## Capstone Project Use Case 2 Documentation

### 

### **Company Profile**

SaleSpheres is an e-commerce company with the support of engineers who are passionate in emerging technologies and with desires to excel with the Company on driving innovation frontier. Every individual has different values to contribute and with great attitude of creating a collaborative environment.

### **Site Reliability Engineer Team**

Our engineers are the pillars behind our innovative products. They leverage their expertise in AWS cloud engineering skills to design, develop, and implement robust and scalable software solutions. Their skills range from front-end development for intuitive user interfaces to back-end systems that power our products' functionality.

### **Project**

Project Name: SaleSpheres

Repository: https://github.com/flowstarts2020/SaleSpheres.git

The Site Reliability Engineer Team at SaleSpheres is currently working on an exciting project manifold complexity in systems’ health monitoring.

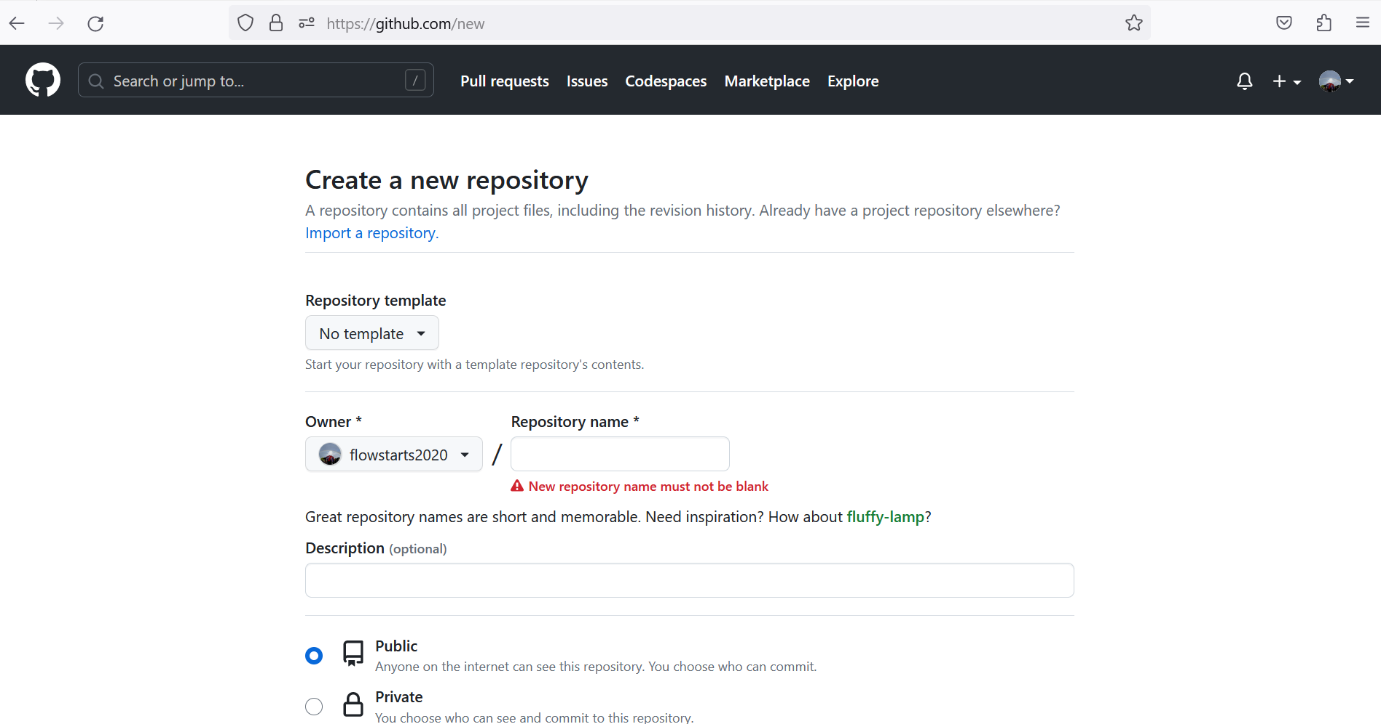
This project involves the development of application in AWS that implements a serverless architecture components as well as exploring Sense Deep a third-Party tool. The project will commerce with this objective and with further iteration improvement versions to be better.

### 

### **Getting started**

To get the serverless architecture running locally:

## Step 1: Create a code repository in GitHub

Create via github account [](https://user-images.githubusercontent.com/69182919/242765673-d9709a6c-1013-46f7-9fa5-1f703b4abda0.png)

## Step 2: Clone the code repository into local machine

Clone by the following command on visual studio code - PS C:> git clone <https://github.com/flowstarts2020/SaleSpheres.git>

## Step 3: Create index.js file

Example of an index.js file which was done via visual studio code

module.exports.handler = async (event) => {

const respose = {

"timestamp": Date.now().toString(),

"status\_code": 200,

"body": "body"

}

console.log(respose)

return {

statusCode: 200,

body: JSON.stringify(

{

message: "Go Serverless v3.0! Your function executed successfully!",

input: respose,

},

null,

2

),

};

};

module.exports.error = async (event) => {

const respose = {

"timestamp": Date.now().toString(),

"status\_code": 400,

"body": "body error"

