Group 1 Capstone Project

-SaleSpheres

Alan How

Jian Tao

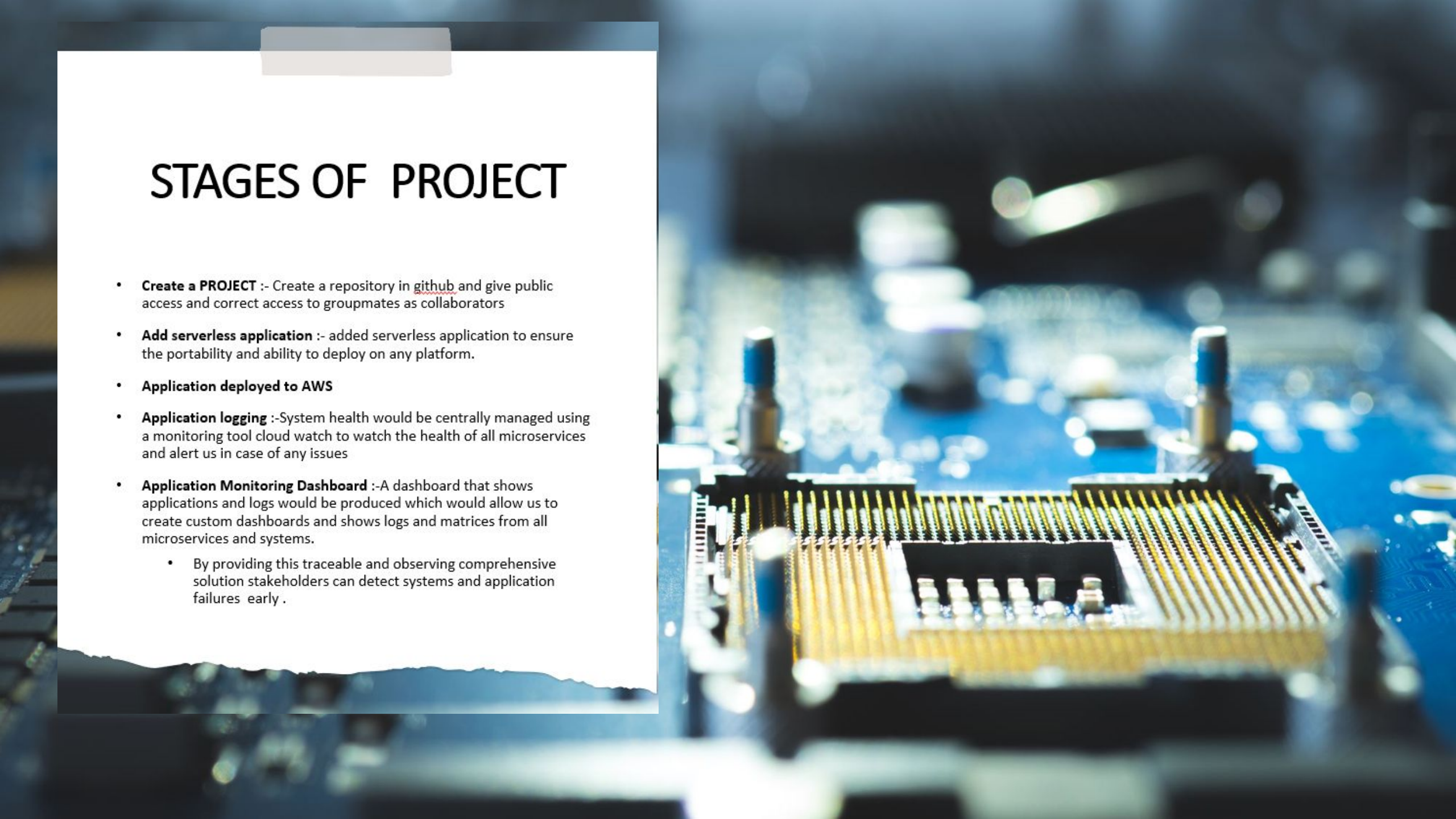
Elezabeth

Monitoring Large Application SRE(Site Reliability Engineering)

- A comprehensive monitoring solution that can detect and flag out systems and applications failure early

STAGES OF PROJECT

- **Create a PROJECT** :- Create a repository in github and give public access and correct access to groupmates as collaborators
- **Add serverless application** :- added serverless application to ensure the portability and ability to deploy on any platform.
- **Application deployed to AWS**
- **Application logging** :-System health would be centrally managed using a monitoring tool cloud watch to watch the health of all microservices and alert us in case of any issues
- **Application Monitoring Dashboard** :-A dashboard that shows applications and logs would be produced which would allow us to create custom dashboards and shows logs and matrices from all microservices and systems.
    - By providing this traceable and observing comprehensive solution stakeholders can detect systems and application failures early .



Company Profile

- SaleSpheres is a software company driving and pushing innovation frontiers.
- Support Engineers are passionate in emerging technologies and desires to excel and work for success of the company
- Every individual has different values and capabilities to contribute and with great attitude of creating a collaborative environment.

Customer – MNC Company (XYZ)

