**What is AWS Lambda?**

Serverless, event-driven compute service.

Functions that run code without servers.

Automatically Scale

Pay only for the time you use.

Its automatically Scalable.

Lambda runs the code with high availability.

Do not pay idle time!

**Triggered by Events.**

**Put to S3.**

**Updates to DynamoDB table.**

**Call to API endpoint.**

**Serverless Deployment: How It Works and Practical Examples.**

**There are no charges to be paid when you’re running your Lambda.**

**Languages supported by AWS Lambda.?**

**Node Js, Python, Java & More.**

**These programming languages consists of various libraries and more.**

**What are the various ways to access Ec2?**

The Ec2 can be accessed with the help of command line interfaces and web-based interfaces. AWS lambda contains tools that support for power shell in Windows operating system.

**Serverless deployment** is the latest trend in the cloud arena. You have your code available, but its only executed when a request is made for it. It’s all provisioned within milliseconds and discarded afterward. You only pay for what you use. Sounds nice, doesn’t it? It’s also easier to update whenever you must.

**Serverless, the Concept**

Whenever we refer to **serverless** or **serverless** architecture, we are talking about relying on a third-party service. So, your provider oversees managing and assigning computer resources for your code to execute.

In the end, it provides you with a runtime environment for your code execution. This kind of service is more commonly known as function as a service **(FaaS)**. It scales automatically whenever you need more resources. Finally, you only must pay for the resources you use.

**A New Deployment Way**

In the last paragraph, I quickly reviewed what serverless means. Using this kind of architecture, you must handle deployments in a different way.

Most likely, you’ll have a **continuous-integration** and **continuous-delivery** pipeline. When you complete your development process, your delivery pipeline kicks in. So, your process starts by taking down your application containers.

Afterward, it launches new containers with your code’s latest features. If you have any kind of **high availability configuration**, you must make sure all your nodes are replaced. Then you are done.

When using a **serverless architecture**, you only must package your updated code and give it to your provider. By the way, this step completely depends on your **FaaS** provider. And you are done.

Subsequently, your provider oversees making your updated code available for any new requests. Since each function is isolated in its own environment, the update won’t affect any code during execution. Also, old functions will die after completion. The update will seem instantaneous.

**Drawbacks**

Just updating your code looks like a dream, nevertheless, FaaS is not always the right solution.

For example, it’s not the best scenario for executing long-running applications, as it can be more expensive than having a virtual machine or an already provisioned container.

Another drawback is vendor lock in. This happens because every vendor has its own way for deployment, libraries, and software version requirements. If you need to change to a different provider, you may need to change big part of your code. Right now, no vendors are compatible.

Additionally, we have resource provisioning, or cold start. The first time you execute the function, resources to execute the function need to be initialized. Subsequent request will function better because resources are ready, but if the function is not invoked for some time, the resources may be released.

Graphical user interface, application, Teams

Description automatically generated

Security

As in every application, security should be taken into place since starting the project. Some recommendations include giving your functions the least privileges possible, review third party dependencies, encrypt your data when you need to call other **APIs** or functions, use **environment variables** for credentials, and logging.

Specifically for the last point, it’s better to have a place to concentrate all your logs, not just your serverless functions, for further analysis. For this, I suggest you give Scaler a try.

**Setting up the Scenario**

I have already described how the deployment may work but let’s look at a specific example. Serverless is better for small functions, microservices, or tasks that are very quick to execute. In addition, there isn’t support for many languages yet. So, I will use Python as an example with the most common providers: Google Cloud, Amazon Web Services, and Microsoft Azure.

Serverless” means that you don’t need to take care of the servers running your application.

**Implementation of the Amazon Web Services Lambda for my server less computing task that will include** monitoring add **event-based triggers.**

**Lambda is a server less service, run an application or deploy a piece of code, we can write them as a function and integrate with the source services. Lambda is event driven. If any source event happens then we can call or integrate or process the function or the code.**

**AWS Lambda** allows you to run codes without the help of managing. **servers** or **provisioning**.

You must pay for the **computing time** when you **consume data**. There are no charges to be paid when you are not running your code. Using **Lambda**, you can quickly **run codes** for any application or **backend service virtually**, without any administration. You must

upload the code and rest everything is taken care of by **Lambda**. **Lambda** runs and scales your code with **high availability**. You can even set the code up to trigger from the other AWS available or give it a call directly from the mobile app or any web.