}

const err = new Error(respose)

return {

statusCode: 400,

body: JSON.stringify(

{

message: "error",

input: err.message.body,

},

null,

2

),

};

};

## Step 4: Create serverless.yml

Example of a serverless.yml file which was done via visual studio code

salesphere

service: salesphere

frameworkVersion: '3'

provider:

name: aws

runtime: nodejs18.x

region: ap-southeast-1

functions:

api:

handler: index.handler

events:

- httpApi:

path: /

method: get

error:

handler: index.error

events:

- httpApi:

path: /error

method: get

plugins:

- serverless-offline

## Step 5: Deploy and verify that the serverless application is working

Test and execute the following commands $ npm install

and then deploy with:

$ serverless deploy

## Step 6: Create CI/CD pipeline with GitHub Actions

Create main.yml in .github/workflows folder. An example of a main.yml

name: CICD for Serverless Application

run-name: ${{ github.actor }} is doing CICD for Serverless Application

on:

push:

branches: [ main, "\*"]

jobs:

pre-deploy:

runs-on: ubuntu-latest

steps:

- run: echo "🎉 The job was automatically triggered by a ${{ github.event\_name }} event"

- run: echo "🐧 This job is now running on a ${{ runner.os }} server hosted by GitHub!"

- run: echo "🔎 The name of your branch is ${{ github.ref }} and your repository is ${{ github.repository }}."

install-dependencies:

runs-on: ubuntu-latest

needs: pre-deploy

steps:

- name: Check out repository code

uses: actions/checkout@v3

- name: Run Installation of Dependencies Commands

run: npm install

scan-dependencies:

runs-on: ubuntu-latest

needs: install-dependencies

steps:

- name: Check out repository code

uses: actions/checkout@v3

- name: Run Scanning of Dependencies Commands

run: npm audit

deploy:

name: deploy

runs-on: ubuntu-latest

needs: install-dependencies

strategy:

matrix:

node-version: [18.x]

steps:

- uses: actions/checkout@v3

- name: Use Node.js ${{ matrix.node-version }}

uses: actions/setup-node@v3

with:

node-version: ${{ matrix.node-version }}

- run: npm ci

- name: serverless deploy

uses: serverless/github-action@v3.2

with:

args: deploy

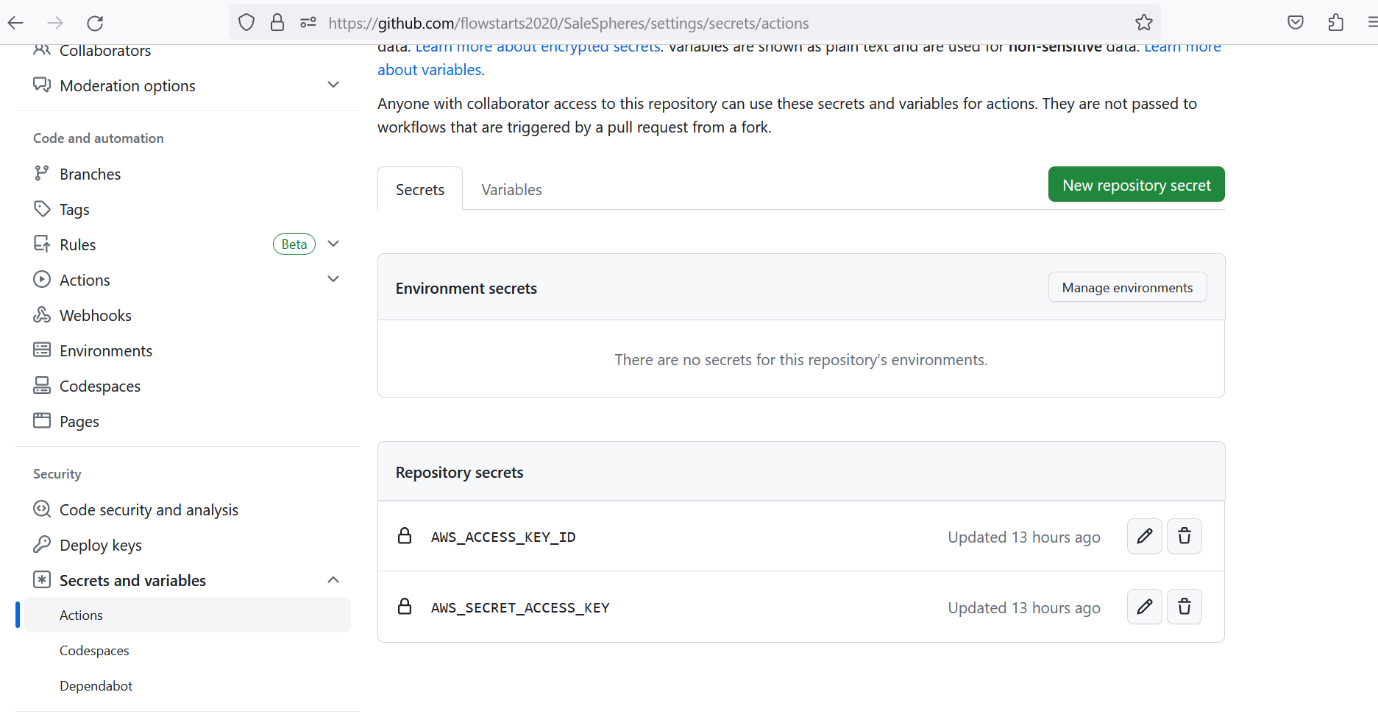
env:

AWS\_ACCESS\_KEY\_ID: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

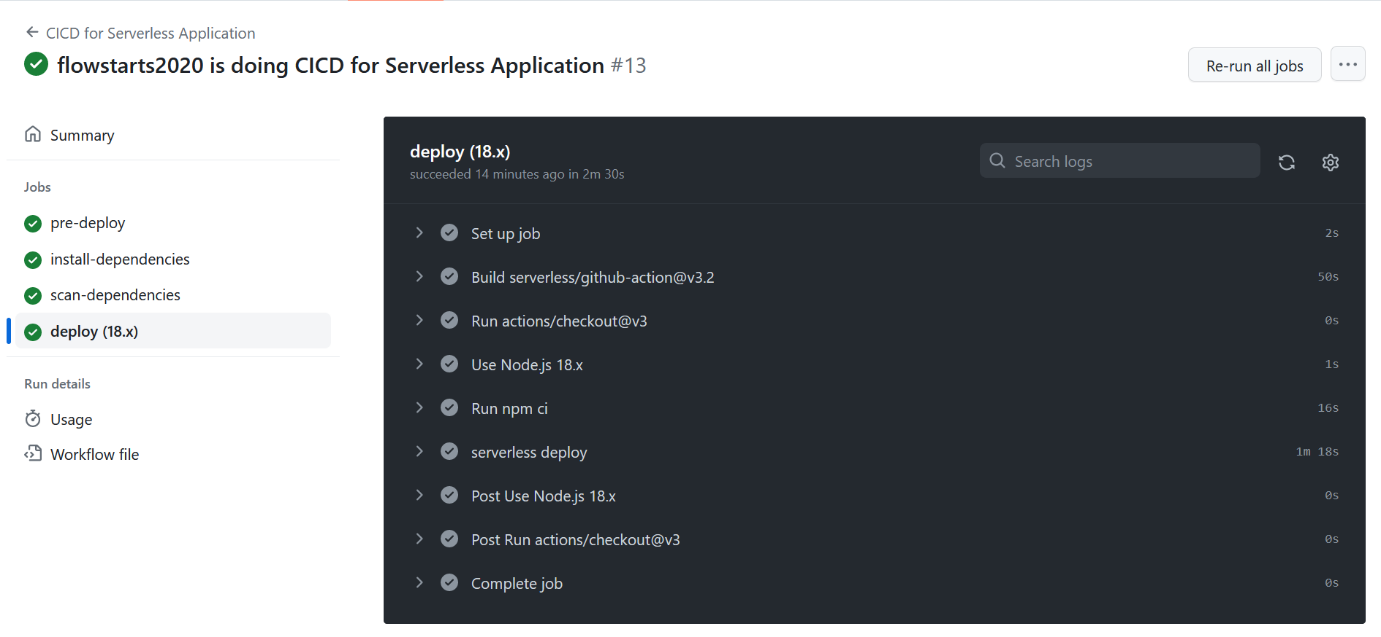
AWS\_SECRET\_ACCESS\_KEY: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

## Step 7: Add AWS\_ACCESS\_KEY\_ID and ASW\_SECRET\_ACCESS\_KEY to GitHub Secrets

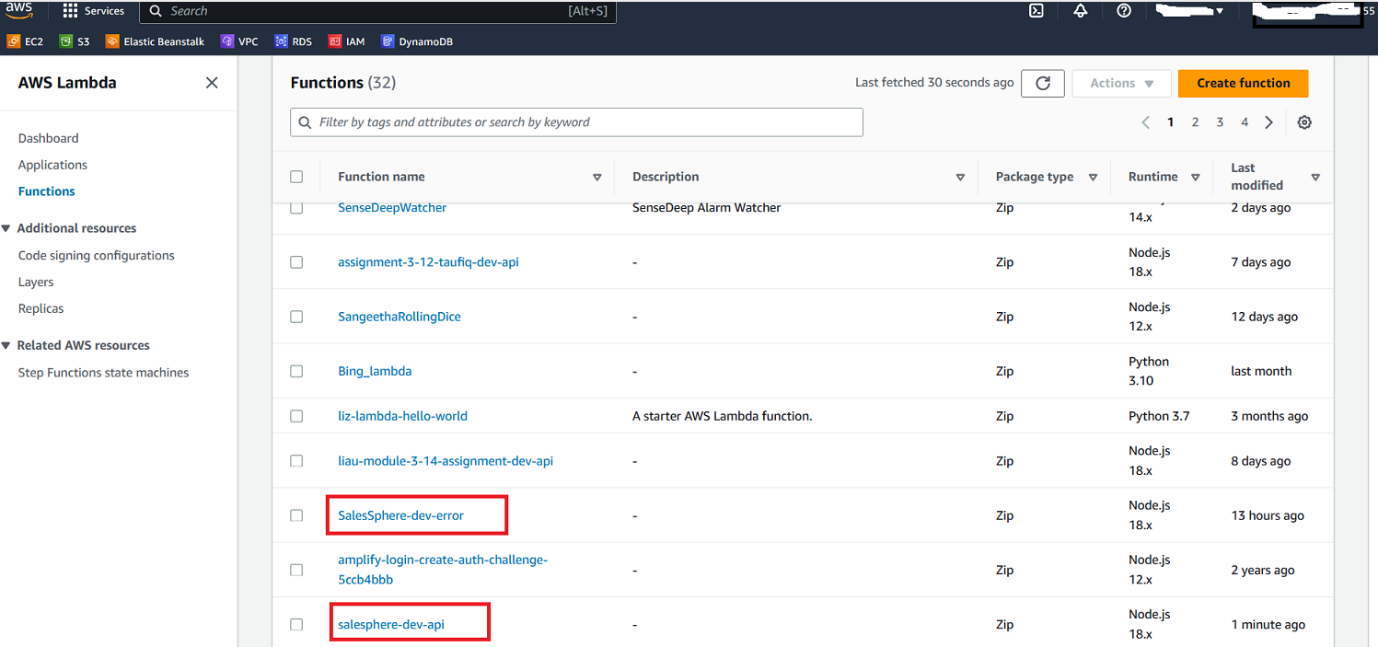
Always keep AWS\_ACCESS\_KEY\_ID and ASW\_SECRET\_ACCESS\_KEY in privately. They are not meant to share or let public known for prevention of data breaches and landed to the wrong hands (hackers)

