

JON BONSO AND CARLO ACEBEDO

---

**AWS CERTIFIED  
DEVELOPER  
ASSOCIATE  
DVA-C02  
EXAM**

---



**Tutorials Dojo Study Guide**



## TABLE OF CONTENTS

<b>INTRODUCTION</b>	<b>6</b>
<b>AWS CERTIFIED DEVELOPER ASSOCIATE EXAM OVERVIEW</b>	<b>7</b>
Exam Details	7
Exam Domains	8
Exam Scoring	9
<b>AWS CERTIFIED DEVELOPER ASSOCIATE EXAM - STUDY GUIDE AND TIPS</b>	<b>11</b>
Study Materials	11
AWS Services to Focus On	13
Common Exam Scenarios	16
<b>AWS Deep Dives</b>	<b>20</b>
AWS Identity Access Manager (IAM)	20
IAM Identities	20
IAM Policy	22
IAM Policy Structure	22
Identity-based Policy vs. Resource-based Policy	24
Cross-account access	27
IAM:PassRole Permission	31
AWS STS	34
STS API Operations	34
AWS Lambda	36
Synchronous vs. Asynchronous Invocations	37
Event source mappings	41
Event filtering	42
Execution Environment Lifecycle	44
Reducing Cold Starts	47
Execution Environment Reuse	47
Environment variables	48
Lambda function URL	49
Deploying Codes with External Dependencies	50
Lambda Layers	51
Concurrency and Throttling	52
Connecting a Lambda Function to a VPC	53
Versions and Aliases	55
Amazon API Gateway	57



REST API vs. HTTP API vs. Websocket API	58
Proxy vs. Non-proxy integration	58
Stage variables	59
Mapping Templates	60
Invalidating Cache	62
Cross-Origin Resource Sharing (CORS)	62
Authorizers	64
Usage Plans	66
Amazon DynamoDB	67
Core Components	67
Local Secondary Index vs. Global Secondary Index (LSI vs. GSI)	68
Projections	69
Scan & Query operations	69
Calculating the Required Read and Write Capacity Unit for your DynamoDB Table	70
DynamoDB Streams	73
DynamoDB Accelerator (DAX)	75
DynamoDB Transactions	77
DynamoDB Time-To-Live (TTL)	78
DynamoDB Global Tables	79
AWS CloudFormation	81
Stack	81
Parts of CloudFormation Template	81
Intrinsic Functions and Pseudo Parameters	82
Using dynamic references to securely pass credentials	84
AWS Serverless Application Model (SAM)	86
SAM Template	86
Commonly used SAM CLI commands	87
AWS Step Functions	88
States	88
Input and Output Processing	89
AWS ElasticBeanstalk	96
Deployment policies	96
Blue/Green deployment	102
.ebextensions	104
Application Lifecycle Policy	105
EB Command Line Interface (CLI)	107



Amazon Simple Queue Service (SQS)	108
Standard vs. FIFO queue	108
Concepts	108
Amazon Cognito	110
User Pool vs. Identity Pool	110
Granting access for unauthenticated Identities	114
AWS Amplify	115
Amplify Studio	115
Amplify Hosting	115
AWS Amplify Libraries	116
Amplify Command Line Interface	116
AWS CodeCommit	116
Authenticating access (HTTPS) on CodeCommit	117
AWS CodeBuild	118
Buildspec file	118
Integrations with other AWS Services	118
AWS CodeDeploy	120
Deployment configurations	120
AppSpec file	121
Deployment groups	123
AWS CodePipeline	124
Manual approval actions	124
AWS CodeArtifact	126
AWS CodeStar	127
Amazon CodeGuru	128
AWS Key Management Service (KMS)	129
AWS KMS Key	129
Envelope Encryption	130
KMS API	130
Amazon CloudFront	132
Viewer Protocol Policy vs Origin Protocol Policy	132
CloudFront event triggers	133
Lambda@Edge vs CloudFront functions	135
Amazon S3	136
Object Storage Classes	136
Minimum Storage Duration	138



Amazon S3 Encryption	138
S3 Event Notifications	139
S3 Object Lambda	141
Other key S3 bucket features	141
AWS X-Ray	143
Key concepts	144
Amazon EventBridge	146
Cross-Account Access to Event Bus	146
Amazon CloudWatch	148
Publishing Custom Metrics	148
Amazon CloudTrail	150
Concepts	150
AWS Secrets Manager	152
AWS Systems Manager Parameter Store	153
Amazon EC2	154
Instance User Data vs Instance Metadata	154
Auto Scaling Group	157
EC2 Image Builder	158
Amazon Elastic Load Balancing (ELB)	159
Load Balancer Types	159
Network Load Balancer (NLB)	159
Gateway Load Balancer (GWLB)	160
ELB components	161
Application Load Balancer Listener Rule Conditions	161
Multi-Value Headers	164
Preserving Client's IP address using the X-Forwarded-For header	164
Amazon ElastiCache	166
Memcached vs. Redis	166
Caching Strategies	166
Amazon Kinesis	171
Kinesis Data Stream	171
Shards	171
Provisioned vs On-demand capacity mode	171
Writing and Reading data in shards	172
Scaling, Resharding and Parallel Processing	174
Kinesis Data Firehose	175