**What restrictions apply to AWS Lambda connection?**

**Lambda** imposes very few restrictions on **operating system** activities and standard language. However, there are few of the activities that have been disabled like for **instance**, **inbound network connections** and **trace calls**, which is a debugging system, and

**TCP port 25 traffic as a measure to anti-spam**.

For **outbound connections IP/TCP** sockets are supportive.

**What events can trigger an AWS lambda function?**

Graphical user interface

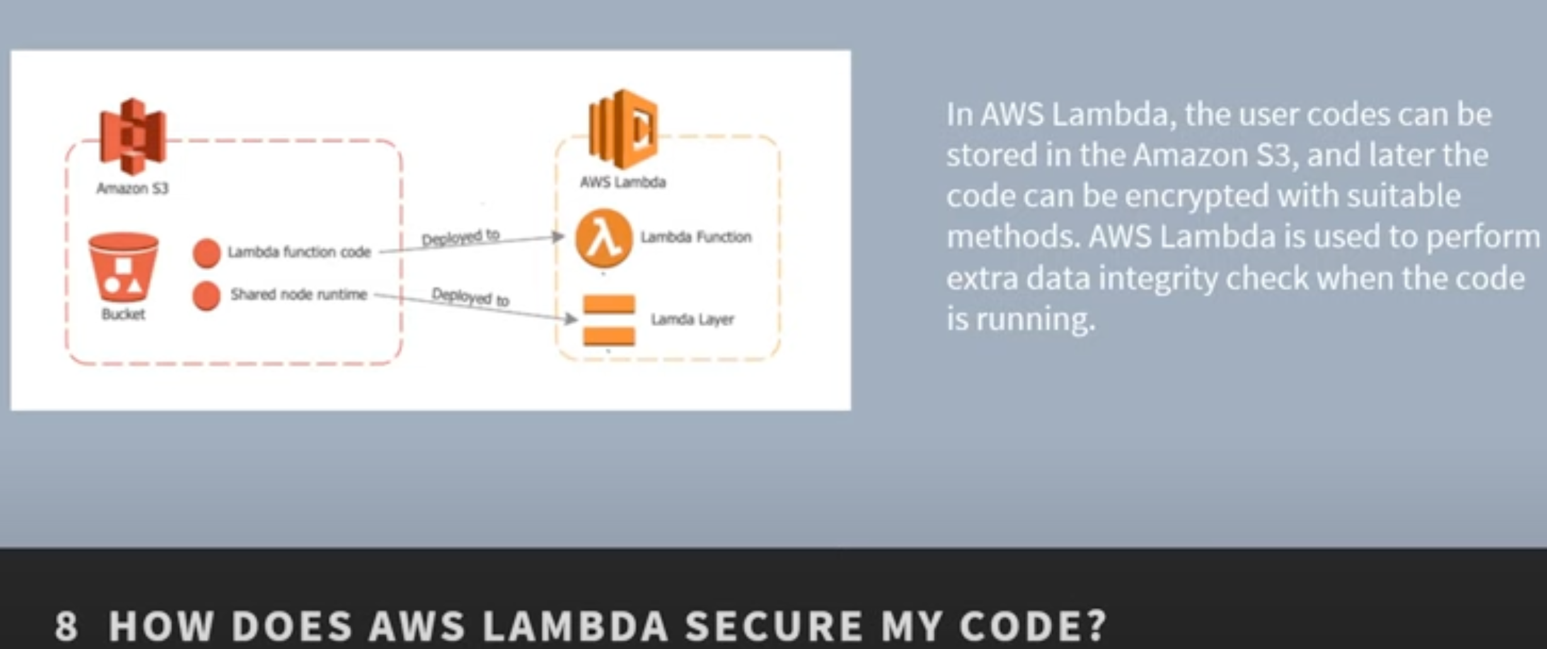
Description automatically generated

**What is a serverless Architecture?**

Serverless architecture is a method where all the server management is done by AWS such that it doesn’t need to manage an infrastructure to build or run an application.

