😊 The First Function with Serverless Framework

What is Serverless?

Serverless computing is a cloud computing execution model where cloud providers automatically manage the infrastructure for application deployment. With serverless computing, developers focus on writing code and defining functions, without worrying about provisioning, scaling, or managing servers. Despite the name "serverless," servers are still involved, but they are abstracted away from the developer.

In a serverless architecture:

- You **do not manage servers** (no provisioning, scaling, or maintenance).
- The cloud provider automatically **scales** resources based on demand.
- You only pay for **what you use**, with costs based on the number of requests or execution time of your code (e.g., AWS Lambda).

Key Characteristics of Serverless:

- **Event-driven**: Functions are executed in response to events (e.g., HTTP requests, database changes).
- **Automatic Scaling**: The platform automatically handles scaling, so you don't need to worry about overloading or underutilization.
- **Stateless**: Serverless applications are typically stateless, meaning they don't retain information between function invocations, but they can use external services like databases to store data.
- **Pay-per-use**: You only pay for the actual execution time and resources used by the function, not for idle server time.

Serverless and AWS Integration

AWS offers several services that help you build and deploy serverless applications. The most prominent is **AWS Lambda**, but there are several other services that complement serverless applications in AWS.

1. AWS Lambda:

- **Function as a Service (FaaS)**: AWS Lambda allows you to run code without provisioning or managing servers. You upload your code (in various languages like Python, Node.js, Java, etc.), and Lambda runs it in response to events, such as HTTP requests via API Gateway, changes in data in S3, or updates in DynamoDB tables.
- **Event-driven**: Lambda functions can be triggered by AWS services like S3, DynamoDB, Kinesis, SNS, CloudWatch Events, and API Gateway.

- **Automatic Scaling**: Lambda automatically scales based on the number of incoming requests, with no need for manual intervention.

2. Amazon API Gateway:

- **API Management**: API Gateway allows you to create, publish, and manage APIs for serverless applications. You can integrate API Gateway with Lambda to invoke Lambda functions when HTTP requests are made to your API.
- **Fully Managed**: It abstracts the complexity of managing APIs and scales automatically with traffic.
- **Security & Authorization**: It offers features like request validation, throttling, and integration with AWS IAM for security.

3. AWS DynamoDB:

- **Serverless Database**: DynamoDB is a fully managed NoSQL database that can be used with serverless applications. It provides low-latency data storage and scales automatically based on traffic.
- **Integrated with Lambda**: Lambda can trigger actions based on changes in DynamoDB tables (e.g., when a new item is added, updated, or deleted).

4. Amazon S3 (Simple Storage Service):

- **Object Storage**: S3 is often used for storing static files (e.g., images, documents) in serverless architectures. Lambda can be triggered to process data as soon as a new object is uploaded to S3.

5. AWS Step Functions:

- **Serverless Orchestration**: AWS Step Functions lets you coordinate multiple Lambda functions and other AWS services into serverless workflows. This is useful for building complex workflows or microservices architectures.

6. AWS EventBridge:

- **Event Bus Service**: EventBridge is a serverless event bus that makes it easier to connect different AWS services and your own applications. It can trigger Lambda functions or forward events to other services when certain conditions are met.

7. AWS SNS (Simple Notification Service) and SQS (Simple Queue Service):

- **Asynchronous Messaging**: SNS is used for sending notifications to other systems or users, while SQS is used for message queuing. Both can be integrated with Lambda to trigger functions based on messages or notifications.

Benefits of Serverless with AWS:

- **Cost-efficient**: With a pay-per-use model, you are billed only for the time your code runs, making serverless very cost-effective.
- **No Infrastructure Management**: AWS automatically takes care of infrastructure management, scaling, patching, and server provisioning.

- **Focus on Code**: Developers can focus solely on writing the application logic instead of managing the environment.
- **Quick Scaling**: Serverless functions scale automatically depending on the number of requests, without requiring manual intervention.
- **High Availability**: AWS ensures high availability by automatically managing availability and fault tolerance.

Example Use Cases:

- **Real-time file processing**: Trigger a Lambda function when a file is uploaded to S3.
- **Web application backends**: Use API Gateway with Lambda to serve as a backend for a web application.
- **Data processing pipelines**: Integrate Lambda with DynamoDB or Kinesis to process streams of data in real-time.
- **Event-driven workflows**: Use EventBridge or SNS to trigger workflows based on system events.

Conclusion:

Serverless computing simplifies application deployment by abstracting infrastructure management. AWS offers a range of services, like Lambda, API Gateway, and DynamoDB, to help you build scalable, cost-efficient, and highly available serverless applications without the need to manage servers.

What is Serverless Framework?

The **Serverless Framework** is an open-source tool designed to simplify the process of building, deploying, and managing serverless applications. It provides a set of abstractions to help developers define and manage the architecture of serverless applications, especially those that are built on cloud platforms like AWS, Azure, or Google Cloud.

The Serverless Framework allows developers to define the entire serverless architecture using simple configuration files and then deploy it with a single command. It handles the deployment of serverless functions (like AWS Lambda), APIs (using API Gateway), and other cloud services that your application depends on, with minimal manual configuration.

Key Features of Serverless Framework:

1. **Declarative Configuration**: You define your application's architecture in a simple configuration file (usually called serverless.yml), where you specify your functions, events, resources, and more.
2. **Multi-cloud Support**: While most commonly used with AWS, the Serverless Framework also supports other cloud providers like Azure, Google Cloud, and more.
3. **Automatic Deployment**: The framework automatically handles the deployment of your Lambda functions, APIs, and other resources like DynamoDB tables, S3 buckets, etc.

4. **Simplified Monitoring & Debugging**: The framework integrates with tools like AWS CloudWatch, allowing you to monitor, log, and debug your functions more easily.
5. **Plugins and Extensions**: The Serverless Framework has a vibrant ecosystem of plugins, which can add additional functionality for deployment, testing, monitoring, security, and more.
6. **CI/CD Integration**: It integrates well with continuous integration and continuous deployment (CI/CD) systems to automate testing and deployment processes.

How Serverless Framework Integrates with AWS:

The Serverless Framework is primarily used with AWS to build and manage serverless applications, especially those that use AWS Lambda, API Gateway, DynamoDB, S3, and other AWS services. Here's how it integrates with AWS:

1. AWS Lambda:

- **Function Deployment**: The Serverless Framework deploys AWS Lambda functions defined in your serverless.yml file. You specify the code for your Lambda functions, and the framework packages and deploys them.
- **Event Triggers**: It automatically configures event triggers for your Lambda functions (e.g., API Gateway for HTTP requests, S3 for file uploads, DynamoDB for table changes, etc.).

