vs CloudFormation vs CodeDeploy

• AWS Elastic Beanstalk makes it even easier for developers to **quickly deploy and manage applications** in the AWS Cloud. Developers simply upload their applications, and Elastic Beanstalk **automatically handles the deployment details** of capacity provisioning, load balancing, auto-scaling, and application health monitoring.
• This **platform-as-a-service solution** is typically for those who want to deploy and manage their applications within minutes in the AWS Cloud without worrying about the underlying infrastructure.
• AWS Elastic Beanstalk supports the following languages and development stacks:
  ○ Apache Tomcat for Java applications
  ○ Apache HTTP Server for PHP applications
  ○ Apache HTTP Server for Python applications
  ○ Nginx or Apache HTTP Server for Node.js applications
  ○ Passenger or Puma for Ruby applications
  ○ Microsoft IIS for .NET applications
  ○ Java SE
  ○ Docker
  ○ Go
• Elastic Beanstalk also supports deployment versioning. It maintains a copy of older deployments so that it is easy for the developer to rollback any changes made to the application.
	• AWS CloudFormation is a service that gives developers and businesses an easy way to create a **collection of related AWS resources** and provision them in an orderly and predictable fashion. This is typically known as "**infrastructure as code**".
	• The main difference between CloudFormation and Elastic Beanstalk is that CloudFormation deals more with the AWS infrastructure rather than applications.
	• AWS CloudFormation introduces two concepts:
	  ○ The **template**, a JSON or YAML-format, text-based file that describes all the AWS resources and configurations you need to deploy to run your application.
	  ○ The **stack**, which is the set of AWS resources that are created and managed as a single unit when AWS CloudFormation instantiates a template.
	• CloudFormation also supports a rollback feature through template version controls. When you try to update your stack, but the deployment fails midway, CloudFormation will automatically revert the changes to their previous working states.
	• CloudFormation supports Elastic Beanstalk application environments. This allows you, for example, to create and manage an AWS Elastic Beanstalk–hosted application along with an RDS database to store the application data.
	• AWS CloudFormation can be used to bootstrap both Chef (Server and Client) and Puppet (Master and Client) software on your EC2 instances.
	• AWS CodeDeploy is a recommended adjunct

	to CloudFormation for managing application deployments and updates.

Additional Notes:

- Elastic Beanstalk, CloudFormation are particularly useful for the **blue-green deployment method** as they provide a simple way to clone your running application stack.
- CloudFormation is best suited for pre-baking AMIs.
- CodeDeploy is best suited for performing in-place application upgrades. For disposable upgrades, a cloned environment can be set up with Elastic Beanstalk and CloudFormation.



References:
https://aws.amazon.com/CloudFormation/faqs/
https://aws.amazon.com/elasticbeanstalk/faqs/
https://d1.awsstatic.com/whitepapers/overview-of-deployment-options-on-aws.pdf
https://aws.amazon.com/codedeploy/faqs/

Security Group vs NACL

Security Group	Network Access Control List
Acts as a firewall for associated Amazon EC2 instance	Acts as firewall for associated subnets
Control both inbound and outbound traffic at the instance level	Controls both inbound and outbound traffic at the subnet level
You can secure your VPC instances using only security groups	Network ACLs are additional layer of defense
Supports allow rules only	Supports allow and deny rules
Stateful (Return traffic is automatically allowed, regardless of any rules)	Stateless (Return traffic must be explicitly allowed by rules
Evaluates all rules before deciding whether to allow traffic	Evaluates rules in number order when deciding whether to allow traffic, starting with the lowest numbered rule.
Applies only to the instance associated to it	Applies to all instances in the subnet it is associated with
Has separate rules for inbound and outbound traffic	Has separate rules for inbound and outbound traffic
A newly created security group denies all inbound traffic by default	A newly created nACL denies all inbound traffic by default.
A newly created security group has an outbound rule that allows all outbound traffic by default	A newly created nACL denies all outbound traffic by default.
Instances associated with a security group can't talk to each other unless you add rules allowing it.	Each subnet in your VPC must be associated with a network ACL. If none is associated, the default nACL is selected.
Security groups are associated with network interfaces.	You can associate a network ACL with multiple subnets; however, a subnet can be associated with only one network ACL at a time

Your VPC has a default security group with the following rules:

1. Allow inbound traffic from instances assigned to the same security group.
2. Allow all outbound IPv4 traffic and IPv6 traffic if you have allocated an IPv6 CIDR block.