**How long can an AWS lambda function execute?**

The complete execution must take place within 300 seconds from placing the calls to AWS Lambda. 3 seconds is the default timeout however you can set any timeout value between 1 to 300 seconds.



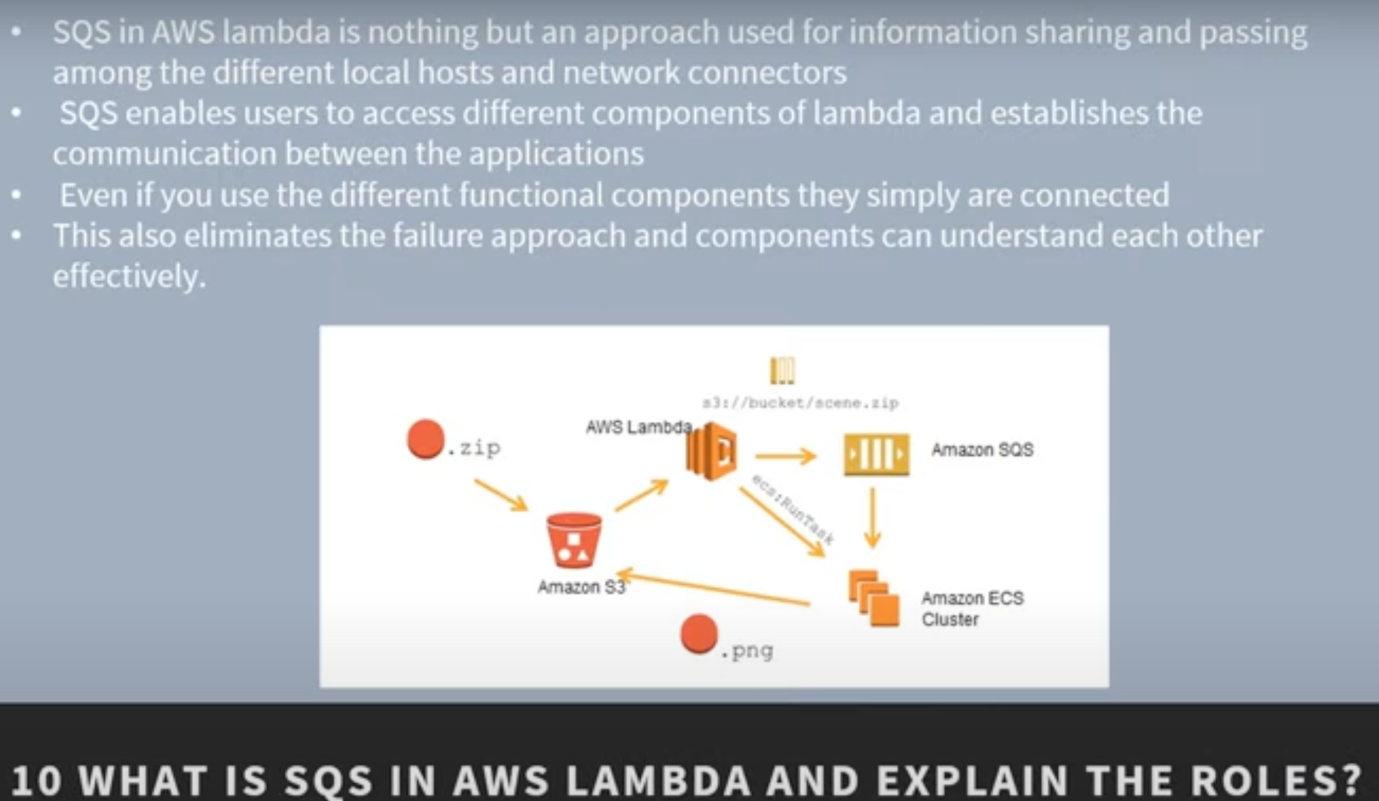
**What is an elastic Blockage in AWS Lambda?**

The **elastic storage** is basically a **virtual storage** are where the user can start working on network related tasks.