Amazon Elastic Container Service (ECS)	175
Amazon ECS Container Instance Role vs Task Execution Role vs Task Role	175
Amazon OpenSearch Service (successor to Amazon Elasticsearch Service)	177
Amazon Elastic Kubernetes Services (Amazon EKS)	178
Amazon Athena	178
AWS Cloud Development Kit (CDK)	180
Amazon Route 53	181
DNS Management	181
Traffic Management	183
AWS Web Application Firewall (WAF)	183
Amazon MemoryDB for Redis	185
AWS AppConfig	186
AWS AppSync	187
AWS Systems Manager	188
AWS Certificate Manager	189
<b>COMPARISON OF AWS SERVICES</b>	<b>190</b>
S3 Standard vs S3 Standard-IA vs S3 One Zone-IA vs S3 Intelligent Tiering	191
RDS vs DynamoDB	194
Global Secondary Index vs Local Secondary Index	197
EC2 Container Services ECS vs Lambda	200
Elastic Beanstalk vs CloudFormation vs OpsWorks vs CodeDeploy	201
Security Group vs NACL	204
Application Load Balancer vs Network Load Balancer vs Gateway Load Balancer	206
CloudTrail vs CloudWatch	209
<b>FINAL REMARKS AND TIPS</b>	<b>210</b>
<b>ABOUT THE AUTHORS</b>	<b>211</b>



## 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 - 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.

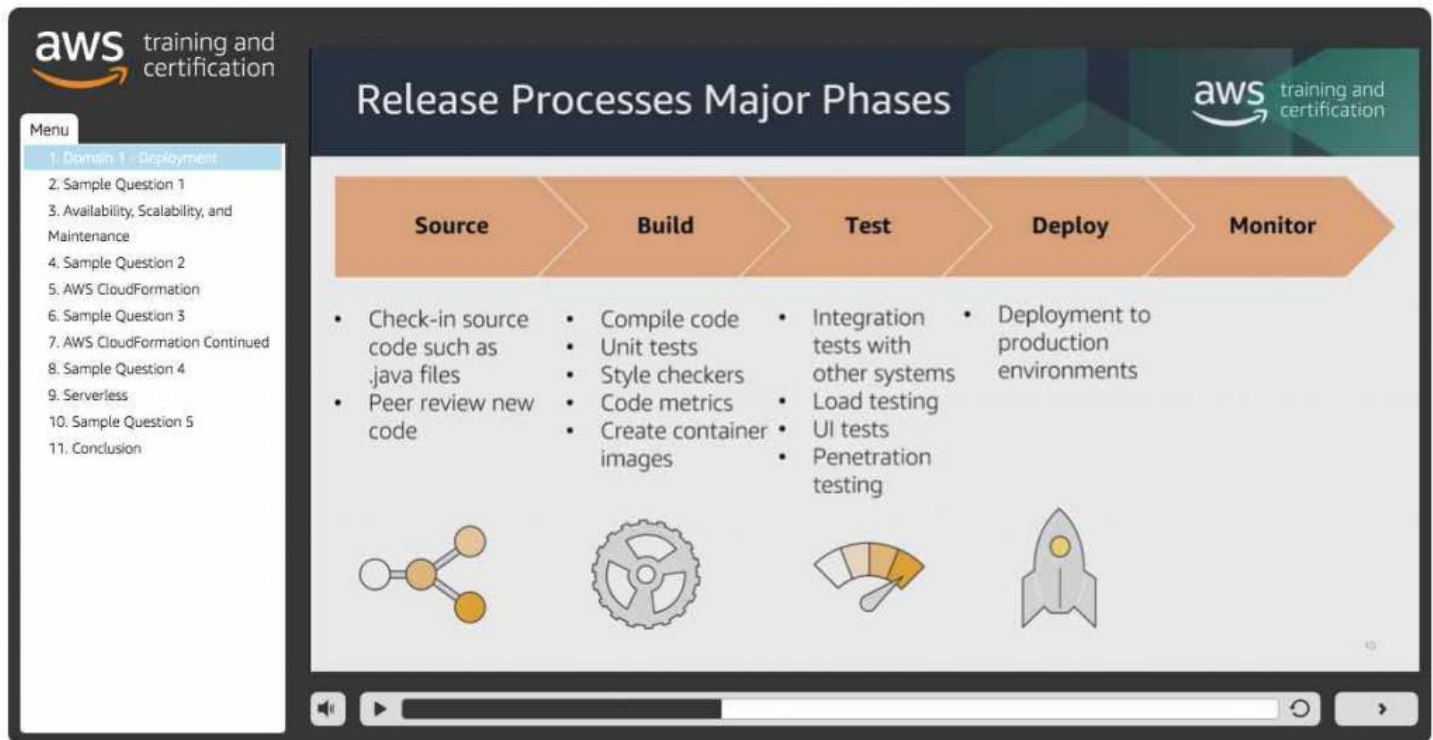
Having prior experience in programming and scripting for both standard, containerized, and/or serverless applications will greatly make your review easier. Additionally, we recommend having an AWS account available for you to play around with to better visualize parts in your review that involve code. 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 or OpsWorks. 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 CodeCommit / 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. CodeCommit is a git repository, so having knowledge in Git will be beneficial. 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 Server Migration Services](#), 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 <i>Unable to import module</i> 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 <code>/tmp</code> 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' header is present on the requested resource" error message.	Enable CORS in the API Gateway Console.
A website integrated with API Gateway requires user requests to reach the backend server <b>without intervention</b> 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?	<ol style="list-style-type: none"><li>1. Send a request with a <code>Cache-Control: max-age</code> header.</li><li>2. Enable the <code>Require Authorization</code> option on your API cache settings.</li></ol>
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 DynamoDB 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 <code>ProvisionedThroughputExceededException</code> 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
<b>Deployment and Security</b>	
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 AWS CodeCommit 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
<b>Relevant API/CLI commands</b>	
A Developer needs to decode an encoded authorization failure message.	Use the <code>aws sts decode-authorization-message</code> command.
How can a Developer verify permission to call a CLI command without actually making a request?	Use the <code>--dry-run</code> parameter along with the CLI command.
A Developer needs to deploy a CloudFormation template from a local computer.	Use the <code>aws cloudformation package</code> and <code>aws cloudformation deploy</code> 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 <code>VisibilityTimeout</code> value using the <code>ChangeMessageVisibility</code> API and delete the message using the <code>DeleteMessage</code> 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