SOFTWARE COMPANY

CUSTOMERS ARE MANY LIKE INSIGHT, DBS , SMBC

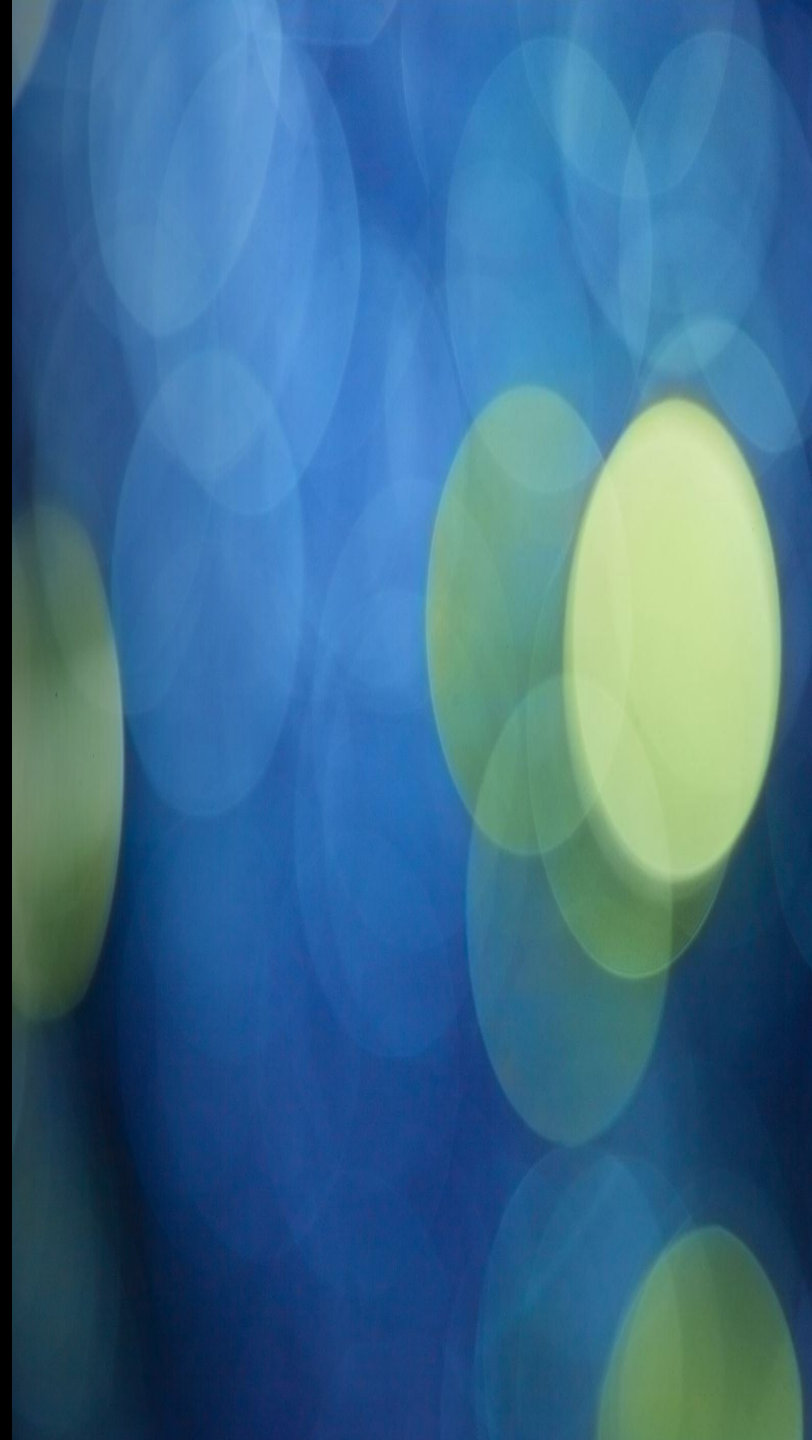
FINDING ON HEALTH CHECK FOR A SOLUTION FROM US TO CLEAR THEIR PAINS AND NEEDS

CLOUD SOLUTIONING HEALTH CHECK

DASH BOARD CONTAINS INVOCATIONS, DURATION, ERRORS , CONCURRENT EXECUTION, THROTTLES ,

WE ARE GIVING A PROOF OF CONCEPT(POC) HOW TO MONITOR THE CLOUD SOLUTION

FIRST WE ARE MONITORING SERVERLESS PLATFORM BASED ON LAMBDA



Site Reliability Engineer Team

- Our engineers are the pillars of 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.