[](https://user-images.githubusercontent.com/69182919/242764756-1988e2c1-64b2-48b7-975d-311a67cfac09.png)

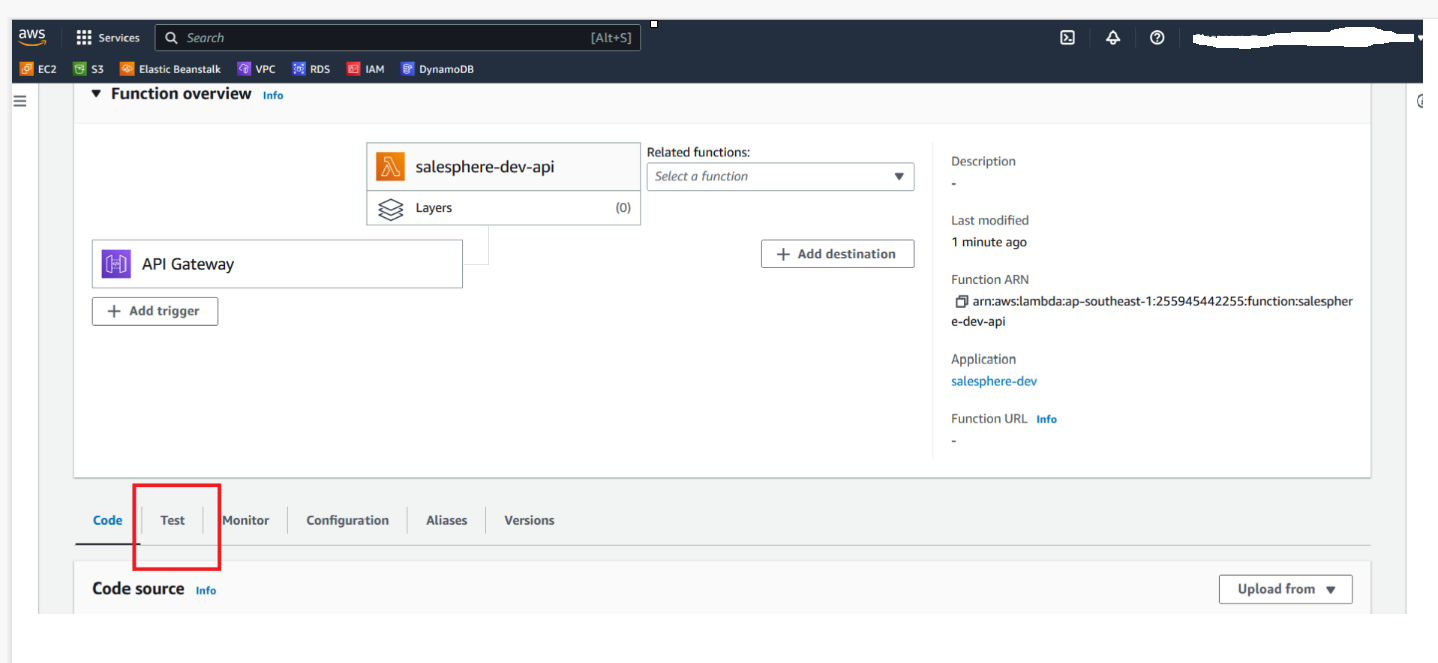
## Step 8: Push changes to GitHub to start the workflow

Commit changes locally and push it to GitHub. Navigate the repo on GitHub, click on the **Actions** tab to see the workflows. Confirm all tests passed and without errors [](https://user-images.githubusercontent.com/69182919/242764104-d2b06c16-74ae-4997-92a8-32757c41c9c5.png)

