Intro Lambda, Lambda execution, Lambda Permissions

AWS Lambda

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second. You pay only for the compute time you consume - there is no charge when your code is not running. With AWS Lambda, you can run code for virtually any type of application or backend service - all with little to no administration in regards to environment provisioning and scaling.
Read the Developer Guide for AWS Lambda here:
https://docs.aws.amazon.com/lambda/latest/dg/getting-started.html

The Lambda Execution Environment

When AWS Lambda executes your Lambda function, it provisions and manages the resources needed to run your Lambda function. When you create a Lambda function, you specify configuration information, such as the amount of memory and maximum execution time that you want to allow for your Lambda function.
It takes time to set up an execution context and do the necessary "bootstrapping", which adds some latency each time the Lambda function is invoked. You typically see this latency when a Lambda function is invoked for the first time or after it has been updated because AWS Lambda tries to reuse the execution context for subsequent invocations of the Lambda function.
After a Lambda function is executed, AWS Lambda maintains the execution context for some time in anticipation of another Lambda function invocation. In effect, the service freezes the execution context after a Lambda function completes, and thaws the context for reuse, if AWS Lambda chooses to reuse the context when the Lambda function is invoked again. This execution context reuse approach has the following implications:
Objects declared outside of the function's handler method remain initialized, providing additional optimization when the function is invoked again. For example, if your Lambda function establishes a database connection, instead of reestablishing the connection, the original connection is used in subsequent invocations. We suggest adding logic in your code to check if a connection exists before creating one.
Each execution context provides 512 MB of additional disk space in the /tmp directory. The directory content remains when the execution context is frozen, providing transient cache that can be used for multiple invocations. You can add extra code to check if the cache has the data that you stored. For information on deployment limits, see AWS Lambda limits.
Background processes or callbacks initiated by your Lambda function that did not complete when the function ended resume if AWS Lambda chooses to reuse the execution context. You should make sure any background processes or callbacks in your code are complete before the code exits.
Read more about the Lambda execution context here:
https://docs.aws.amazon.com/lambda/latest/dg/runtimes-context.html

Lambda Permissions

There are two types of permissions to be aware of when working with AWS Lambda. The first type is **Execution Permissions.** A lambda function’s execution permissions are controlled by an IAM Role. When you create a lambda function you associate an IAM Role with the function. This Role will contain policies that either allow or deny specific API calls to be made. In order for the code running in the lambda function to make calls to AWS APIs the execution role must contain the permissions for those API calls. Read more about execution permissions by clicking here: https://docs.aws.amazon.com/lambda/latest/dg/lambda-intro-execution-role.html
The second type of permission to be aware of with AWS Lambda is **resource-based policies.** Resource-based policies are used to control access to invoking and managing a lambda function. Resource-based policies let you grant usage permission to other accounts on a per-resource basis. You also use a resource-based policy to allow an AWS service to invoke your function.
For Lambda functions, you can grant an account permission to invoke or manage a function. You can add multiple statements to grant access to multiple accounts, or let any account invoke your function. To read more about resource-based permissions click here:
https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html

Lambda Execution

When your code executes in Lambda, the service will spin up a micro-virtual machine that will download your code and all the dependencies required and then execute it. The technology used for those micro-VMs is called Firecracker which is an open source virtualization technology. AWS created this technology with Lambda in mind as well as for multi-tenant container which is being used in the AWS Fargate service. By open-sourcing it, we now allow anyone to contribute and expand the capabilities of this technology. It also allows anyone to use this to build their own version of Lambda on-premises if you really wanted to. You can find more information about this technology here:
https://firecracker-microvm.github.io/