Your VPC has a default network ACL with the following rules:

1. Allows all inbound and outbound IPv4 traffic and, if applicable, IPv6 traffic.
2. Each network ACL also includes a non-modifiable and non-removable rule whose rule number is an asterisk. This rule ensures that if a packet doesn't match any of the other numbered rules, it's denied.

Application Load Balancer vs Network Load Balancer vs Gateway Load Balancer

FEATURE	Application Load Balancer	Network Load Balancer	Gateway Load Balancer
Protocols	HTTP, HTTPS, gRPC	TCP, UDP, TLS	IP
Platforms	VPC	VPC	VPC
Health checks	HTTP, HTTPS, gRPC...	TCP, HTTP, HTTPS	TCP, HTTP, HTTPS
Cloudwatch Metrics	Yes	Yes	Yes
Logging	Yes	Yes	Yes
Zonal Failover	Yes	Yes	Yes
Connection Draining (deregistration delay)	Yes	Yes	Yes
Load Balancing to multiple ports on the same instance	Yes	Yes	Yes
IP addresses as targets	Yes	Yes (TCP, TLS)	Yes
Load Balancer deletion protection	Yes		
Configuration idle connection timeout	Yes		
Cross-zone load balancing	Yes	Yes	Yes

Sticky sessions	Yes	Yes	Yes
Static IP		Yes	
Elastic IP address		Yes	
Preserve Source IP address	Yes	Yes	Yes
Resource-based IAM permissions	Yes	Yes	Yes
Slow start	Yes		
Web sockets	Yes	Yes	Yes
PrivateLink Support		Yes (TCP, TLS)	Yes (GWLBE)
Source IP address CIDR-based routing	Yes		
Layer 7			
Path-based routing	Yes		
Host-based routing	Yes		
Native HTTP/2	Yes		
Redirects	Yes		
Fixed response	Yes		
Lambda Functions as targets	Yes		
HTTP header-based routing	Yes		
HTTP method-based routing	Yes		
Query string parameter-based routing	Yes		
Security			
SSL offloading	Yes	Yes	
Server Name Indication (SNI)	Yes	Yes	
Back-end server encryption	Yes	Yes	

User authentication	Yes		
Session Resumption	Yes	Yes	
Terminates flow/ proxy behavior	Yes	Yes	Yes

Common features between the load balancers:

- Has instance health check features
- Has built-in CloudWatch monitoring
- Logging features
- Support zonal failover
- Supports connection draining
- Support cross-zone load balancing (evenly distributes traffic across registered instances in enabled AZs)
- Resource-based IAM permission policies
- Tag-based IAM permissions
- Flow stickiness - all packets are sent to one target and return the traffic that comes from the same target.