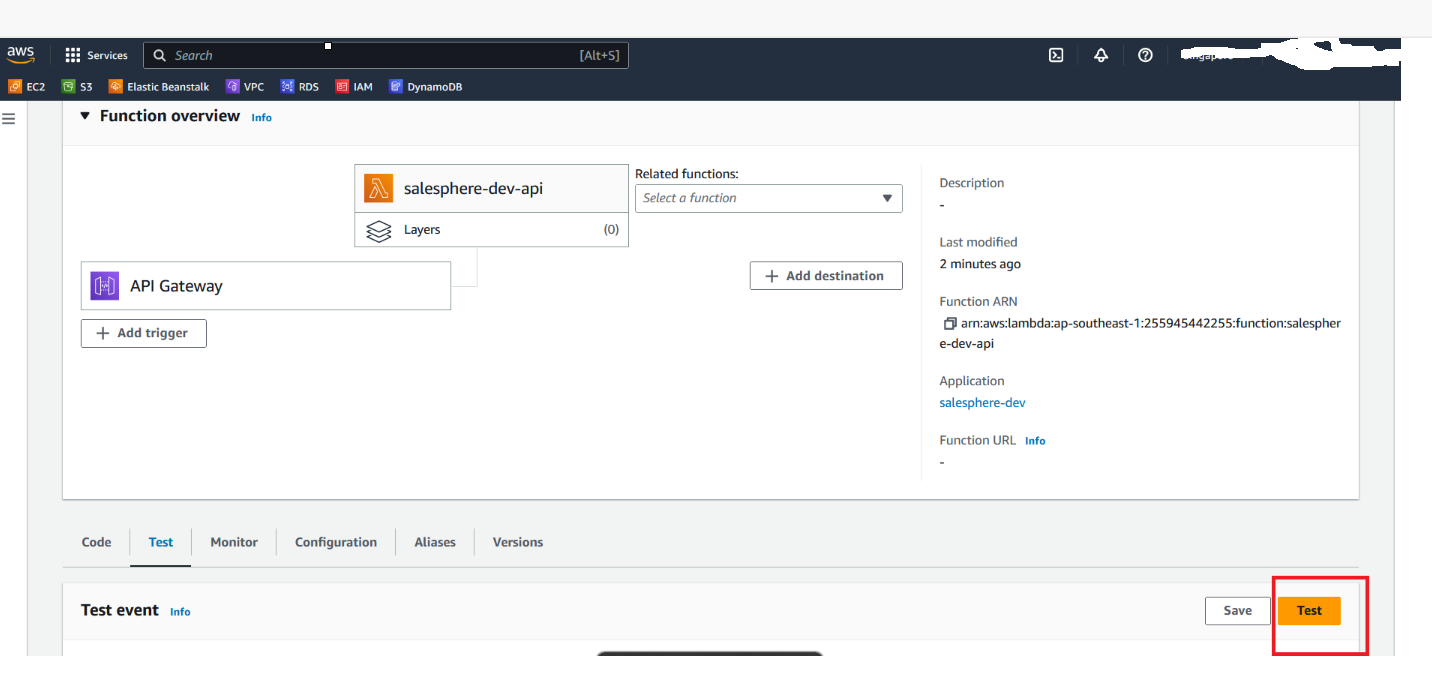
## Step 9: Access AWS Console to check the Lambda application (on salesphere-dev-api and SalesSphere-dev-error)

[](https://user-images.githubusercontent.com/69182919/242772648-61bffb68-74dd-4ed2-963d-8628b5232c8e.png)

## Step 10: Test many times to Invoke the Lambda application

[](https://user-images.githubusercontent.com/69182919/242773118-33662d3d-8d24-46f1-b301-811640f1f7ff.png)