This storage can tolerate faults easily and the user no needs to worry about the loss of data even when the disk damages in the RAID.

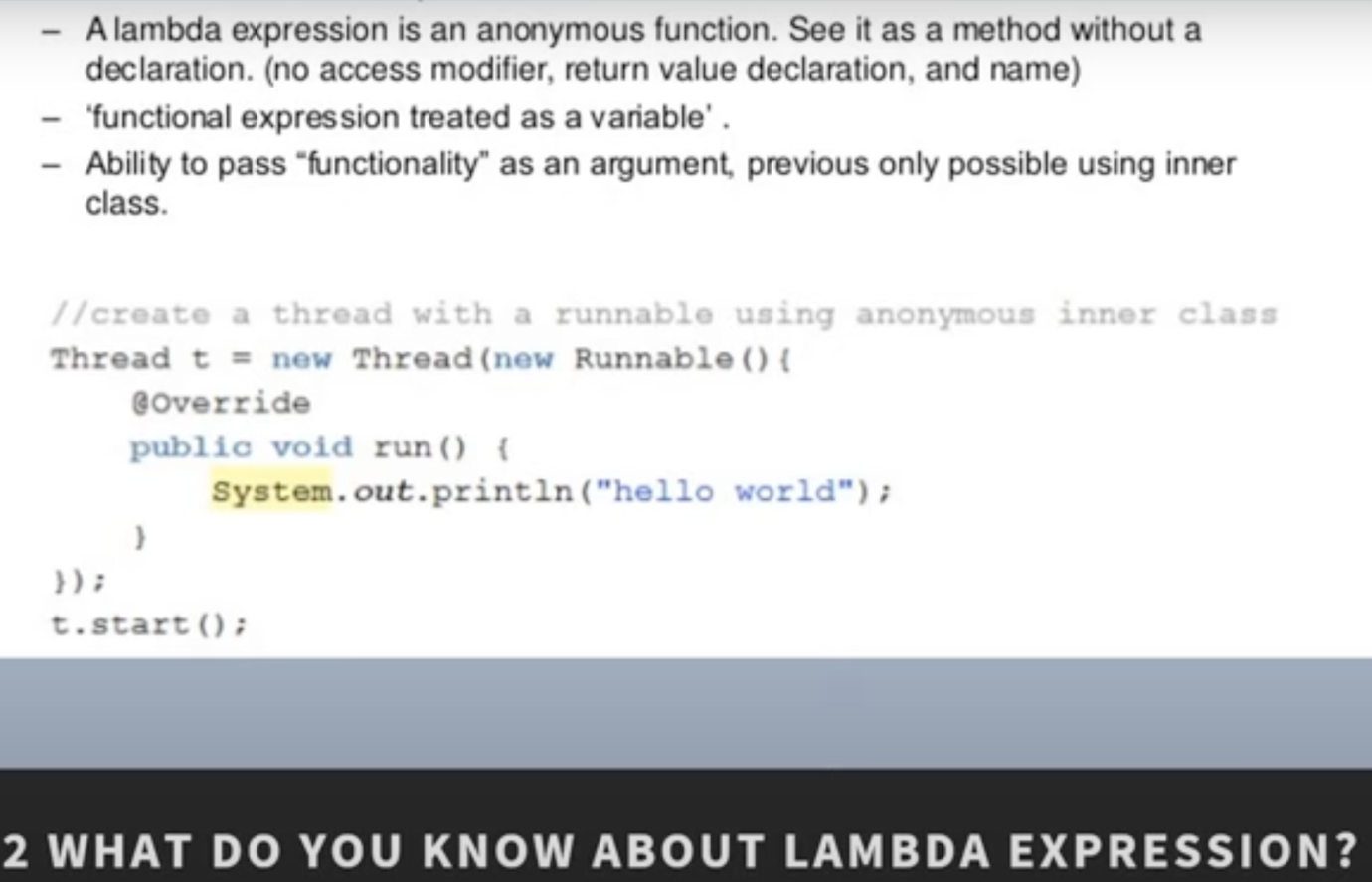
This storage also supports provision and allocating memory storage.