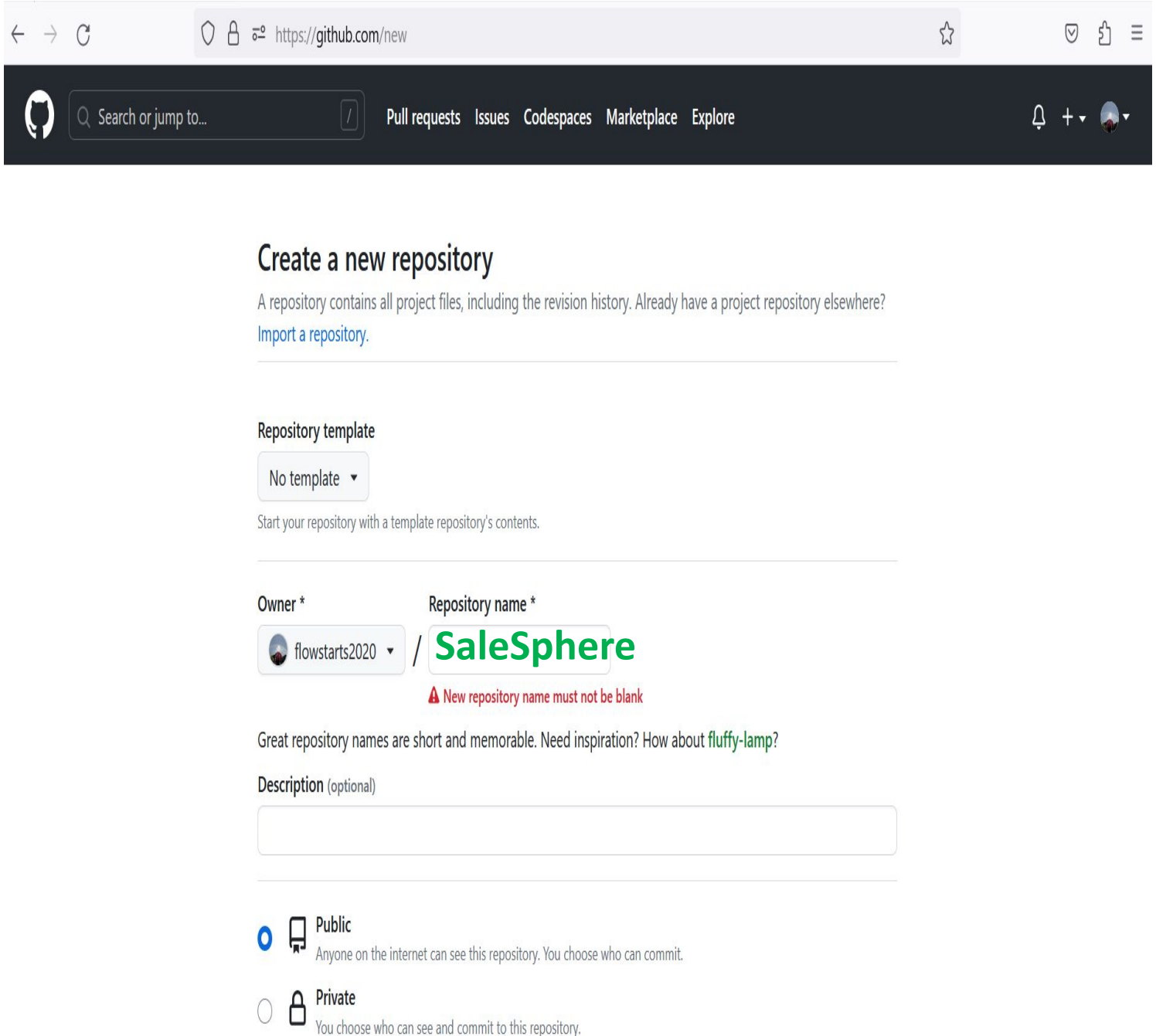
Project-SaleSpheres

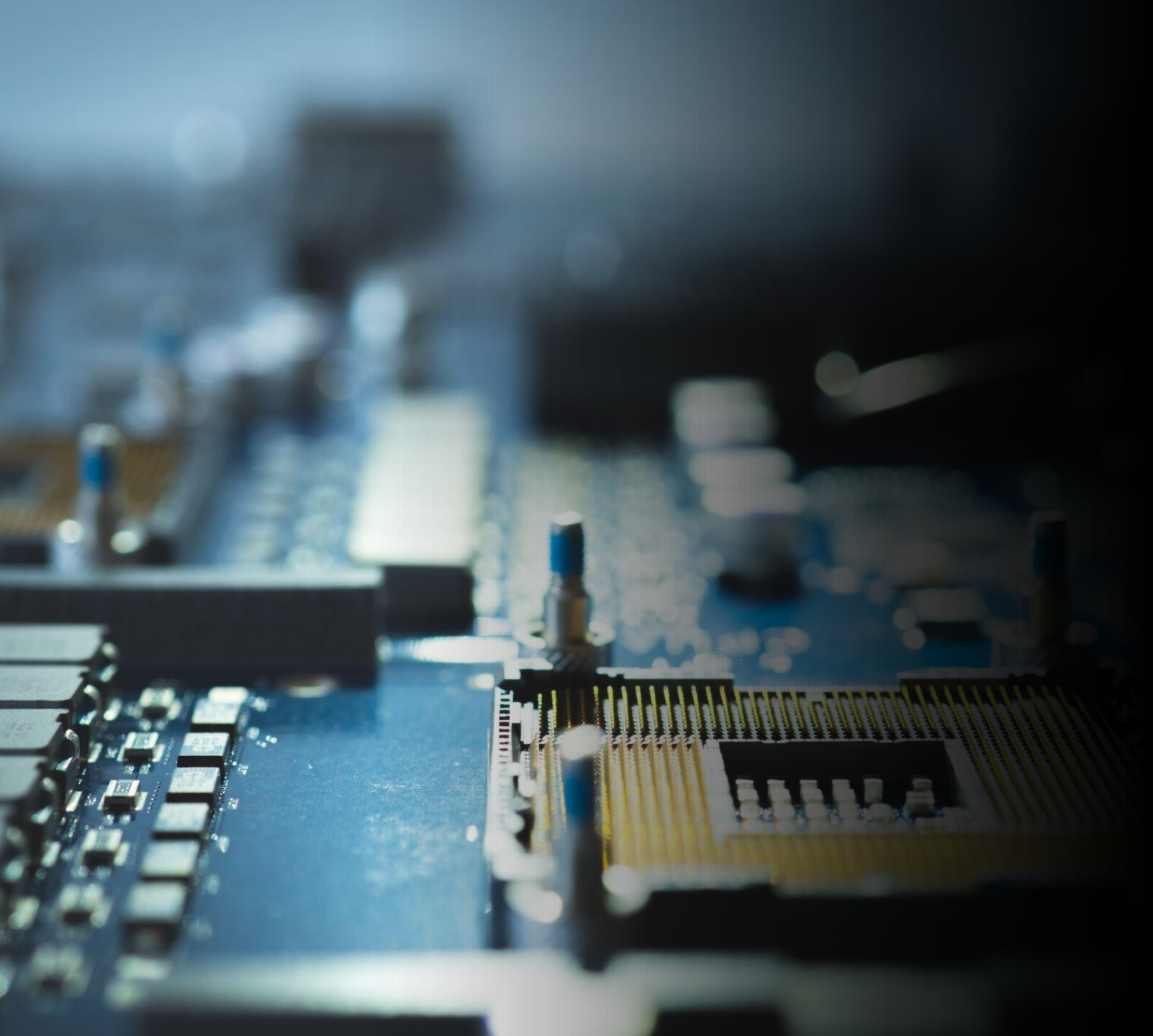
- Repository: **https://github.com/flowstarts2020/SaleSpheres.git**
- The Site Reliability Engineer Team at **SaleSpheres** is currently working on an exciting project manifold complexity in systems' health monitoring.
- This project involves the development of application in AWS that implements a serverless 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.

To get the serverless architecture running locally:

Step 1:

Create a repository in GitHub





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

```
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 3:
Create index.js file

Example of ourindex.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 succe
        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,
      },
```

```
 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
```

```
error:
  handler: index.error
  events:
    - httpApi:
        path: /error
        method: get

plugins:
  - serverless-offline
```

Step 4: Create serverless.yml

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

Step 5:

Deploy and verify that the serverless application is working

Test and execute the following commands > $ npm install

then deploy with ->

$ serverless deploy

Step 6:

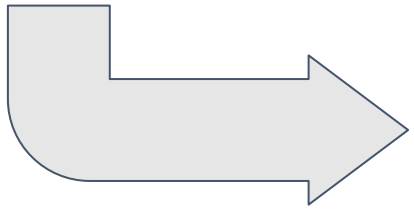
Run and test with GitHub Actions

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

```yaml
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
```

```yaml
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
```

Continue..

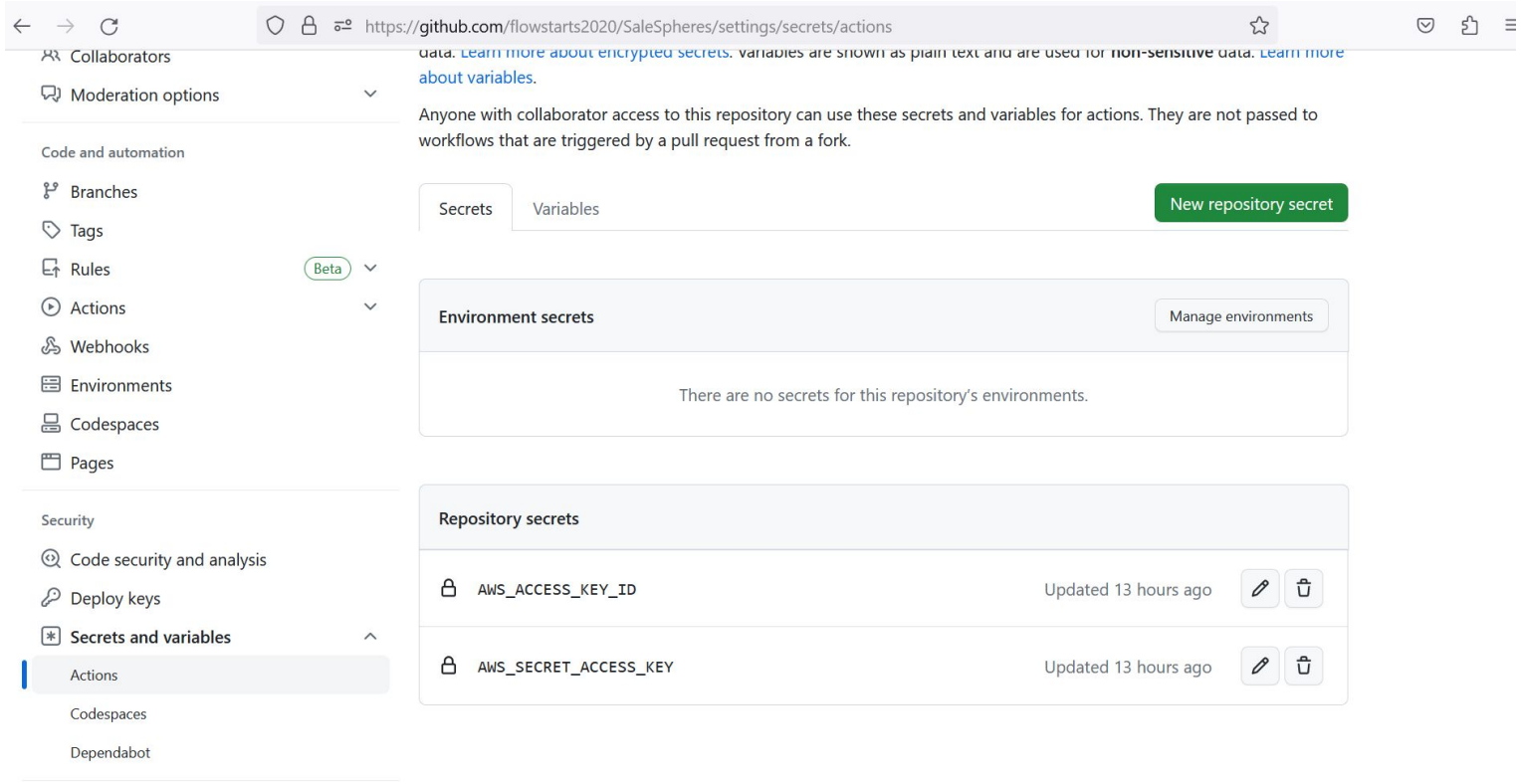


```yaml
- name: serverless deploy
  uses: serverless/github-action@v3.2
  with:
    args: deploy
  env:
    AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
    AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
```