2. Amazon API Gateway:

- **API Deployment**: You can easily define and deploy RESTful APIs using API Gateway in the serverless.yml configuration. The Serverless Framework automates the creation of the necessary API Gateway endpoints that map to your Lambda functions.
- **Security & Authorization**: The framework also allows you to configure API Gateway's security settings, such as using IAM roles, AWS Cognito for authentication, or API keys for access control.

3. Amazon DynamoDB:

- **Resource Definition**: You can define DynamoDB tables in the serverless.yml file, which the Serverless Framework will automatically create and configure as part of the deployment process.
- **Event Triggers**: You can also set up DynamoDB triggers to invoke Lambda functions when specific changes happen in the table.

4. Amazon S3:

- **Event-Driven Lambda Triggers**: The Serverless Framework can automatically configure AWS S3 buckets to trigger Lambda functions when objects are uploaded or modified, allowing you to create event-driven file processing applications.

5. AWS CloudFormation:

- **Infrastructure as Code**: The Serverless Framework uses **AWS CloudFormation** under the hood to create and manage resources. When you deploy your serverless application, CloudFormation handles the provisioning of AWS resources based on the configuration in serverless.yml.
- **Stack Management**: CloudFormation stacks help in managing and deploying all resources as a single unit.

6. IAM Roles and Permissions:

- **Automatic Role Creation**: The Serverless Framework automatically generates the appropriate **IAM roles** for Lambda functions, giving them the permissions they need to interact with other AWS services.
- **Fine-Grained Access Control**: You can specify more granular IAM policies and permissions in the serverless.yml configuration, ensuring that your Lambda functions have the least privileges required for execution.

Benefits of Using Serverless Framework with AWS:

- **Simplicity**: It simplifies the process of deploying and managing serverless applications on AWS, reducing the need for manual configuration.
- **Faster Development**: By abstracting infrastructure management, developers can focus more on writing code and building features.
- **Automated Resource Management**: It automatically handles the creation and management of AWS resources like Lambda, API Gateway, DynamoDB, etc.
- **Scalability and Flexibility**: The Serverless Framework integrates seamlessly with AWS, which provides automatic scaling and flexible resource allocation based on demand.
- **Multi-cloud Support**: While AWS is the most common use case, the Serverless Framework also supports other cloud providers like Azure and Google Cloud, making it easier to migrate or develop multi-cloud applications.

Conclusion:

The **Serverless Framework** is a powerful tool that simplifies building, deploying, and managing serverless applications on AWS. It abstracts much of the complexity associated with configuring AWS services and helps developers focus on writing code. By leveraging AWS services like Lambda, API Gateway, and DynamoDB, it makes it easier to deploy scalable, event-driven serverless applications.

Lab Summary and End Goal

In this lab, we explored how to use the **Serverless Framework** to create, deploy, manage, and test an AWS Lambda function. Here's a quick breakdown:

1. **Setup and Configuration**: Installed AWS CLI, Node.js, and the Serverless Framework. Configured AWS CLI with necessary credentials to interact with AWS services.

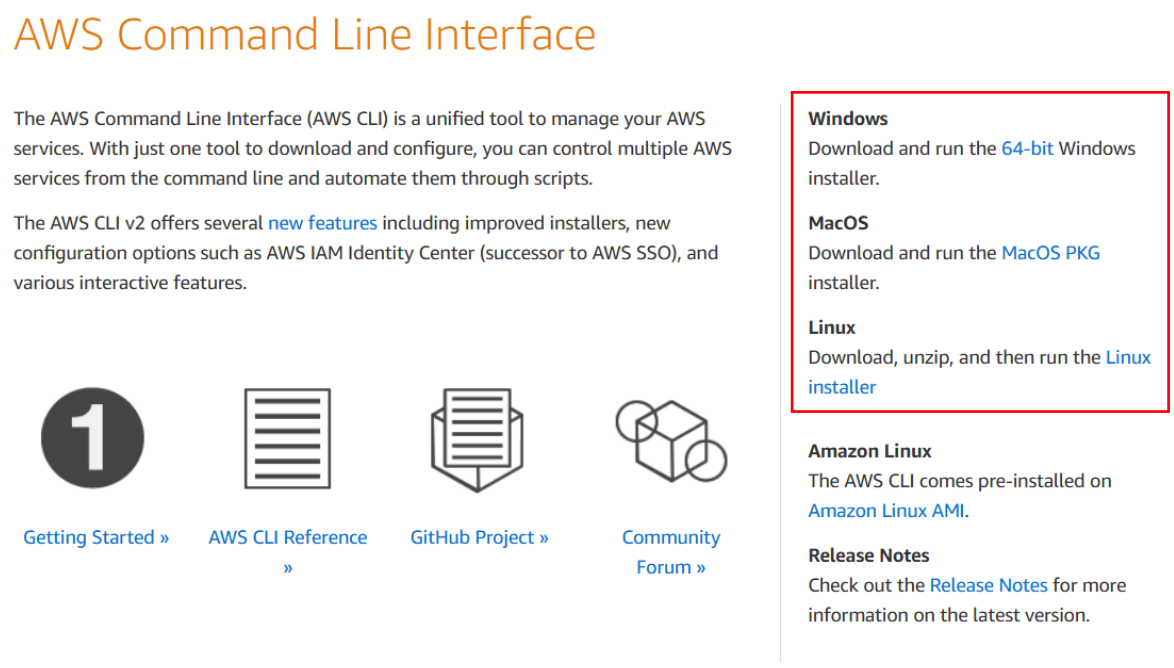
2. **Project Initialization and Deployment**: Created a Serverless Framework project with a simple Lambda function template. We deployed it, which automatically set up an AWS CloudFormation stack and created resources like the Lambda function.
3. **Testing and Modifying**: Tested the Lambda function and observed logs and metrics via CloudWatch. We made code changes and explored two deployment methods: full-stack redeployment and single-function deployment.
4. **CLI Commands for Invocation and Logging**: Used Serverless CLI commands to invoke the function and view logs directly from the command line.
5. **Cleanup**: Removed all resources created during the lab.