Sometimes on an emergency node, this can also be connected to the API.

****

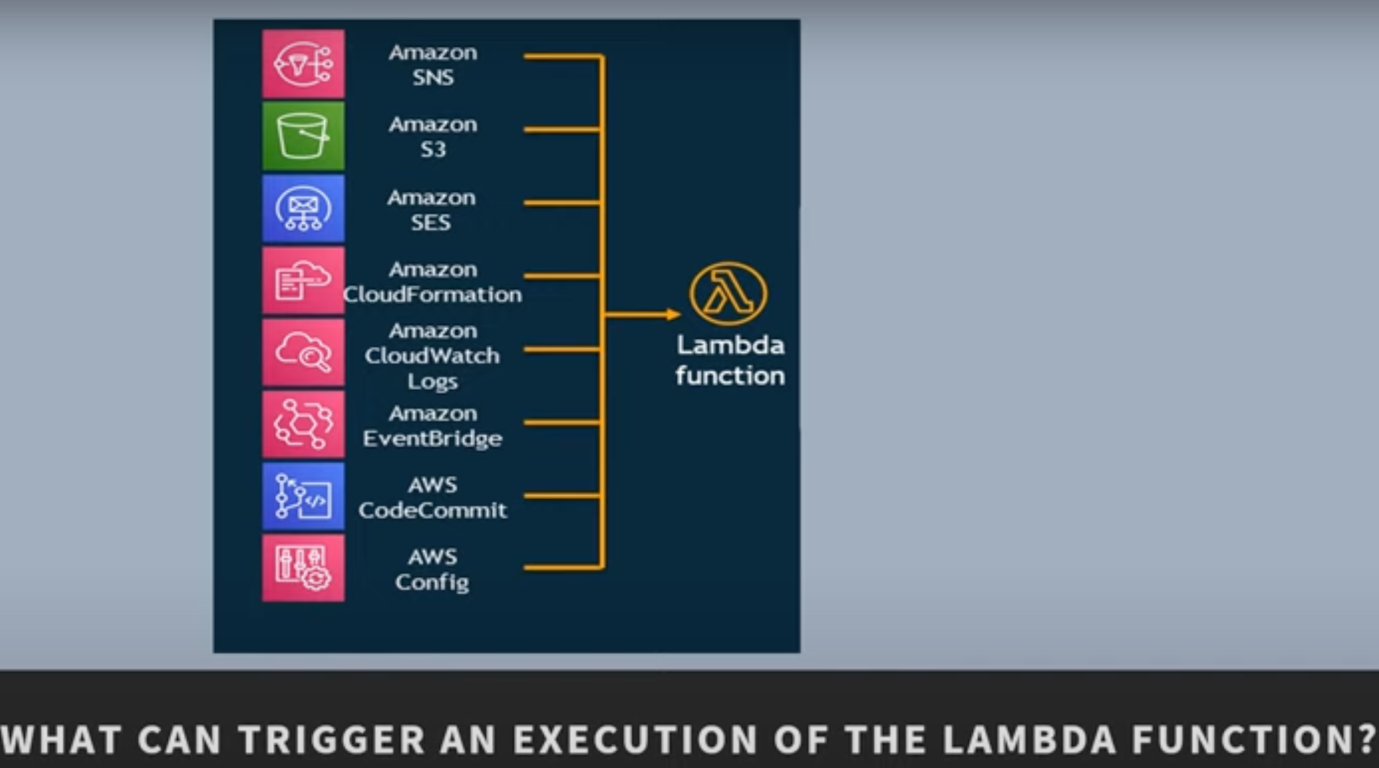
**Is vertical Scaling Possible in Lambda?**

Yes, it is possible and in fact, it is one of the best features in the AWS lambda.

****

**How to run code in Aws without managing servers**

AWS Lambda



**How to run code in Aws**

You upload your code and then specify context information to **AWS Lambda** to create a function.