Step 7:

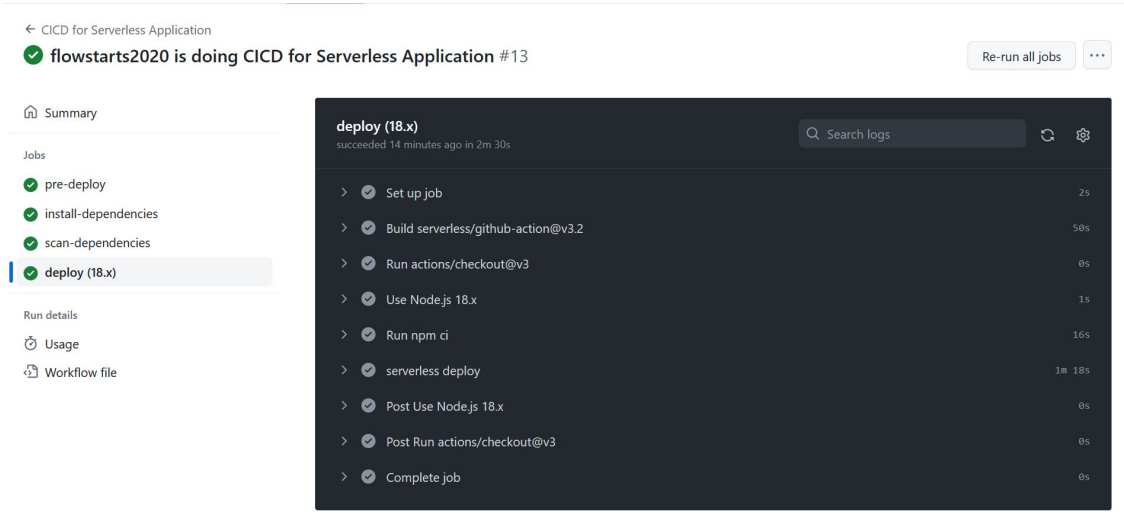
ADD
AWS_ACCESS_KEY_ID
&
AWS_SECRET_ACCESS_KEY
to GitHub Secrets

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)



Step 8:

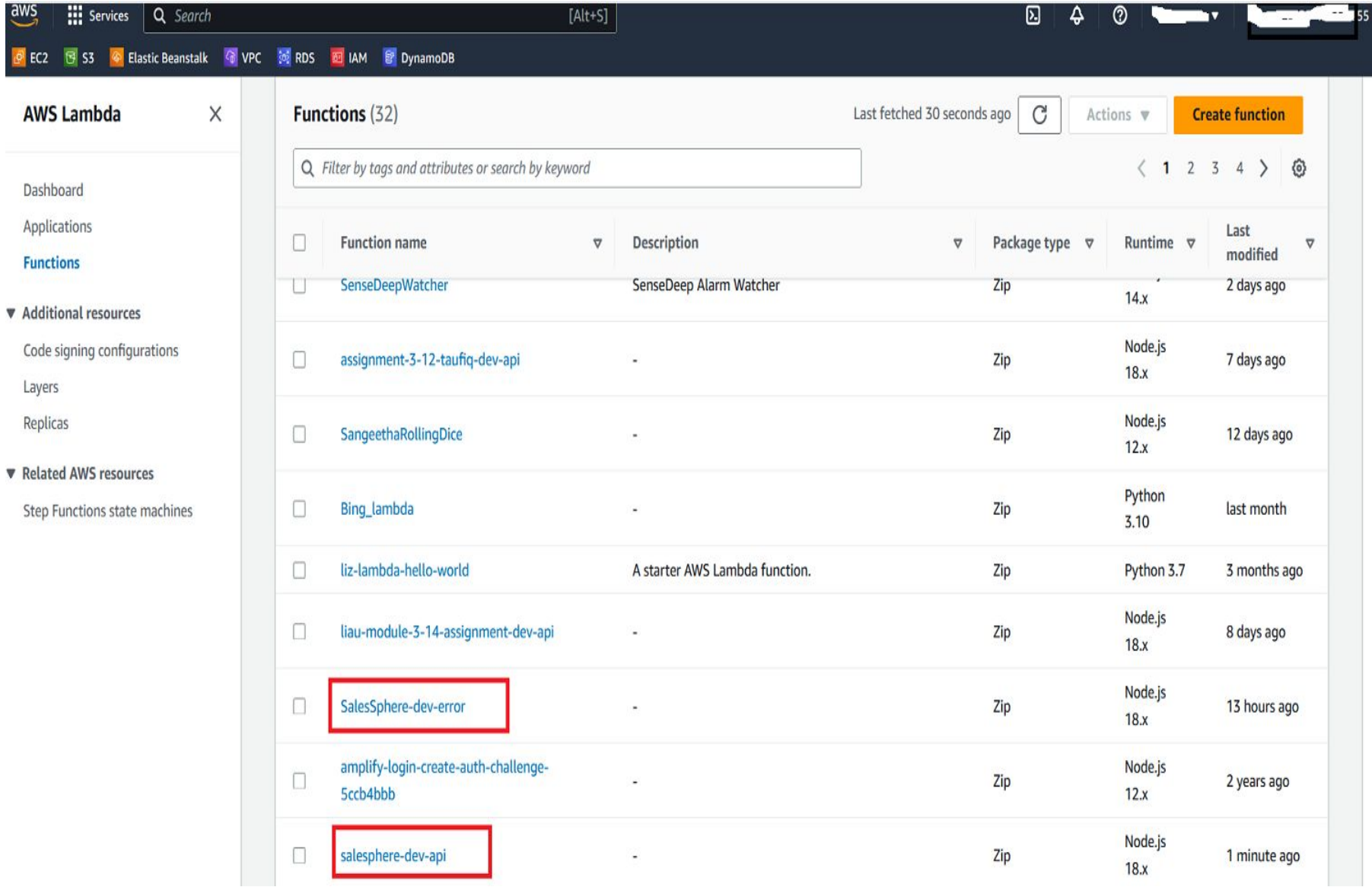
Push changes to GitHub to start the workflow