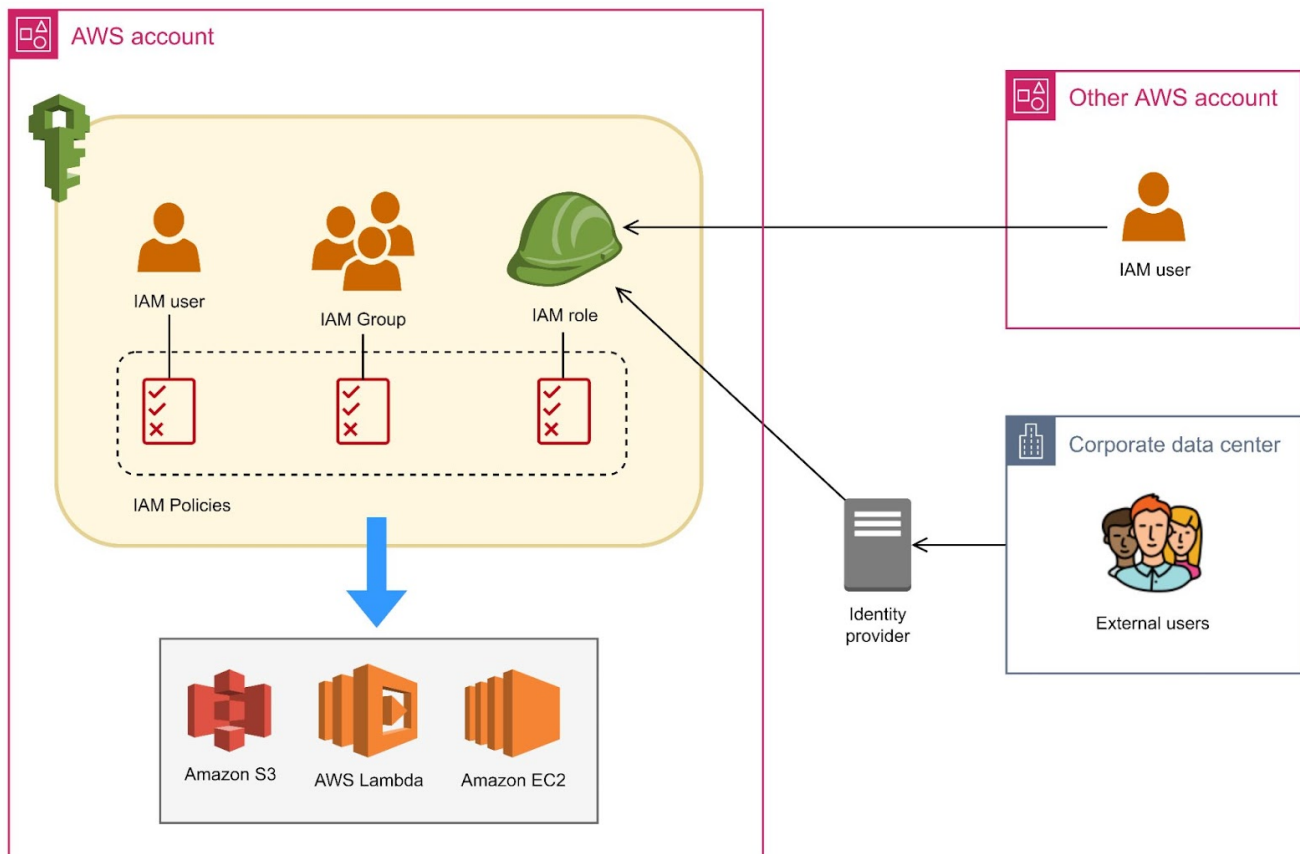
## AWS Deep Dives

### AWS Identity Access Manager (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.

### Select type of trusted entity

 <b>AWS service</b> EC2, Lambda and others	 <b>Another AWS account</b> Belonging to you or 3rd party	 <b>Web identity</b> Cognito or any OpenID provider	 <b>SAML 2.0 federation</b> Your corporate directory
--	---	---	--

Allows AWS services to perform actions on your behalf. [Learn more](#)

### Choose a use case

#### Common use cases

##### EC2

Allows EC2 instances to call AWS services on your behalf.

##### Lambda

Allows Lambda functions to call AWS services on your behalf.