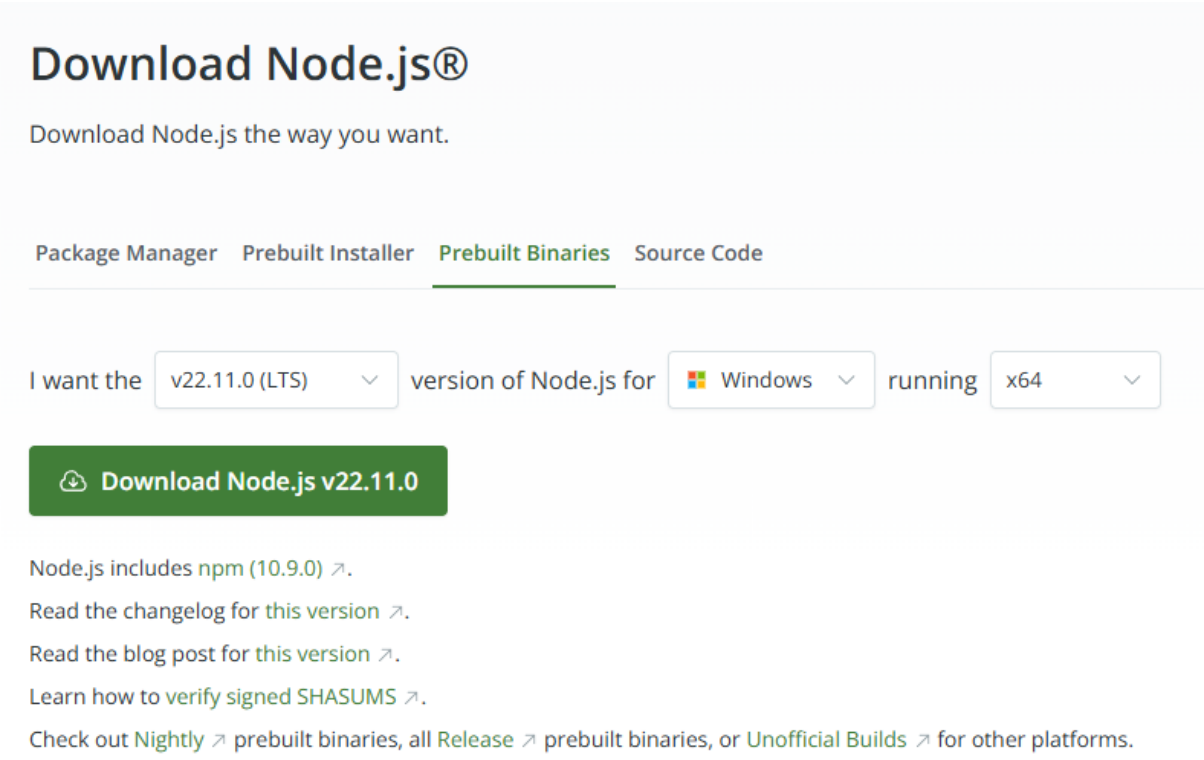
End Goal

The end goal was to gain practical experience using the Serverless Framework for deploying and managing serverless applications on AWS, highlighting how it simplifies setup, testing, and version control for Lambda functions and associated cloud resources.

😄 To begin with the Lab:

1. In this lab we will work with Serverless Framework to deploy our lambda function. Serverless Framework aims to ease the pain of **creating, deploying, managing, and debugging** lambda functions. It integrates well with CI/CD tools.
2. It has CloudFormation support so your entire stack can be deployed using this framework.
3. First, we need to install the Serverless framework on our local machine. But before that, we need to install the dependencies such as Node and AWS CLI.
4. So, on a browser of your choice search for AWS CLI and install it. Then search for node and download the LTS version of Node.js.

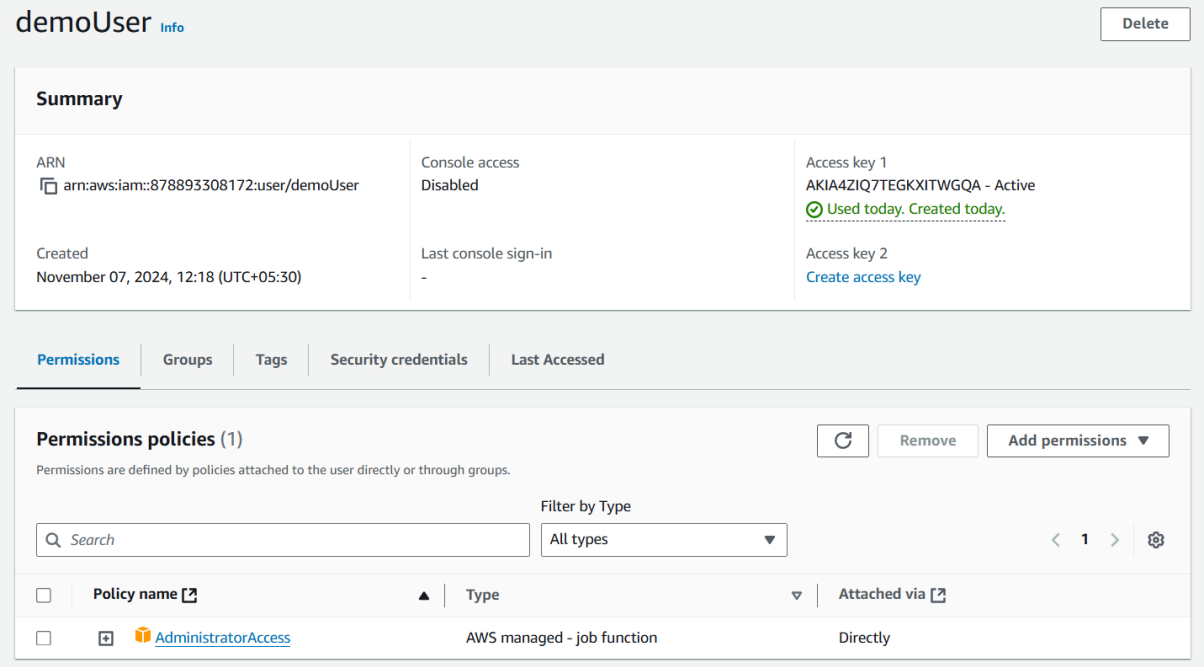




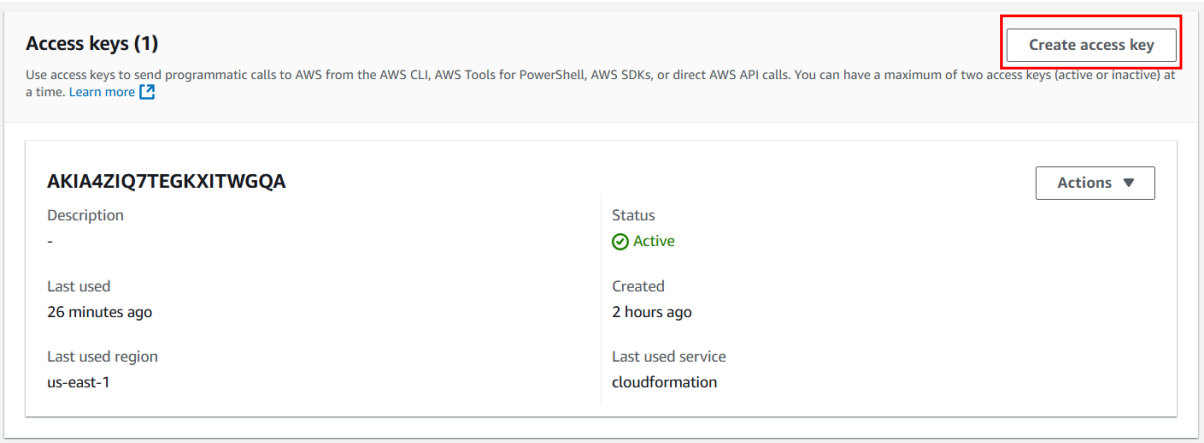
5. To check whether both of them are installed correctly, you can run these commands to check their versions.

```
C:\>aws --version
aws-cli/2.17.34 Python/3.11.9 Windows/10 exe/AMD64

C:\>node --version
v22.11.0
```

6. Once you have installed both of them then you need to go to your AWS Console and navigate to IAM. Create a user with administrative privileges.