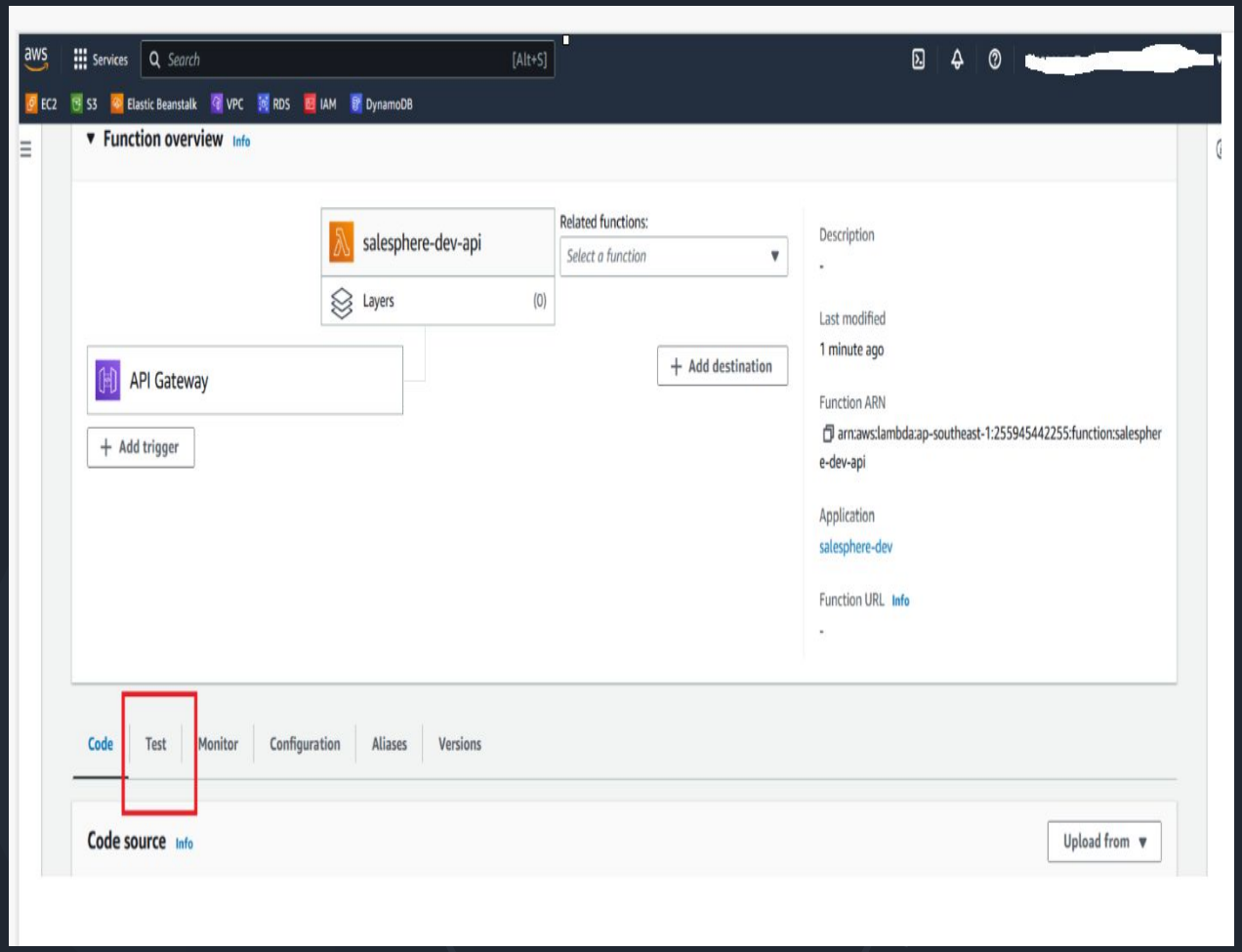


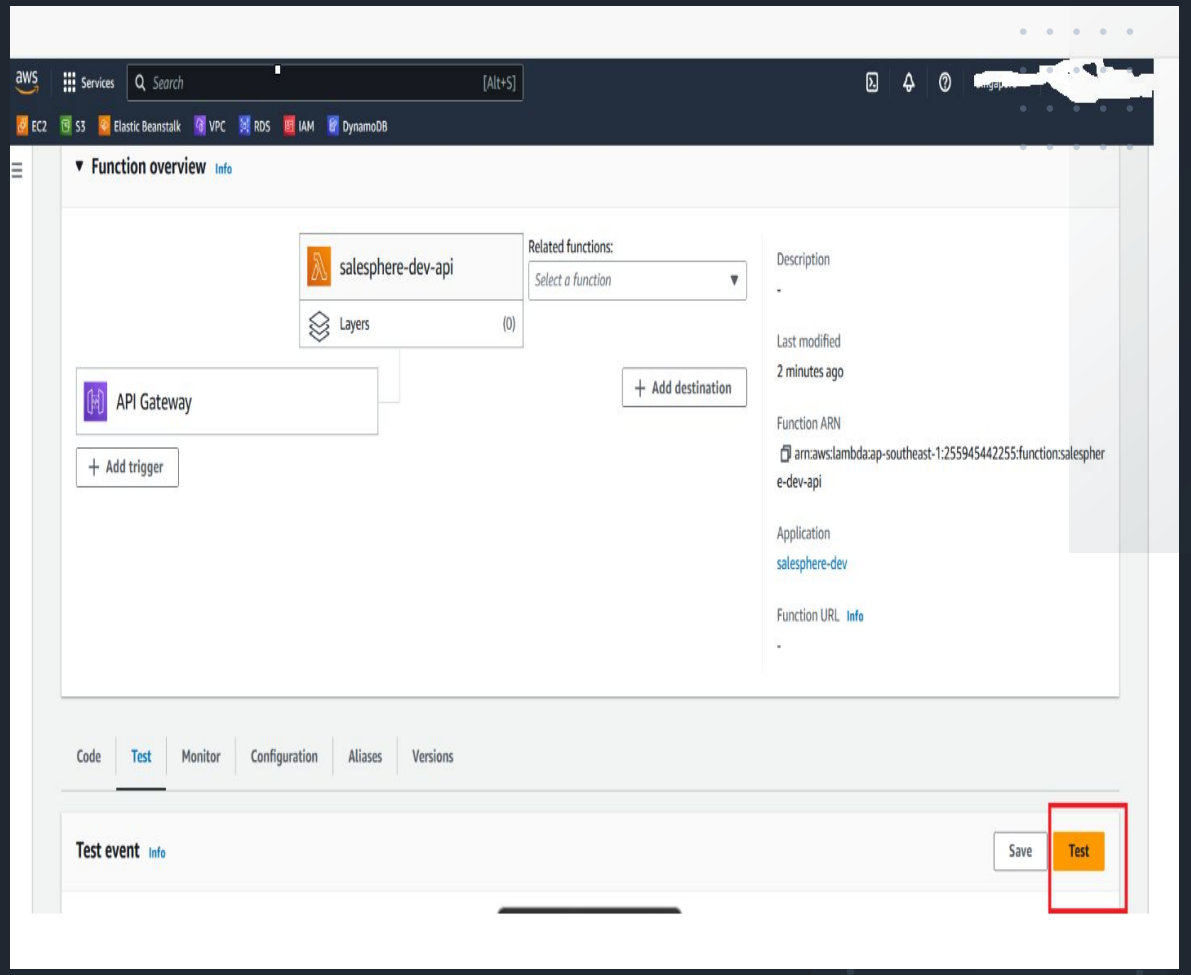
•Commit changes locally and push it to GitHub. Navigate the repo on GitHub, click on the **Actions** tab to see the workflows. Confirm all tests passed and without errors