**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/id_users.html)

[https://docs.aws.amazon.com/IAM/latest/UserGuide/access\\_policies.html](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html)

[https://docs.aws.amazon.com/IAM/latest/UserGuide/id\\_groups.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:

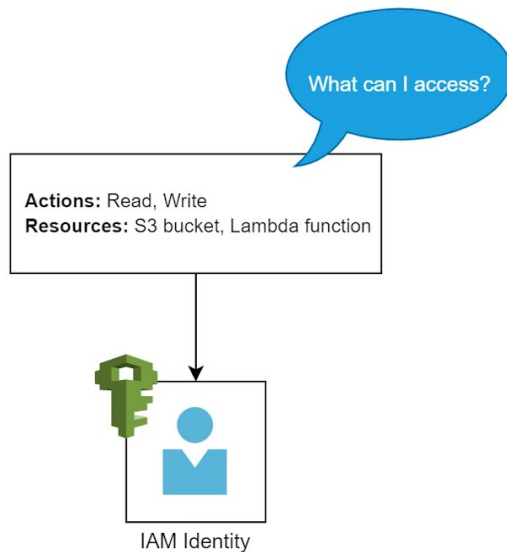
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-policy-structure.html>

<https://aws.amazon.com/blogs/security/back-to-school-understanding-the-iam-policy-grammar/>

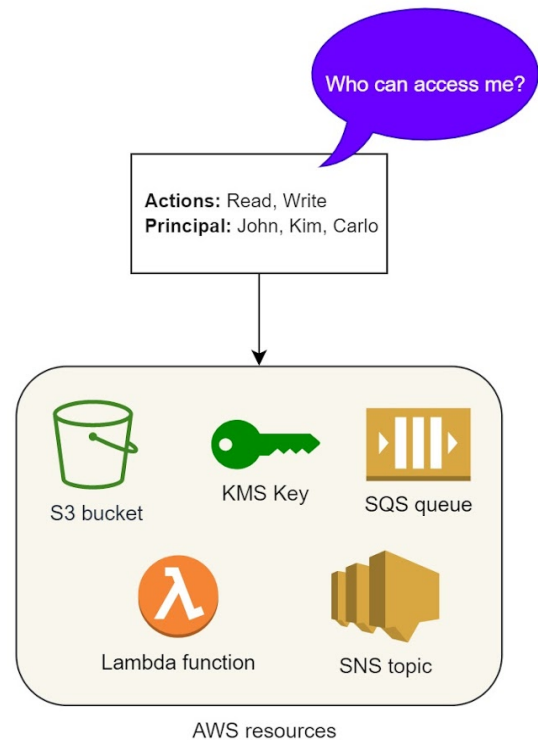
### Identity-based Policy vs. Resource-based Policy

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.

## Identity-based policy



## Resource-based policy



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 **Resource-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.