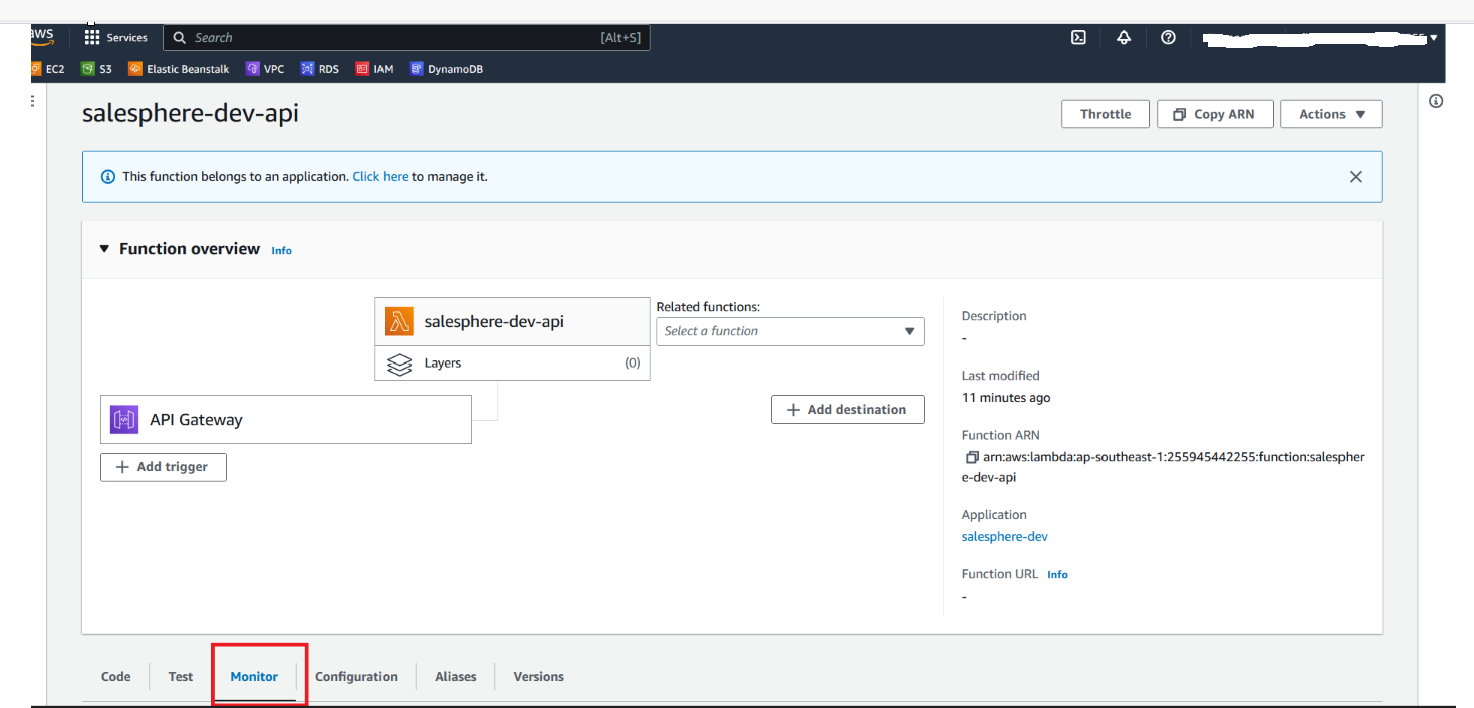
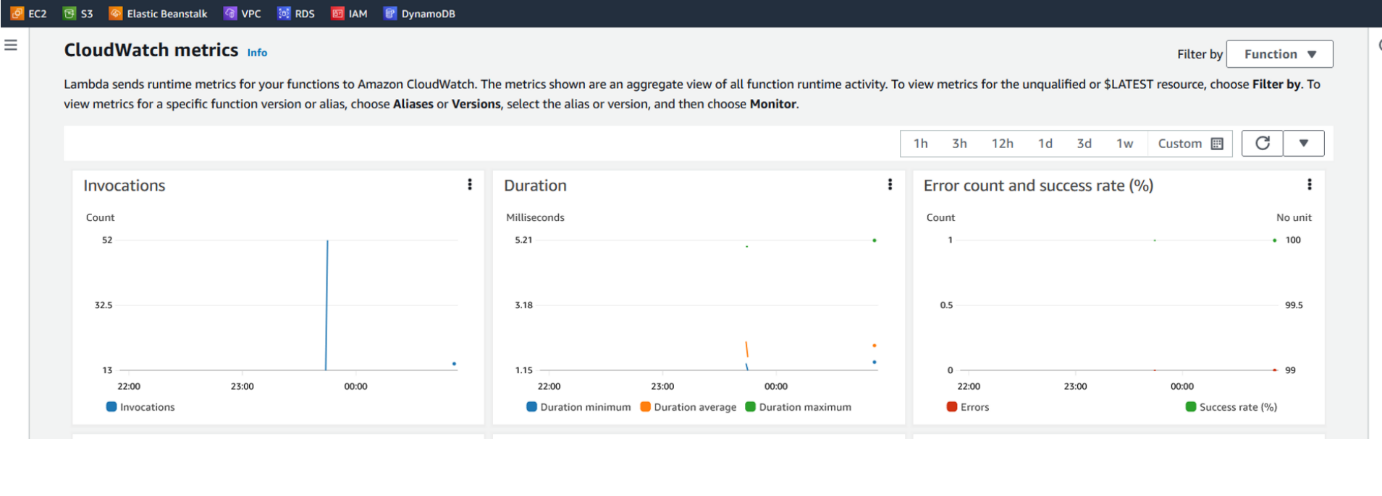
Go to the test and click

[](https://user-images.githubusercontent.com/69182919/242773674-0a5f810d-c08c-45eb-a172-37e942359282.png)

Click a few times on the orange test box to invoke

(for both on salesphere-dev-api and SalesSphere-dev-error)

## Step 11: Check any readings when monitoring the Lambda application

[](https://user-images.githubusercontent.com/69182919/242774118-f002170d-cafd-4ea0-8aae-1afac018fc64.png)[](https://user-images.githubusercontent.com/69182919/242774503-54834832-ae06-4c4f-a877-97f8a077002a.png)