Step 9:

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

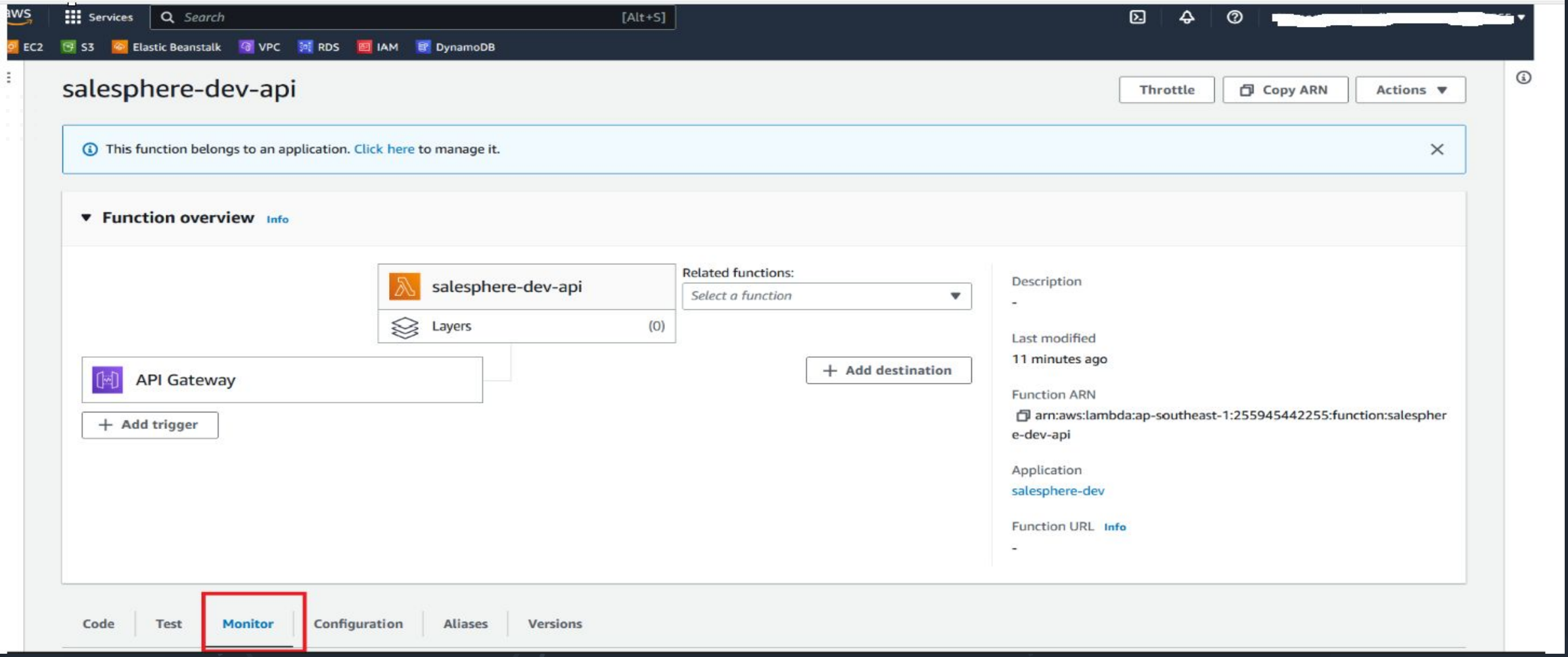






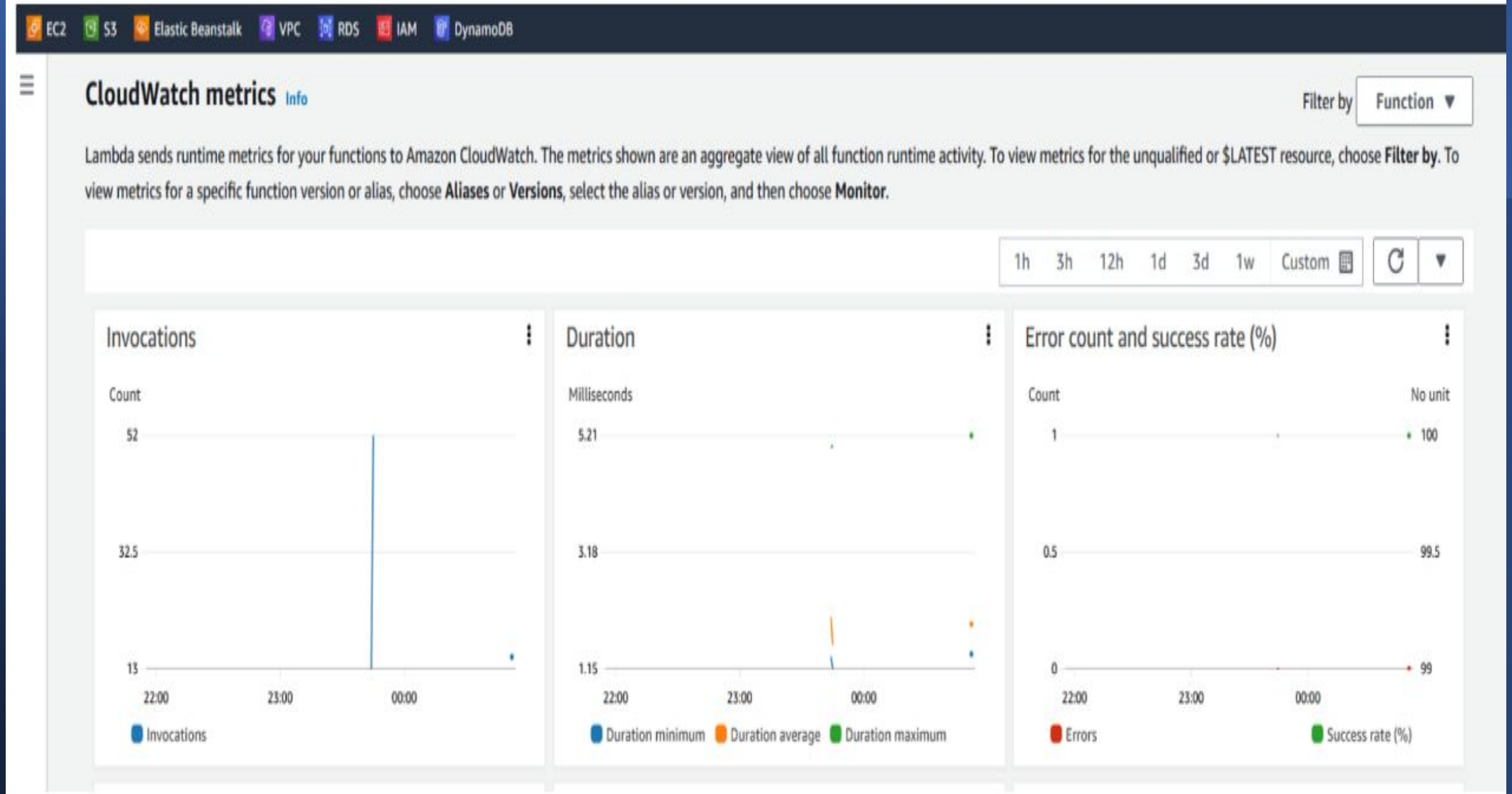
Step 10:

Test to Invoke the Lambda application



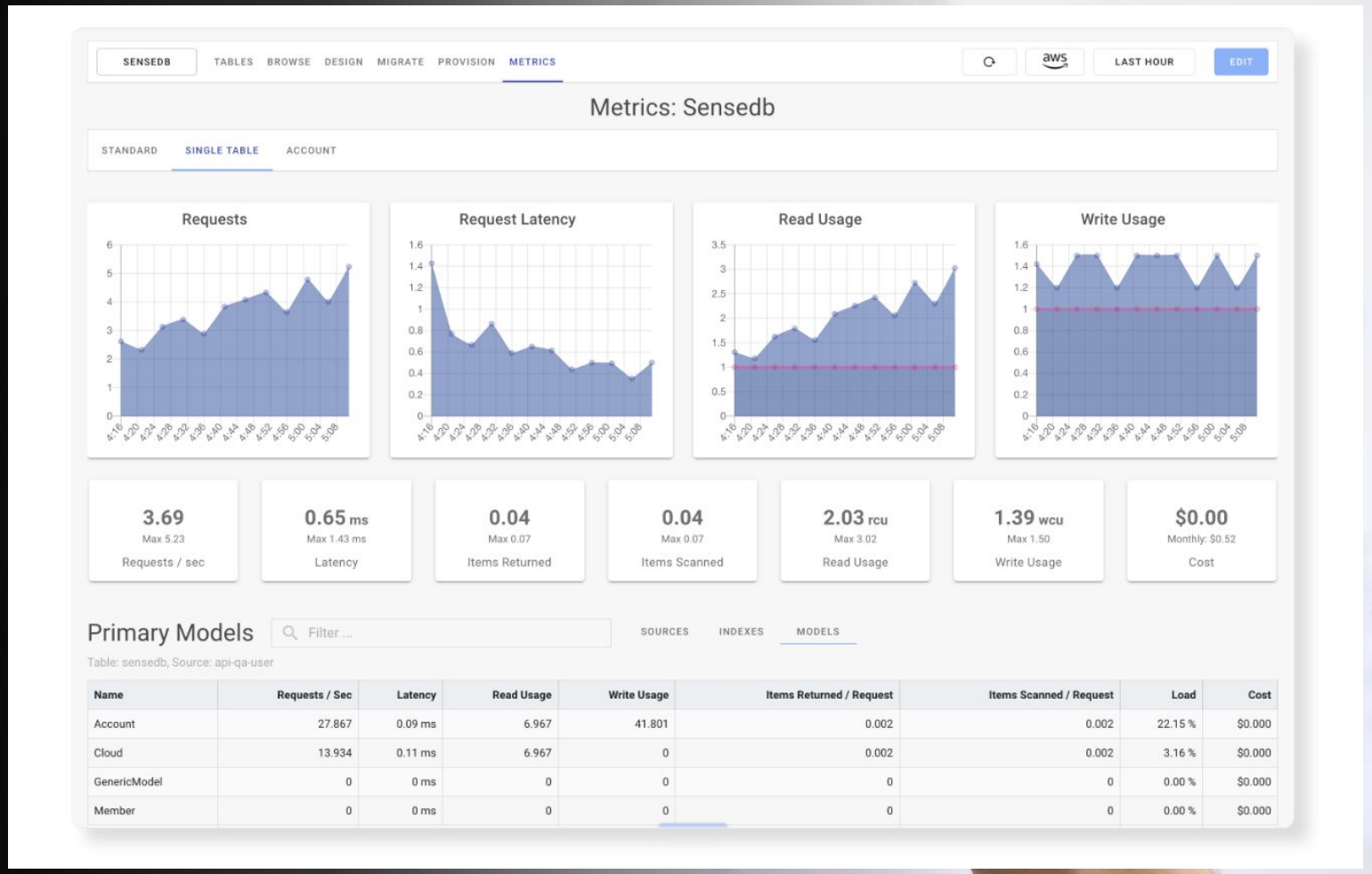
Step 11:

Check any readings when monitoring the Lambda application


EC2
S3
Elastic Beanstalk
VPC
RDS
IAM
DynamoDB
CloudWatch metrics Info
Filter by
Function
Lambda sends runtime metrics for your functions to Amazon CloudWatch. The metrics shown are an aggregate view of all function runtime activity. To view metrics for the unqualified or $LATEST resource, choose Filter by. To view metrics for a specific function version or alias, choose Aliases or Versions, select the alias or version, and then choose Monitor.
1h
3h
12h
1d
3d
1w
Custom
Invocations
Count
52
32.5
13
22:00
23:00
00:00
Invocations
Duration
Milliseconds
5.21
3.18
1.15
22:00
23:00
00:00
Duration minimum
Duration average
Duration maximum
Error count and success rate (%)
Count
No unit
1
0.5
0
100
99.5
99
22:00
23:00
00:00
Errors
Success rate (%)