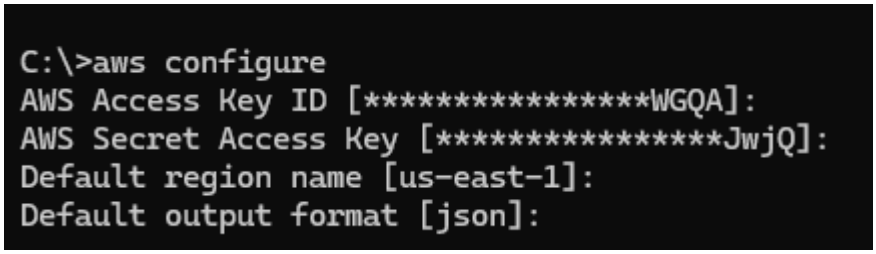
7. Now go to the Security credentials of your user and create an access and secret access key for your user.



8. Once they are created download them, then you need to come back and open the Command prompt in your laptop and run this command.

Aws configure

9. Then it will ask you to write the access and secret access keys copy and paste them here. After that, it will ask you to give a region name where the resources will be created and the default output format will be JSON.



10. Now we are going to install Serverless Framework. Open your command prompt if you closed it. Then write the same command given below.

    npm install -g serverless

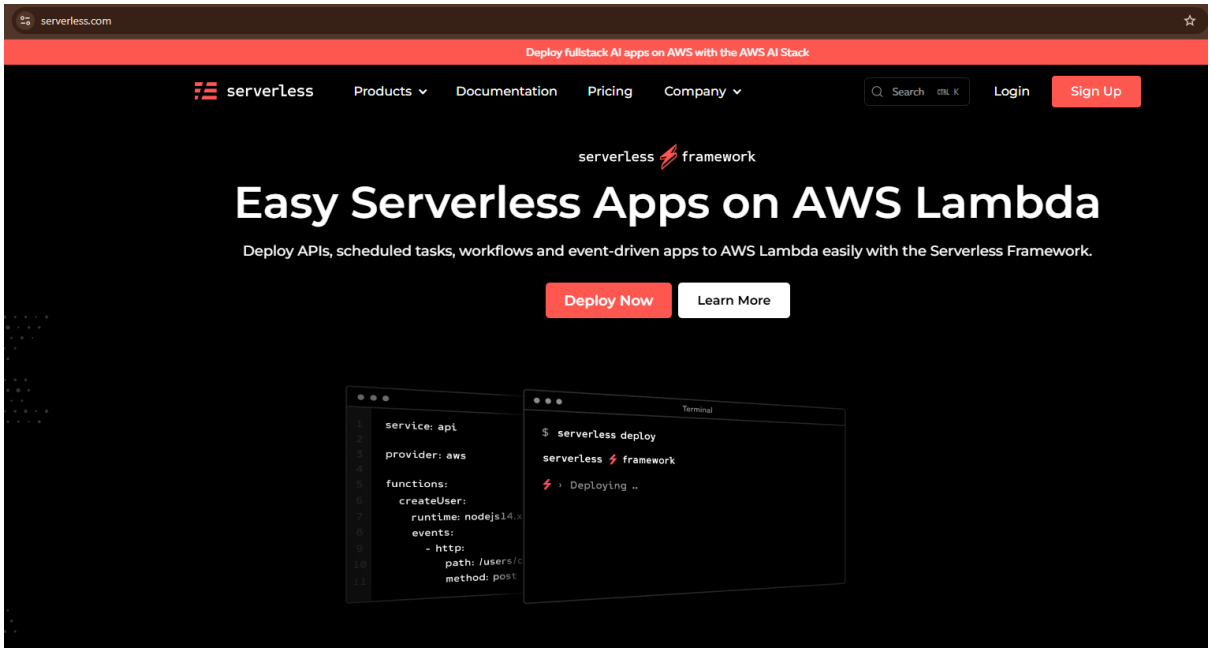
11. I had already installed it so it just changed the packages for me but for you will it will do a clean installation.

```
C:\>npm install -g serverless

changed 57 packages in 4s

15 packages are looking for funding
  run `npm fund` for details
```

12. After the installation just type **serverless** in the command prompt and it will give you this type of information as shown below. This means that we have successfully installed serverless on our laptop.

```
C:\>serverless

(node:8548) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
Serverless ϟ Framework

Welcome to Serverless Framework V.4

Create a new project by selecting a Template to generate scaffolding for a specific use-case.

? Select A Template: ...
> AWS / Node.js / HTTP API
  AWS / Node.js / Express API
  AWS / Node.js / Express API with DynamoDB
  AWS / Node.js / Scheduled Task
  AWS / Node.js / Simple Function
  AWS / Python / HTTP API
  AWS / Python / Flask API
  AWS / Python / Flask API with DynamoDB
  AWS / Python / Scheduled Task
  AWS / Python / Simple Function
  AWS / Compose / Serverless + Cloudformation + SAM
```

13. This is the official website for Serverless framework, you need to visit this website and create your free account here. Only after creating your account, you will be able to work with it. **So, it is an important step.**



14. So, we have completed our setup for Serverless Framework now we will create our first function using it.
15. Now what you can do is create an empty folder and navigate to it using the Command Prompt as you can see below.

```
C:\Serverless>dir
 Volume in drive C is Windows-SSD
 Volume Serial Number is 4ED4-19FB

 Directory of C:\Serverless

07-11-2024  15:01    <DIR>          .
               0 File(s)              0 bytes
               1 Dir(s)  104,792,862,720 bytes free

C:\Serverless>
```