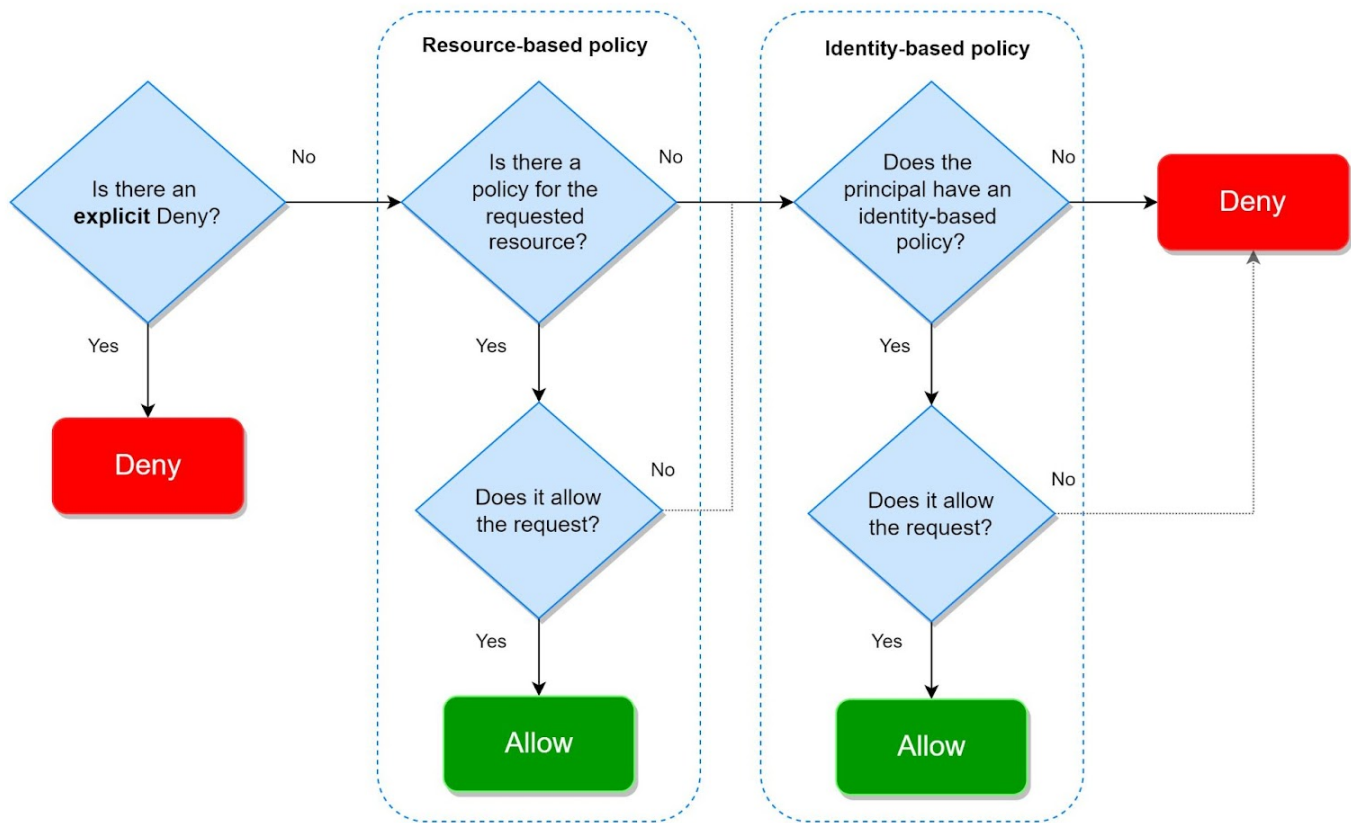


The following S3 bucket policy is an example of how the Resource element is used to grant resource-level permissions. You can see that the tdojo S3 bucket contains two folders, john-folder and dave-folder, to which IAM users John and Dave have read access, respectively.

```
{
  "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/access_policies_identity-vs-resource.html)

[https://docs.aws.amazon.com/IAM/latest/UserGuide/reference\\_policies\\_evaluation-logic.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://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

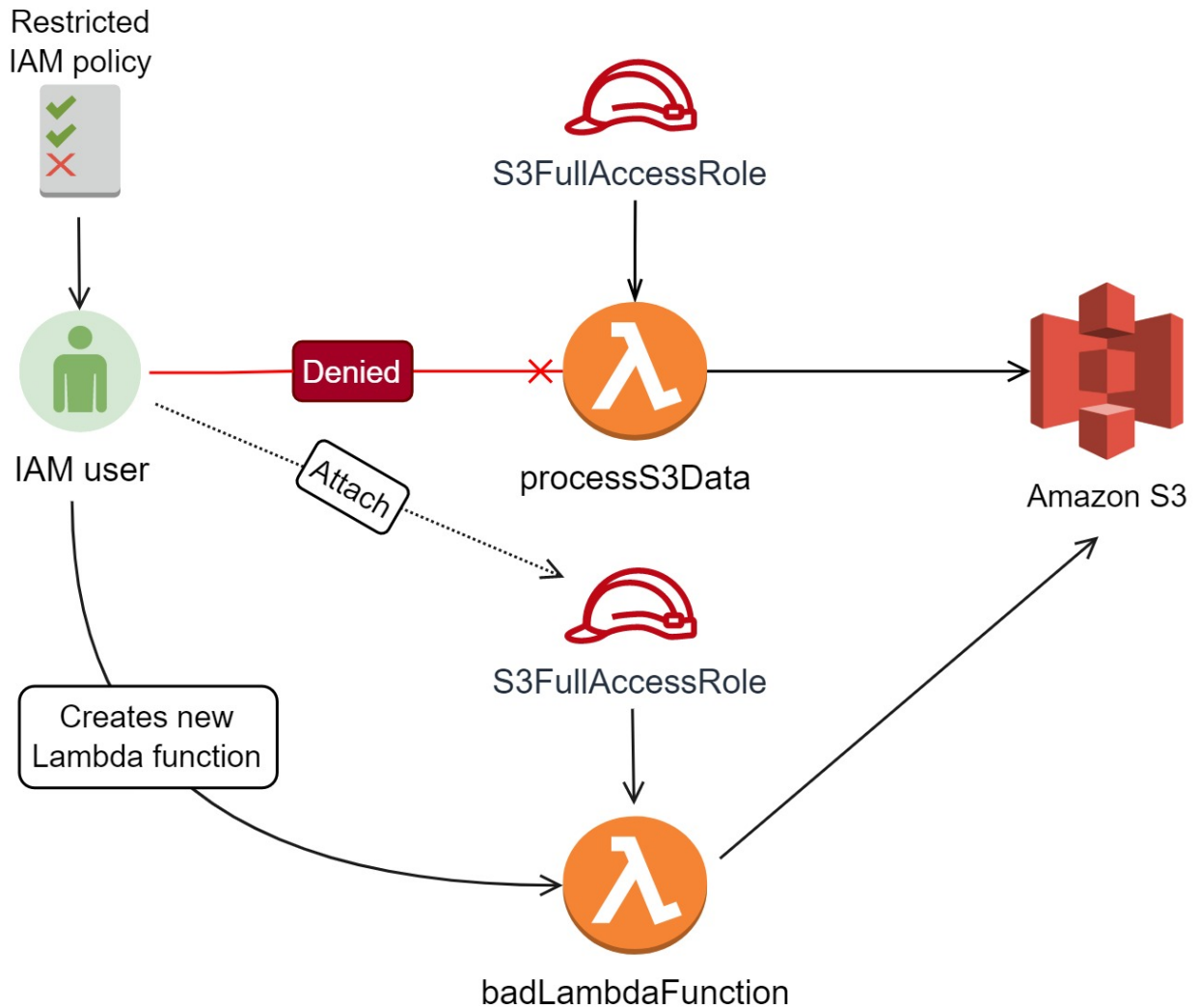
The **iam:PassRole** action grants a user the permission to attach IAM roles to an AWS resource. This is simple yet powerful permission that warrants due scrutiny when constructing IAM Policies. You can, for example, use the