The context information specifies the execution environment (language, memory requirements, a timeout period, and IAM role) and points to the function you'd like to invoke within your code.

**What is the difference between the Anonymous class and Lambda function..?**

One main difference is the use of keywords. The keywords in Lambda functions are used to resolve to enclose the functional classes.

While with the **anonymous** class the keywords are used to resolve the **anonymous functional** class itself.

**What are the use cases for which Lambda was actually designed?**

The overall response to the clicls made on the website, image uploading, sensors reacting monitoring, as well as reading fron the IOT devices are some of the use cases of AWS lambda.

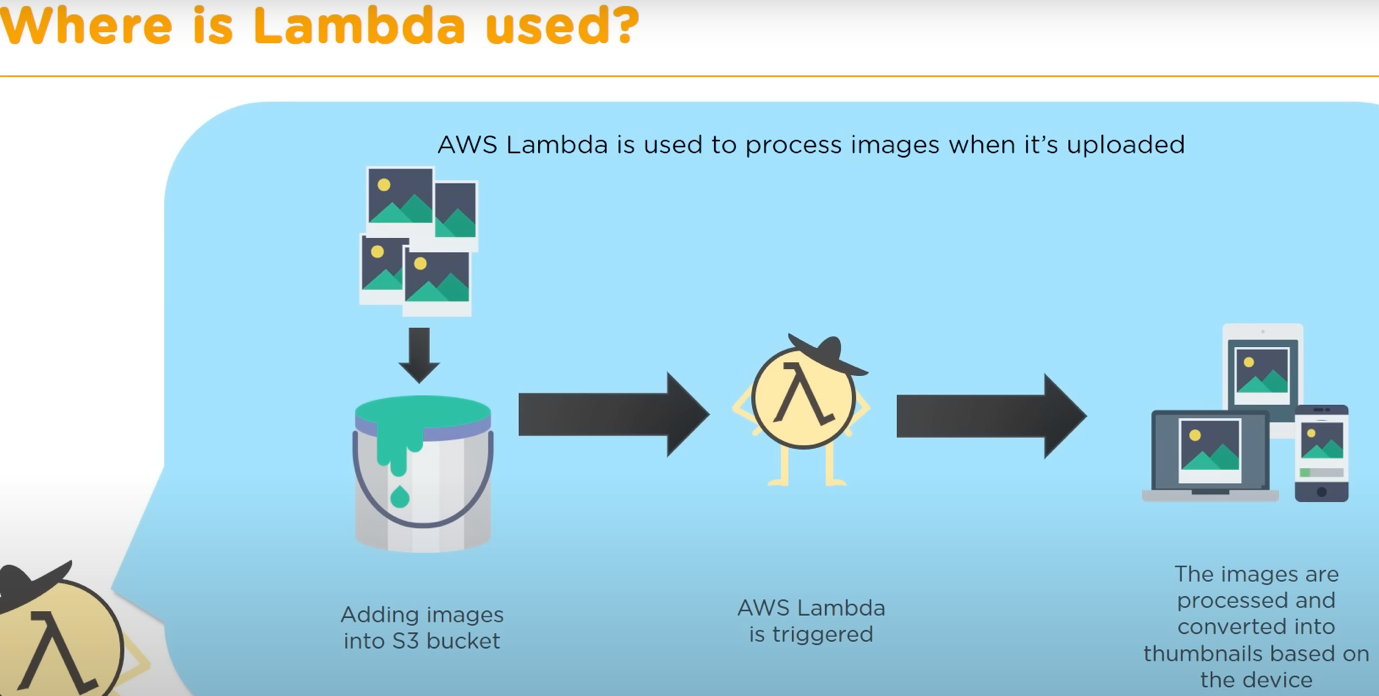
However, access is not just limited to this only.

There are several other tasks that can also be accomplished with Lambda.

Back-end services can be provisioned automatically with Lambda.

Graphical user interface, text, application, chat or text message

Description automatically generated



Graphical user interface

Description automatically generated with low confidence

With an increasing number of requests, an increasing number of containers are created.

When the requests reduce the number of containers reduce as well.

You are only charged for te amount of time that a function is running inside these containers.