16. Run the Serverless command. Use the arrow keys to come to the Simple function template using Node.js and click on Enter key on your keyboard.

```
C:\Serverless>Serverless

(node:10640) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
Serverless ⅂ Framework

Welcome to Serverless Framework V.4

Create a new project by selecting a Template to generate scaffolding for a specific use-case.

? Select A Template: ...
  AWS / Node.js / HTTP API
  AWS / Node.js / Express API
  AWS / Node.js / Express API with DynamoDB
  AWS / Node.js / Scheduled Task
> AWS / Node.js / Simple Function
  AWS / Python / HTTP API
  AWS / Python / Flask API
  AWS / Python / Flask API with DynamoDB
  AWS / Python / Scheduled Task
  AWS / Python / Simple Function
  AWS / Compose / Serverless + Cloudformation + SAM
```

17. Then it asked me to give my project a name, it also downloaded a template and now it is asking me to create a new APP. So, select the option to create a new APP and click on Enter key.

```
Welcome to Serverless Framework V.4

Create a new project by selecting a Template to generate scaffolding for a specific use-case.

√ Select A Template: · AWS / Node.js / Simple Function

√ Name Your Project: · serverlessdemo1

✔Template Downloaded

This Template contains a Serverless Framework Service. Services are stacks of AWS resources, and can contain your entire application or a part of
 it (e.g. users, comments, checkout, etc.). Enter a name using lowercase letters, numbers and hyphens only.

While the CLI is free for developers and organizations making less than $2 million annually, Serverless Framework's Dashboard offers additional f
eatures such as Observability, State, Secrets & AWS Access sharing, which can incur a cost. To enable these additional features, add your Service
 to an "App". Apps also group your Services together in the Dashboard, for better organization.

Create or select an existing App below to associate with your Service, or skip.

? Create Or Select An Existing App: ...
> Create A New App
  demoapp1
  Skip Adding An App
```

18. When we choose to create a new App it asked us to name it and it create a new App for us.

```
Create or select an existing App below to associate with your Service, or skip.

√ Create Or Select An Existing App: · Create A New App

√ Name Your New App: · serverlessapp1

Your new Service "serverlessdemo1" is ready. Here are next steps:

• Open Service Directory: cd serverlessdemo1
• Install Dependencies: npm install (or use another package manager)
• Deploy Your Service: serverless deploy
```

19. Now on your laptop if you navigate to this location a folder has been created with the name of your project.

Name	Date modified	Type	Size
.serverless	07-11-2024 15:06	File folder	
serverlessdemo1	07-11-2024 15:04	File folder	

20. Using the command prompt go inside of your project folder.

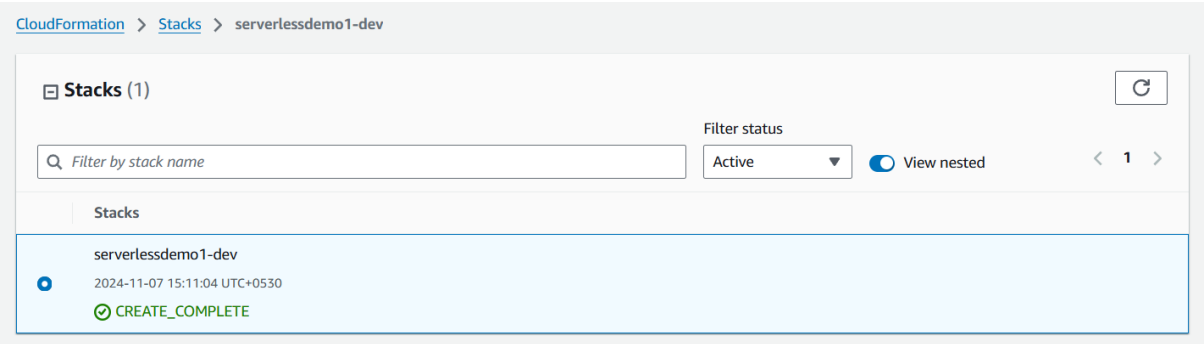
```
C:\Serverless>cd serverlessdemo1

C:\Serverless\serverlessdemo1>dir
 Volume in drive C is Windows-SSD
 Volume Serial Number is 4ED4-19FB

 Directory of C:\Serverless\serverlessdemo1

07-11-2024  15:04    <DIR>          .
07-11-2024  15:06    <DIR>          ..
18-09-2024  10:37                24 .gitignore
18-09-2024  10:37               183 handler.js
18-09-2024  10:37             2,360 README.md
07-11-2024  15:06               429 serverless.yml
               4 File(s)          2,996 bytes
               2 Dir(s)  104,792,530,944 bytes free

C:\Serverless\serverlessdemo1>
```

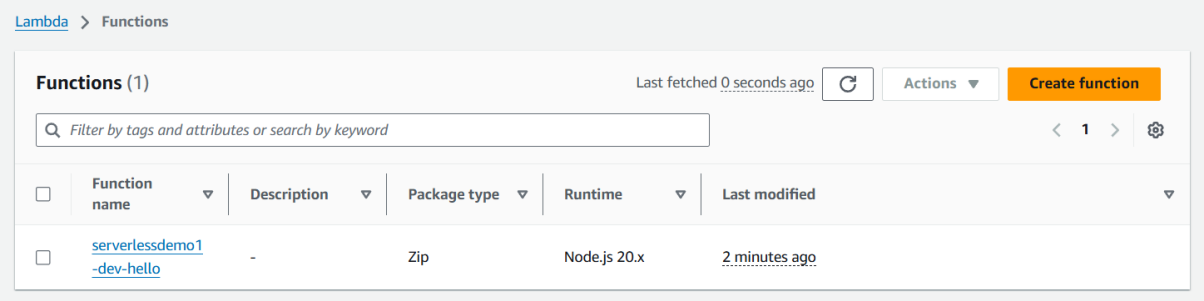
21. Now you need to run this command to deploy your Serverless App. Here you can see that it has deployed a Cloud formation template and created a lambda function.