PassRole permission to prohibit certain IAM users from passing roles that have 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/id_roles_use_passrole.html)

[https://docs.aws.amazon.com/IAM/latest/UserGuide/reference\\_policies\\_examples\\_iam-passrole-service.html](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_examples_iam-passrole-service.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:**

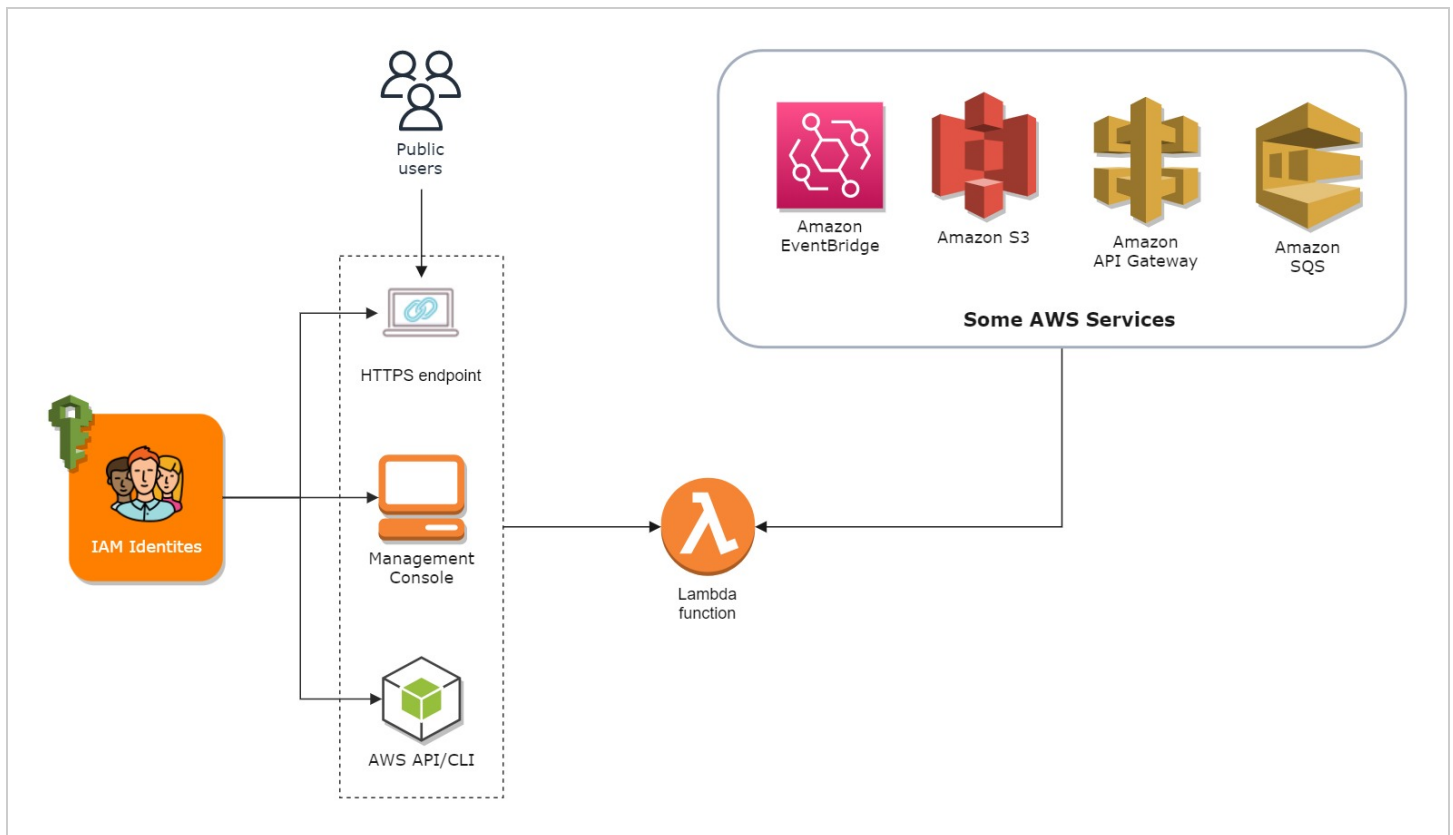
[https://docs.aws.amazon.com/STS/latest/APIReference/API\\_Operations.html](https://docs.aws.amazon.com/STS/latest/APIReference/API_Operations.html)

[https://docs.aws.amazon.com/IAM/latest/UserGuide/id\\_credentials\\_temp\\_request.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.

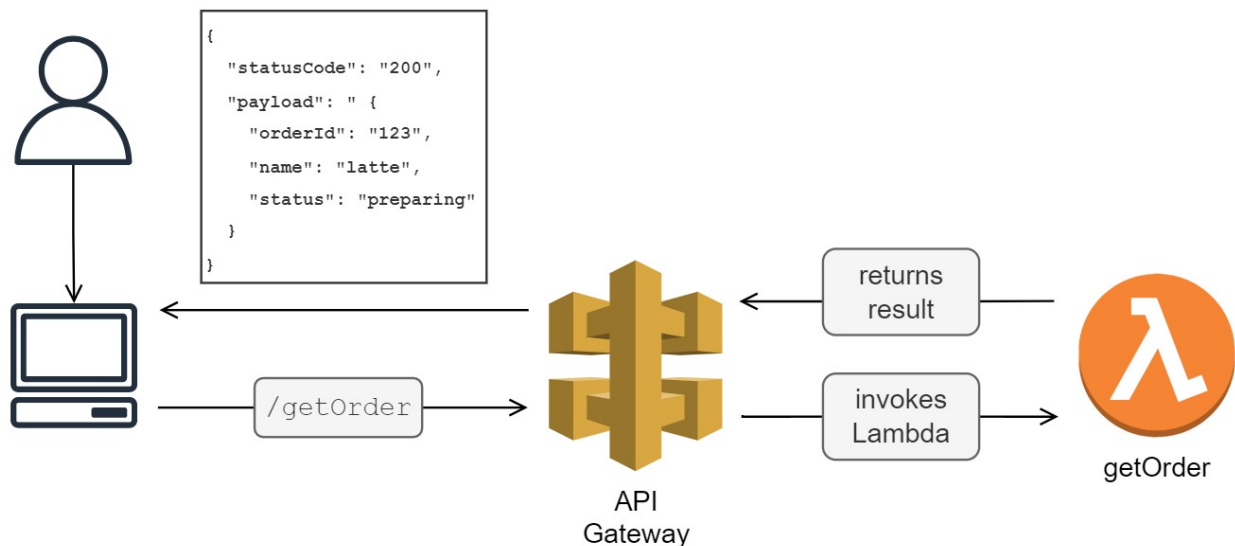


## Synchronous vs. Asynchronous Invocations

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
```