CloudTrail vs CloudWatch

- CloudWatch is a monitoring service for AWS resources and applications. CloudTrail is a web service that records API activity in your AWS account. They are both useful monitoring tools in AWS.
- By default, CloudWatch offers free basic monitoring for your resources, such as EC2 instances, EBS volumes, and RDS DB instances. CloudTrail is also enabled by default when you create your AWS account.
- With CloudWatch, you can collect and track metrics, collect and monitor log files, and set alarms. CloudTrail, on the other hand, logs information on who made a request, the services used, the actions performed, parameters for the actions, and the response elements returned by the AWS service. CloudTrail Logs are then stored in an S3 bucket or a CloudWatch Logs log group that you specify.
- You can enable detailed monitoring from your AWS resources to send metric data to CloudWatch more frequently, with an additional cost.
- CloudTrail delivers one free copy of management event logs for each AWS region. Management events include management operations performed on resources in your AWS account, such as when a user logs in to your account. Logging data events are charged. Data events include resource operations performed on or within the resource itself, such as S3 object-level API activity or Lambda function execution activity.
- CloudTrail helps you ensure compliance and regulatory standards.
- CloudWatch Logs reports on application logs, while CloudTrail Logs provides specific information on what occurred in your AWS account.
- EventBridge (formerly called CloudWatch Events) is a near real-time stream of system events describing changes to your AWS resources. CloudTrail focuses more on AWS API calls made in your AWS account.
- Typically, CloudTrail delivers an event within 15 minutes of the API call. CloudWatch delivers metric data in 5-minute periods for basic monitoring and 1-minute periods for detailed monitoring. The CloudWatch Logs Agent will send log data every five seconds by default.

FINAL REMARKS AND TIPS

Thanks to the innovations brought by the cloud, developers like yourself can work with less inconvenience and more flexibility. It is an exciting time to be in the development field, not only for experienced developers but also for those new to the space. We do believe that there are practices in software development that should continue to be applied even after moving from an on-premises environment to AWS. Although AWS shares what they believe are the best practices for developing applications in their environment, the main benefit actually of the cloud is having the freedom to implement what you think is best for your applications. Additionally, there is less worry with experimentation because resources in AWS are replaceable and should be treated as such. This gives you the opportunity to figure out what works best for your workloads and implement it in a cost-effective manner.

Hopefully, our cheat sheets have helped you a lot in your review for the AWS Certified Developer Associate exam. This AWS certificate is sought after by developers all over the world since it proves that they have the knowledge and skills to develop applications properly in AWS. Passing an AWS certification exam is no easy feat and the AWS community celebrates exam passers with pride and joy. We at Tutorials Dojo are dedicated to helping you achieve these results too. We create our content specifically to help you prepare for your exams and equip you with the important information so you can be successful in your role. We have written blogs, guides, cheat sheets, and practice exams that are also constantly being updated based on our experiences and the feedback of our students. Your feedback is very important to us since it helps us improve our content and serve the community better.

And with that, we at Tutorials Dojo thank you for supporting us through this eBook. If you wish to validate what you have learned so far, now is a great time to check out our AWS Certified Developer Associate Practice Exam as well. You can also try the free sampler version of our full practice test course here. It will fill in the gaps in your knowledge that you are unaware of and give you a sense of the actual exam environment. That way, you'll know what to expect in the actual exam, and you can pace yourself through the questions better. If you have any issues, concerns, or constructive feedback on our eBook, feel free to contact us at support@tutorialsdojo.com.

Good luck with your exam and we'd love to hear back from you.

Your Learning Partners,
Jon Bonso, Carlo Acebedo, and the Tutorials Dojo Team

ABOUT THE AUTHORS

Jon Bonso (10x AWS Certified)

Born and raised in the Philippines, Jon is the Co-Founder of **Tutorials Dojo**. Now based in Sydney, Australia, he has over a decade of diversified experience in Banking, Financial Services, and Telecommunications. He's 10x AWS Certified, an AWS Community Builder, and has worked with various cloud services such as Google Cloud, and Microsoft Azure. Jon is passionate about what he does and dedicates a lot of time creating educational courses. He has given IT seminars to different universities in the Philippines for free and has launched educational websites using his own money and without any external funding.

Carlo Acebedo (5x AWS Certified)

Carlo is a cloud engineer and a content creator at Tutorials Dojo. He's also a member of the AWS Community builder and holds 5 AWS Certifications. Carlo specializes in building and automating solutions in the Amazon Web Services Cloud.