    serverless deploy

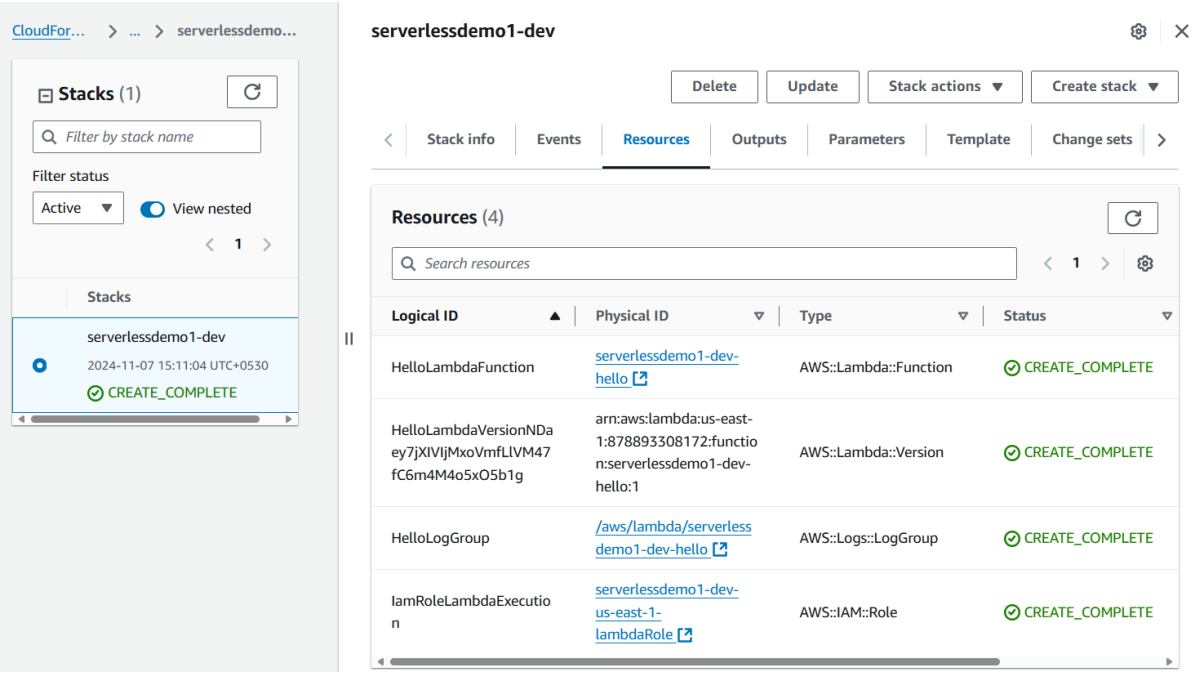
```
C:\Serverless\serverlessdemo1>serverless deploy

(node:24672) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
Deploying "serverlessdemo1" to stage "dev" (us-east-1)

✔Service deployed to stack serverlessdemo1-dev (60s)

functions:
  hello: serverlessdemo1-dev-hello (1.7 kB)
```

22. Now in your AWS Console navigate to Cloud formation and Lambda there you will see your stack and the function.

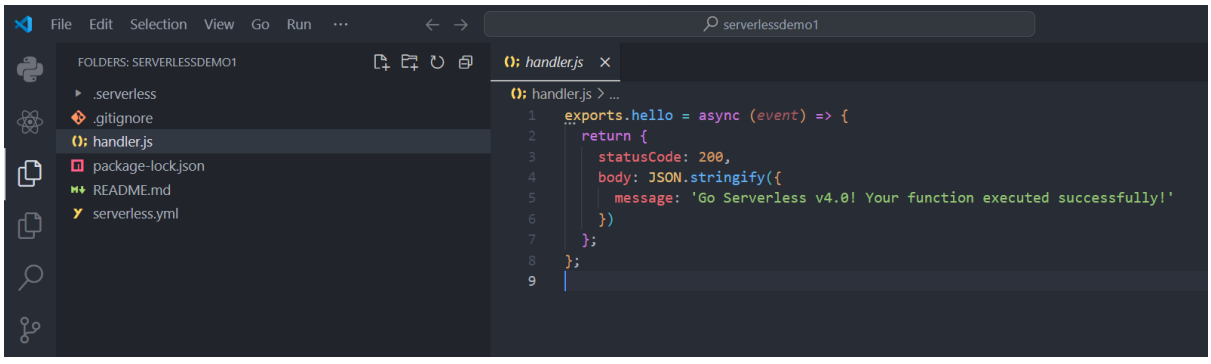




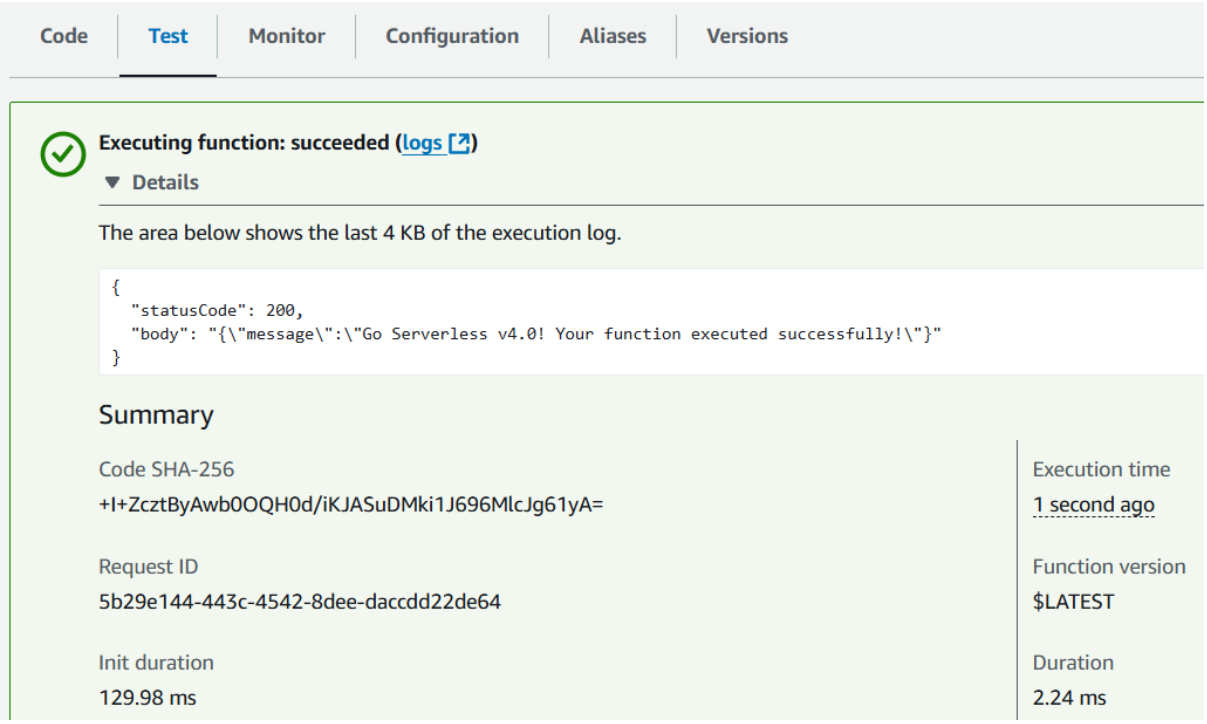
23. Also, if you go inside your cloud formation template and go to resources tab you will see the resources this template has created for you.