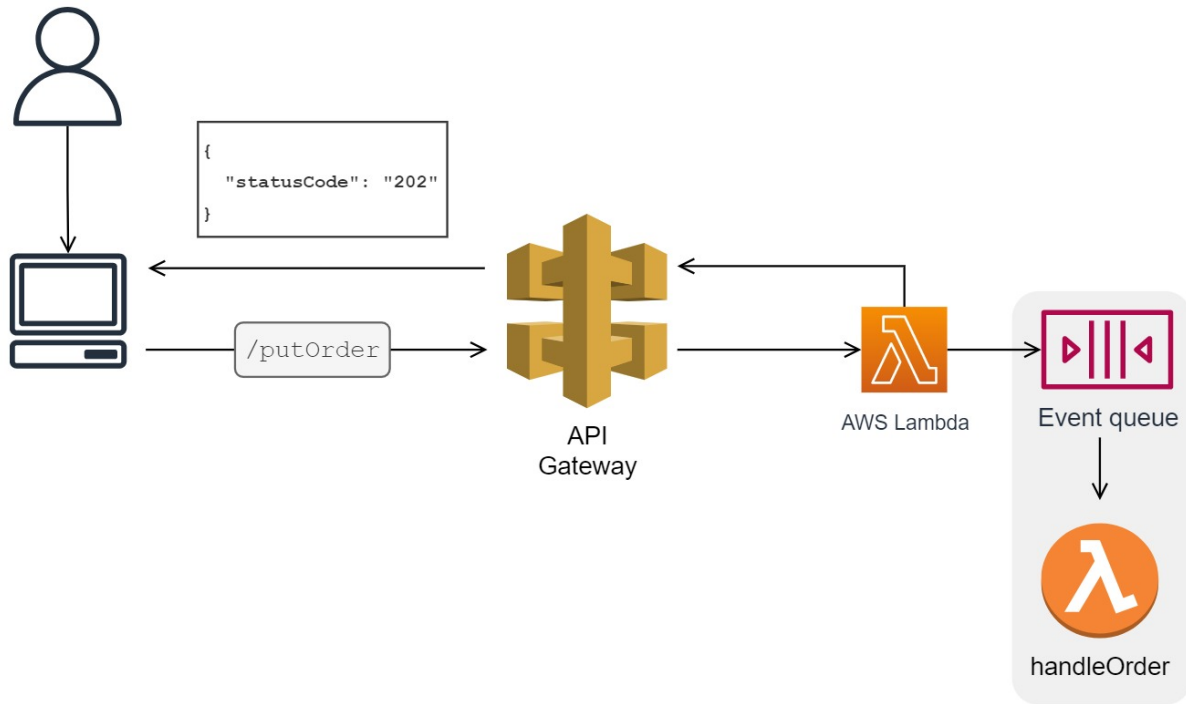
Alternatively, you may just omit the `invocation-type` parameter as AWS Lambda invokes functions synchronously by default.

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

- Amazon API Gateway
- Application Load Balancer
- Amazon Cognito
- Amazon Kinesis 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.



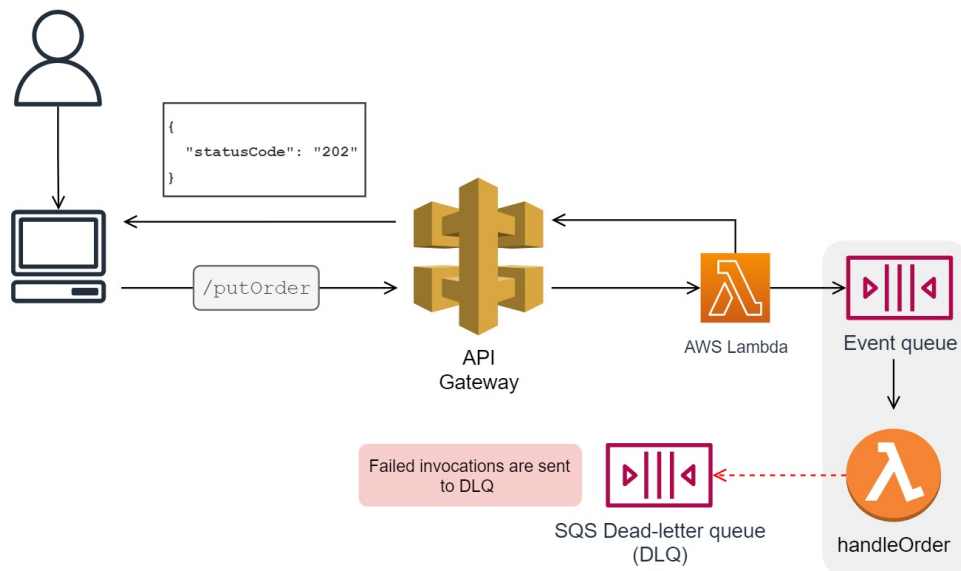
- The Lambda service implements a retry logic for asynchronously-invoked functions
- Good for applications that run in the background.

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 CodeCommit
- 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.