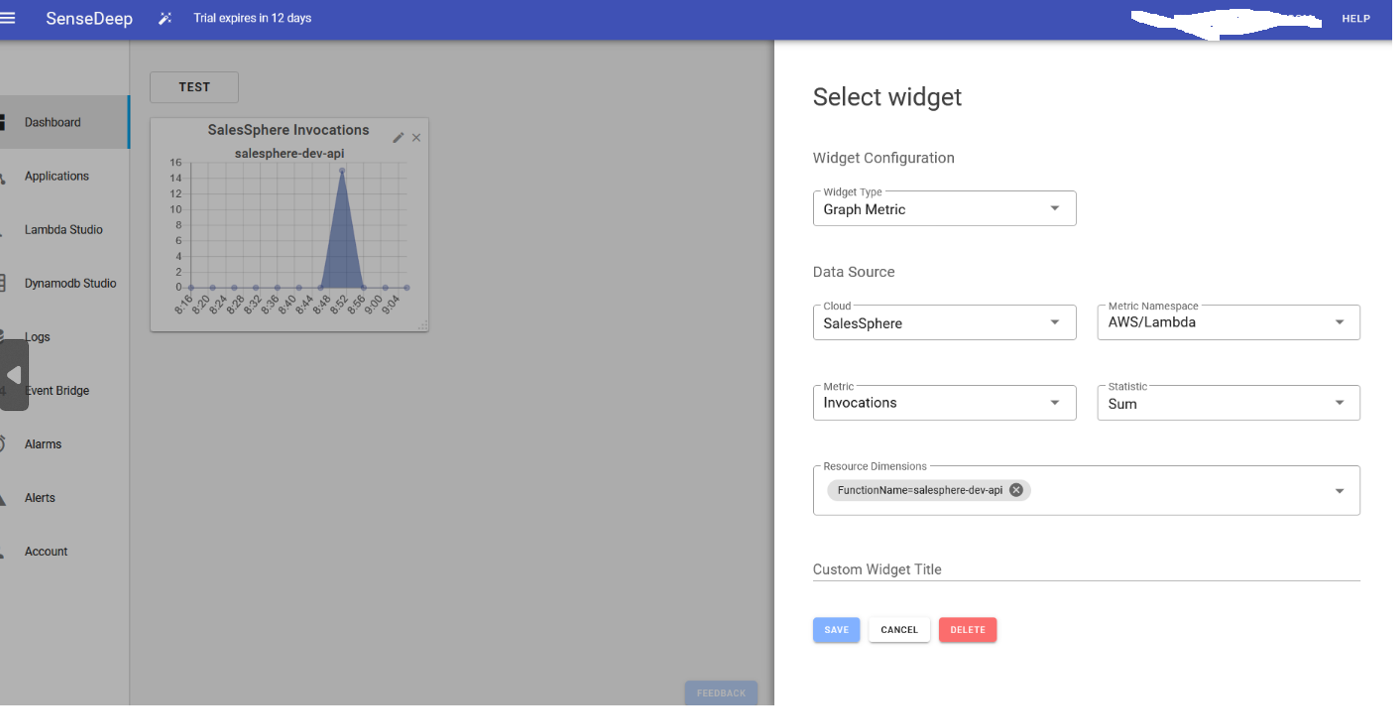
## **Application logging**

## Access and login SenseDeep application

<https://www.sensedeep.com/> can be register for free. For this project, we will access as a free account Do go to this link <https://www.sensedeep.com/doc/kb/starting/setup.html> on how to register and setup

## Create the widgets of graph views on SUM and Average for the related relevant monitoring Lambda application

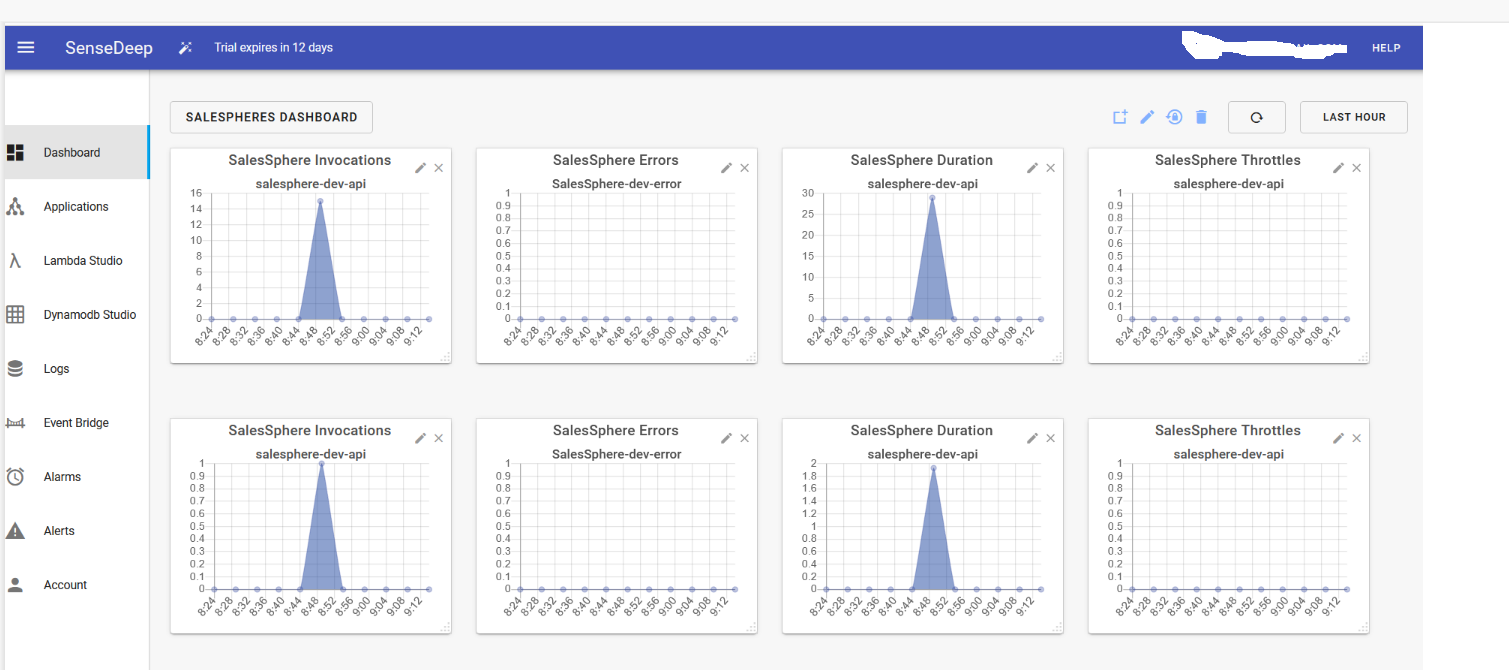
illustrating one screenshot example of a SUM widget on salesphere-dev-api