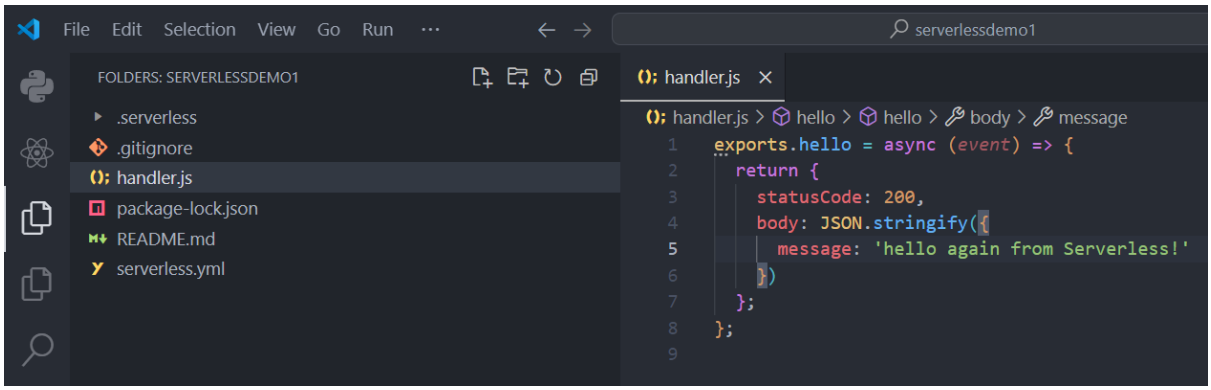
24. Now what you can do is open your Project folder in VS code. Here you will be able to read the code and the template which are used to create this whole scenario.



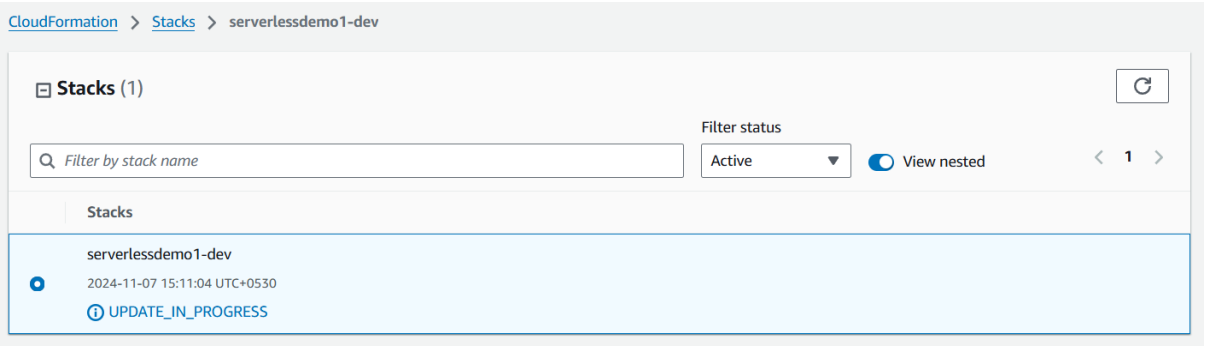
25. Go inside your lambda function and create a new test event and then test your lambda function. You will see the response.



26. So, what if we want to make changes to our code? As you can see below, we have changed the message. Now we are going to deploy our whole project again.



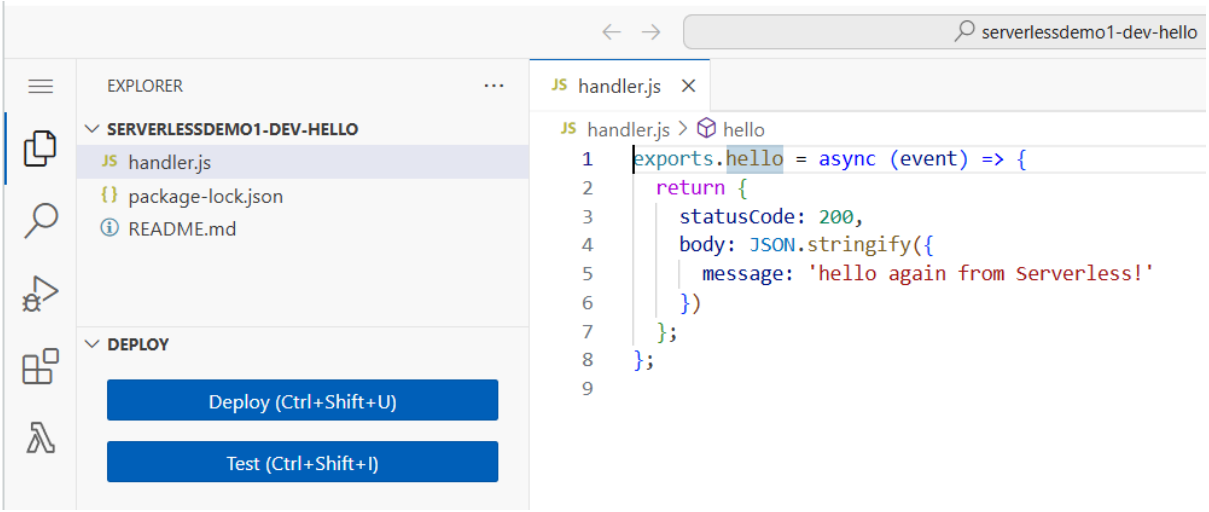
27. The changes will be deployed but this is the long method because it will recreate the stack and then push the changes to our lambda function. Also, if you go to cloud formation you will see that it is showing you the update is in progress and it will take a while.



28. Below you can see that the changes were deployed and it took 43 seconds.

```
C:\Serverless\serverlessdemo1>serverless deploy

(node:24488) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
Deploying "serverlessdemo1" to stage "dev" (us-east-1)

✔Service deployed to stack serverlessdemo1-dev (43s)

functions:
  hello: serverlessdemo1-dev-hello (1.7 kB)
```

29. If you go to the lambda function and refresh the page then in the code tab you will see those changes.



30. Now the smaller way is to just push the changes to our function only. And for that we need to know the name of our function. In VS code open the serverless.yml file and here you will see the name of your function.