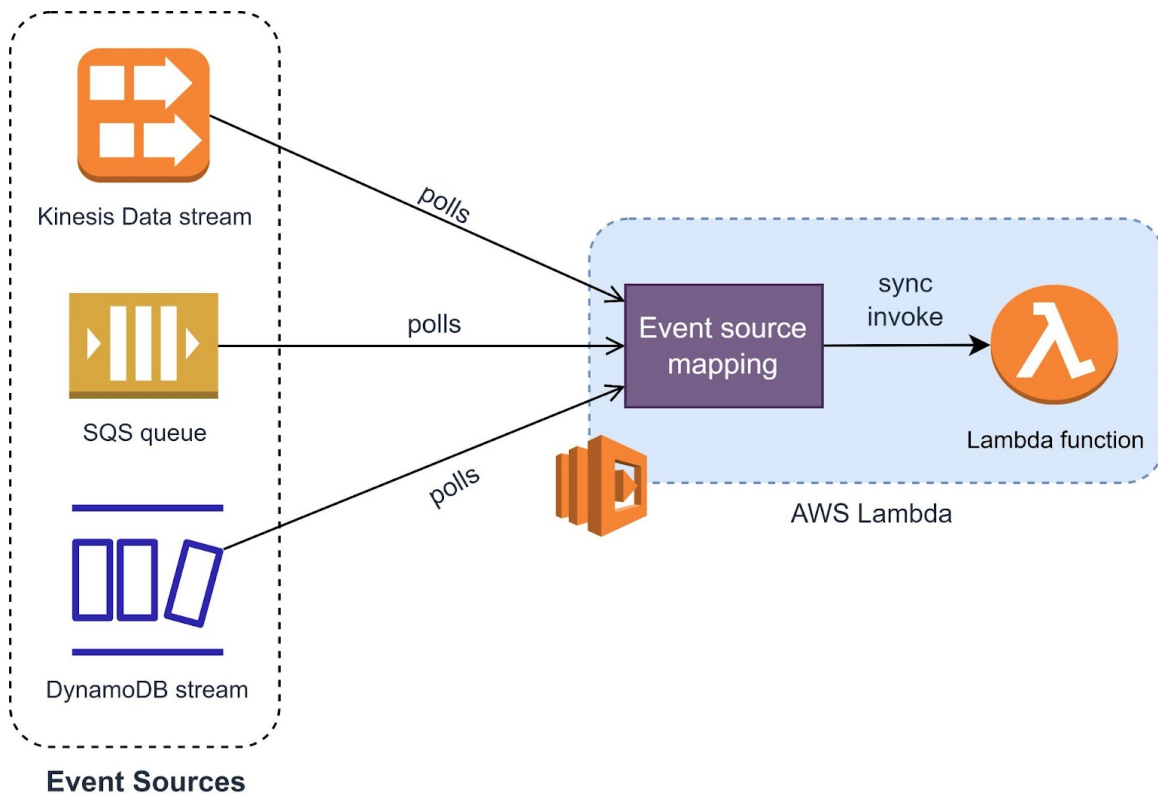
### References:

<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:

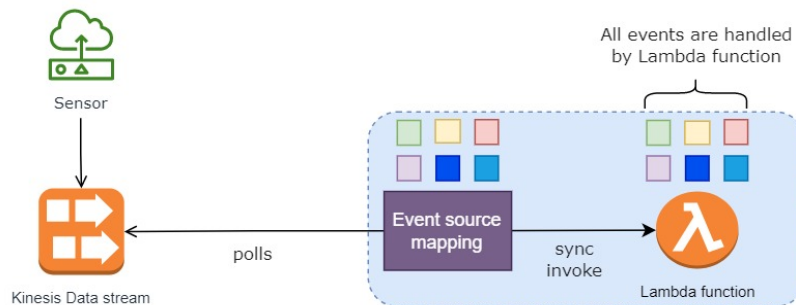
You're tasked to develop a function that reacts to voltage drops or temperature changes that might indicate faulty components in a site. Multiple sensors send readings to a Kinesis Data Stream stream, which must be consumed and processed by a Lambda function.

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

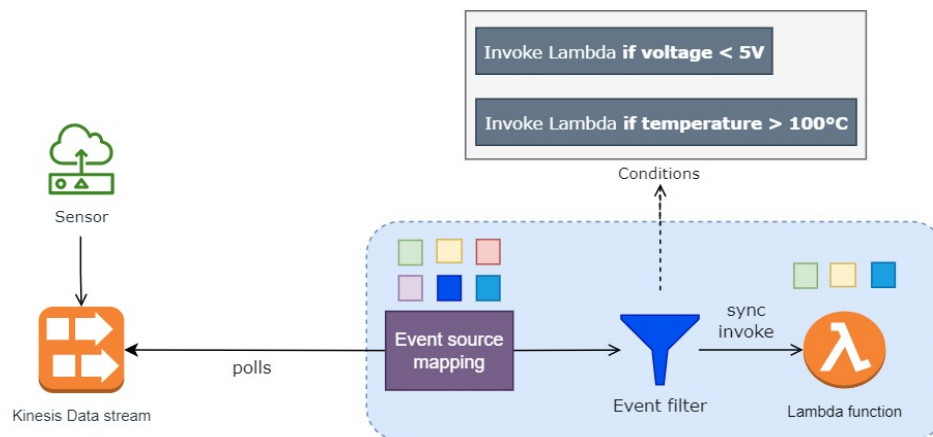
```
def lambda_handler(event, context):
    temperature = event["temperature"]
    voltage = event["voltage"]
    if temperature > 100 or voltage < 5:
        #do something
    else:
        #do nothing
```

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. Suppose sensor data is sent every second, resulting 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.

### Without event filtering



### With event filtering



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.