[](https://user-images.githubusercontent.com/69182919/242771120-2ce1f85b-418c-4441-bdc5-79b5bffe87b6.png)

## **Application Monitoring Dashboard**

## Completion on ready to view centralize Dashboard

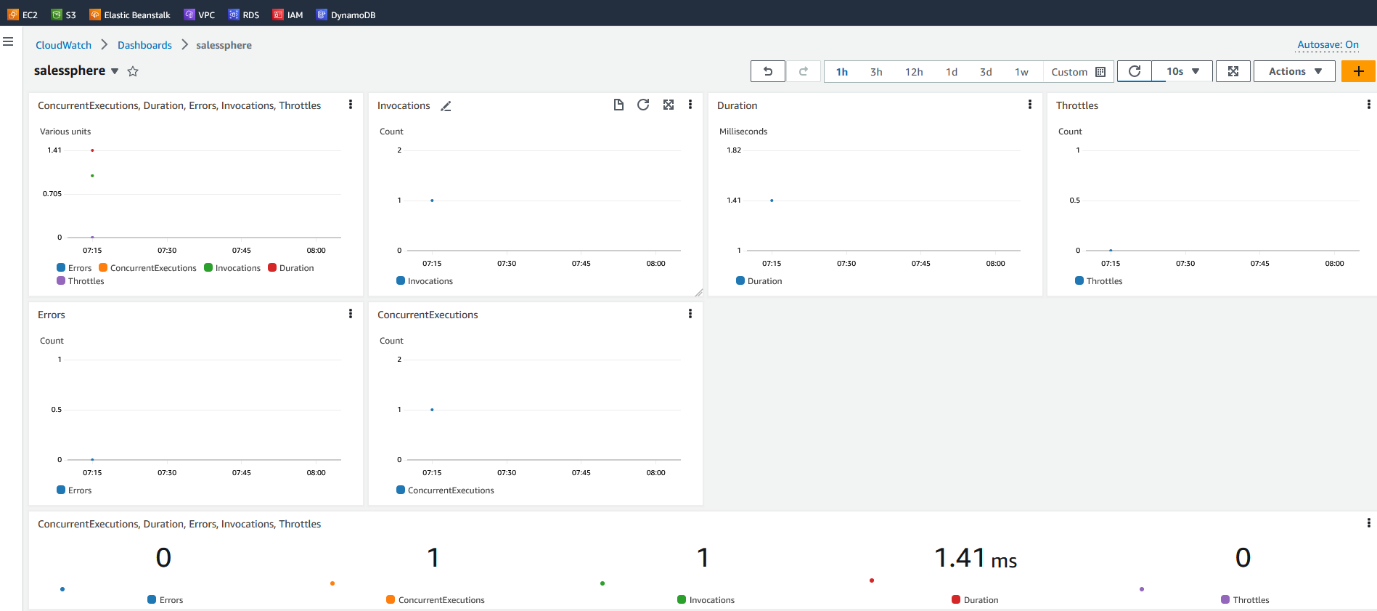
This is the illustration of a detail widget based on SUM and Average hourly on salesphere-dev-api and SalesSphere-dev-error.

SALESPHERES DASHBOARD [](https://user-images.githubusercontent.com/69182919/242771557-c56dac80-741c-48ed-9daf-3a47f34f664f.png)

## **Other information - FAQ**

What if Sensedeep is not available or outage?

No issue, we have the widgets dashboard created on CloudWatch updating based on sale sphere-dev-api. The overall factors on Concurrent Execution, Duration, Errors, Invocations, Throttles and by each factor. We have an overall factor in numbers. Below is an example

[](https://user-images.githubusercontent.com/69182919/242826717-fc5479d9-b61e-4768-9227-3fee03cc0fbe.png)

Definition

Invocations -The number of times that your function code is invoked, including successful invocations and invocations that result in a function error.

Duration - The amount of time that your function code spends processing an event. The billed duration for an invocation is the value of Duration rounded up to the nearest millisecond.

Errors - The number of invocations that result in a function error.

Concurrent Executions - The number of function instances that are processing events. If this number reaches your concurrent executions quota for the Region, or the reserved concurrency limit on the function, then Lambda throttles additional invocation requests.

Throttles - The number of invocation requests that are throttled.

END