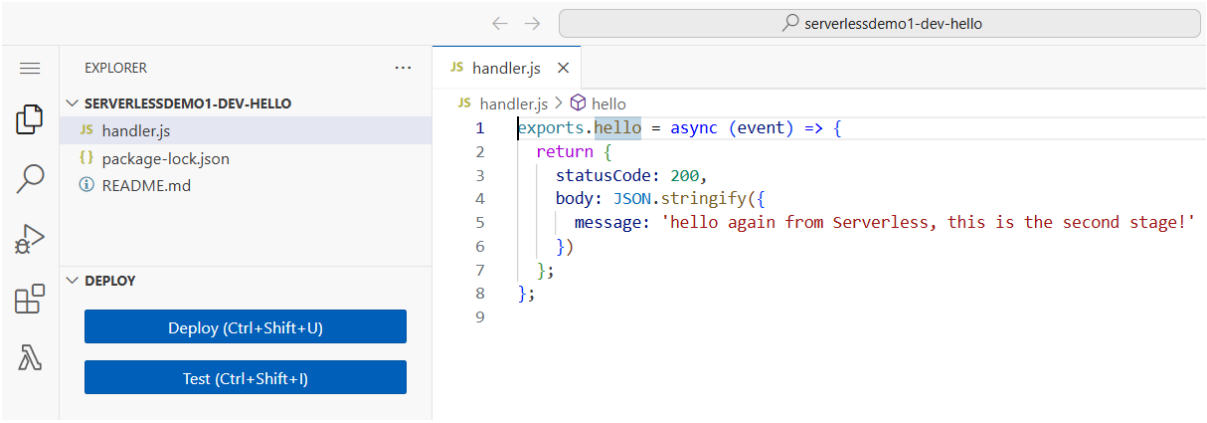
```
serverless.yml > ...
Serverless Framework Configuration - Serverless framework configuration files (reference.json)
# "org" ensures this Service is used with the correct Serverless Framework Access Key.
org: cloudfreeks
# "app" enables Serverless Framework Dashboard features and sharing them with other Services.
app: serverlessapp1
# "service" is the name of this project. This will also be added to your AWS resource names.
service: serverlessdemo1

provider:
  name: aws
  runtime: nodejs20.x

functions:
  hello:
    handler: handler.hello
```

31. Again, make some changes in the handler.js file and save them. After that you need to run this command.

    serverless deploy function -f hello

```
C:\Serverless\serverlessdemo1>serverless deploy function -f hello

(node:26376) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
Deploying function "hello" to stage "dev" (us-east-1)

✔Function code deployed (1s)

Function configuration did not change, so the update was skipped. If you made changes to the service configuration and expected them to be deploy
ed, this most likely means that they can only be applied with a full service deployment: "serverless deploy". (1s)
```

32. This time you can see that the update just took 1 second which is a lot faster approach. Go to lambda and refresh the page you will see the new code there.



33. Now we are going to invoke our lambda function using the CLI for that we just need to run this command.

    serverless invoke -f hello

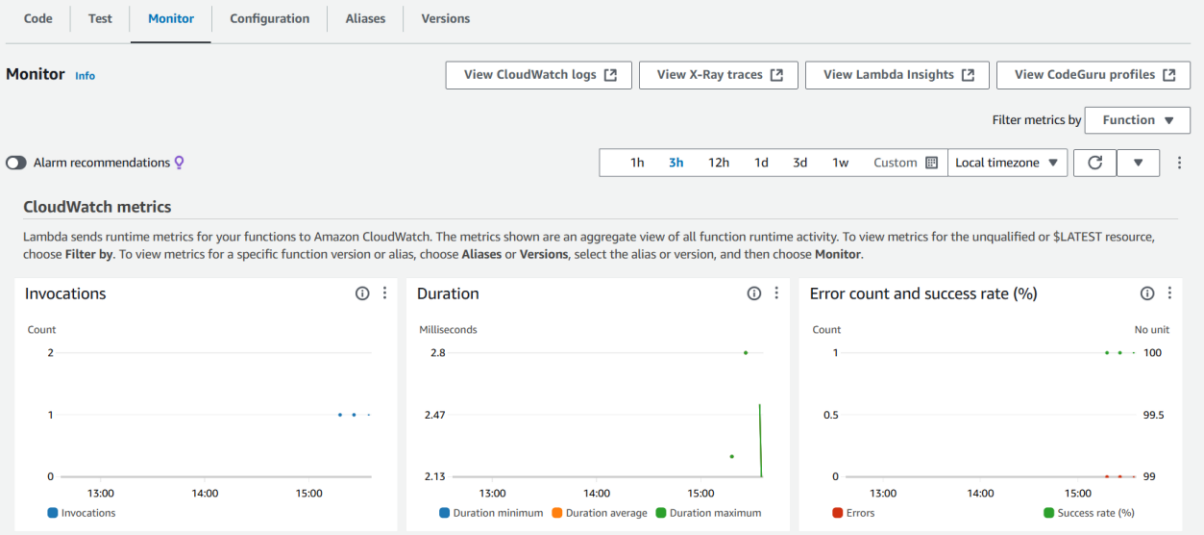
```
C:\Serverless\serverlessdemo1>serverless invoke -f hello

(node:25516) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
{
    "statusCode": 200,
    "body": "{\"message\":\"hello again from Serverless, this is the second stage!\"}"
}
```

34. If we also want to see the logs of our invoked function then we can run this command.

    serverless invoke -f hello --logs

```
C:\Serverless\serverlessdemo1>serverless invoke -f hello --log

(node:24260) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
{
    "statusCode": 200,
    "body": "{\"message\":\"hello again from Serverless, this is the second stage!\"}"
}

--------------------

START - 8afeecf6-fc59-4dd8-aeec-4e27c80543ba - Version: $LATEST

REPORT - 8afeecf6-fc59-4dd8-aeec-4e27c80543ba
Duration: 2.13 ms       Billed Duration: 3 ms   Memory Size: 1024 MB    Max Memory Used: 62 MB
```

35. Also, if you go back to your function in AWS Console and go to monitor tab you will see the invocation in the metrics section. You can also go to cloud watch and watch the log events in cloud watch log groups.



36. If you want to see that logs directly in your CLI then you can run this command.

serverless logs -f hello

```
C:\Serverless\serverlessdemo1>serverless logs -f hello

(node:24384) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
Logs Start Time: 2024-11-07 15:30:02

INIT_START Runtime - Version: nodejs:20.v43     Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:d087c1f568d9696a7d95ae9c69575a67caf267109d
19d9167c4ac5672dab9019
START - 57d0fa05-cb34-4e5f-adb1-1cc33fb39fce - Version: $LATEST
END - 57d0fa05-cb34-4e5f-adb1-1cc33fb39fce
REPORT - 57d0fa05-cb34-4e5f-adb1-1cc33fb39fce
Duration: 2.52 ms       Billed Duration: 3 ms   Memory Size: 1024 MB    Max Memory Used: 62 MB  Init Duration: 135.35 ms

START - 8afeecf6-fc59-4dd8-aeec-4e27c80543ba - Version: $LATEST
END - 8afeecf6-fc59-4dd8-aeec-4e27c80543ba
REPORT - 8afeecf6-fc59-4dd8-aeec-4e27c80543ba
Duration: 2.13 ms       Billed Duration: 3 ms   Memory Size: 1024 MB    Max Memory Used: 62 MB
```

37. Once you are done with the lab you need to run this command to delete all the resources.

serverless remove

```
C:\Serverless\serverlessdemo1>serverless remove

(node:5008) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
Removing "serverlessdemo1" from stage "dev" (us-east-1)

✔Service serverlessdemo1 has been successfully removed (31s)
```