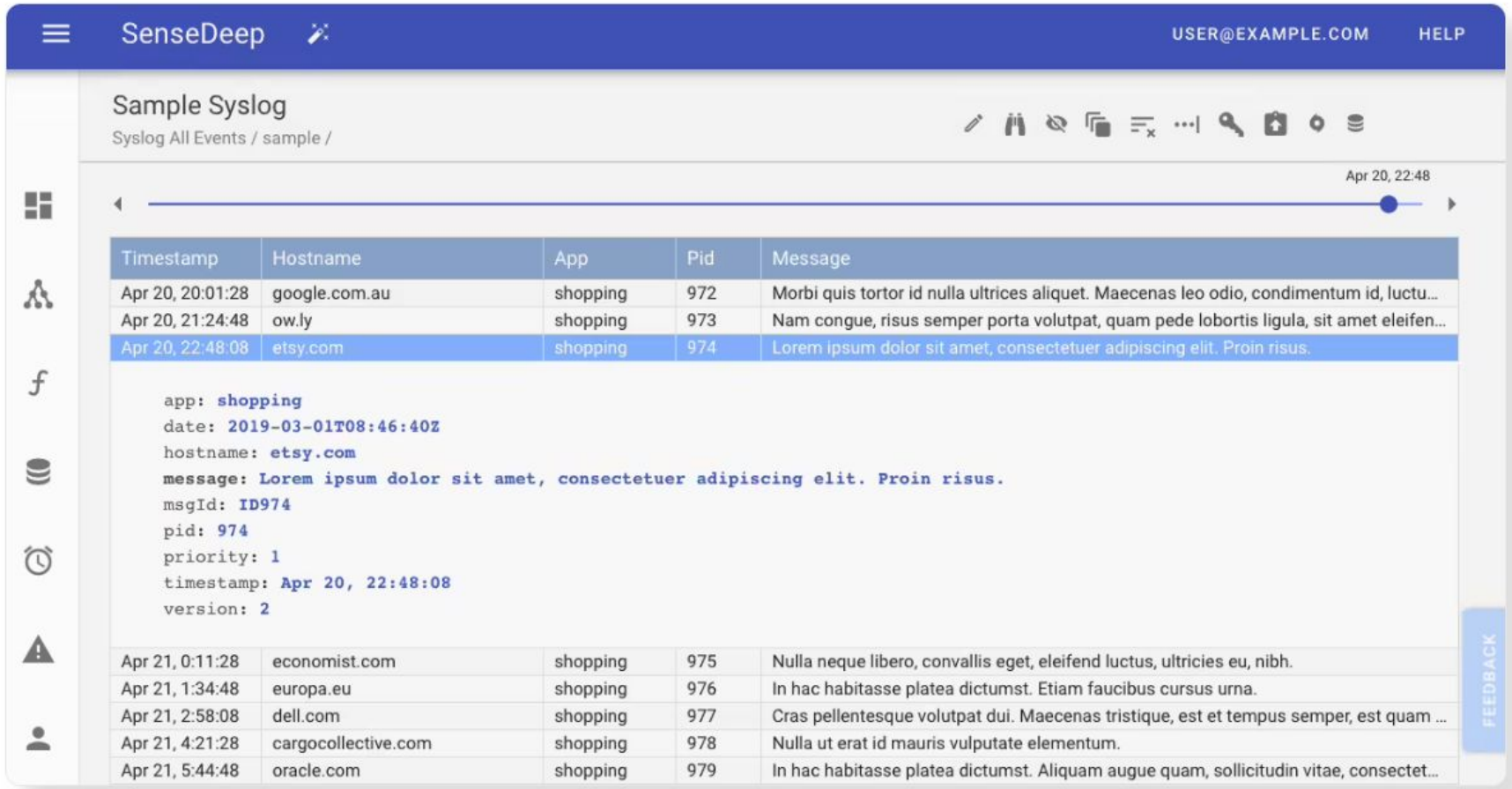
Step 12:
Access SenseDeep application

- 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



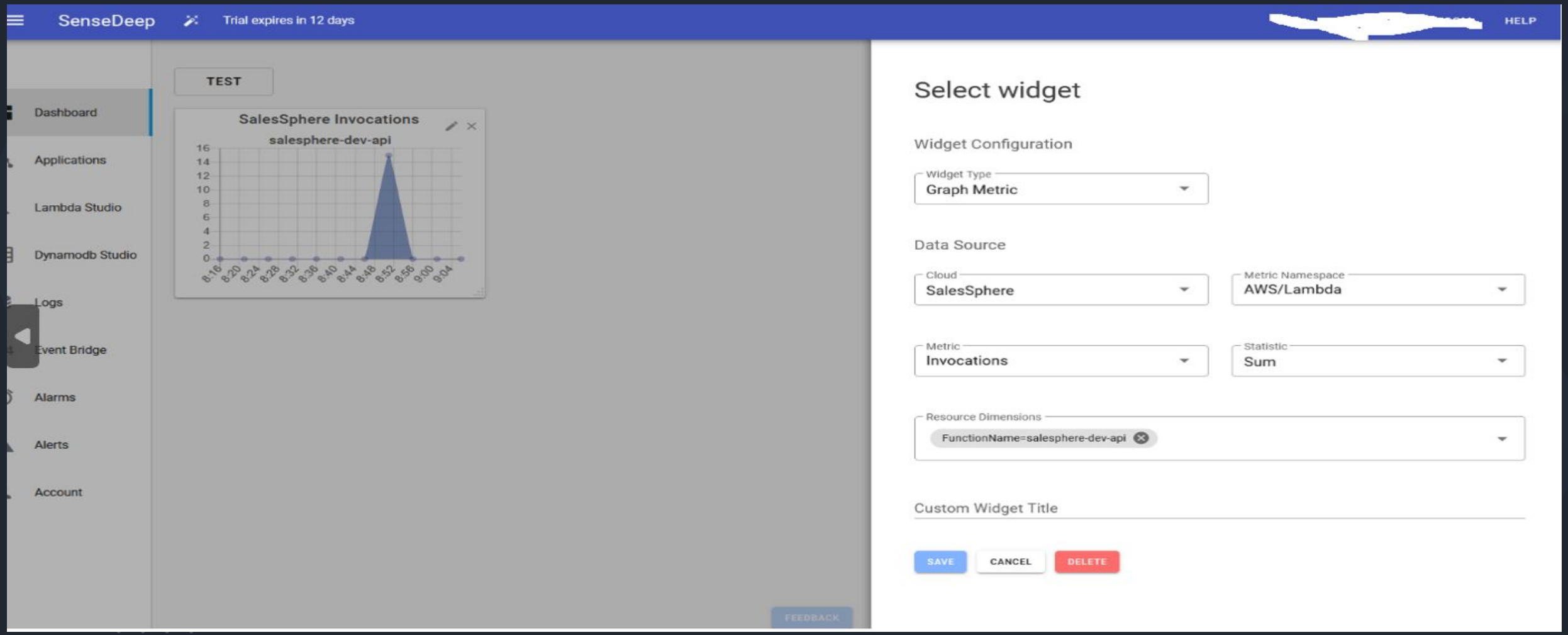
Application logging

- Access and login SenseDeep application
- 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



Step 13:

Design our own graphs on various metrics and various statistic types by extracting data from AWS Lambda.



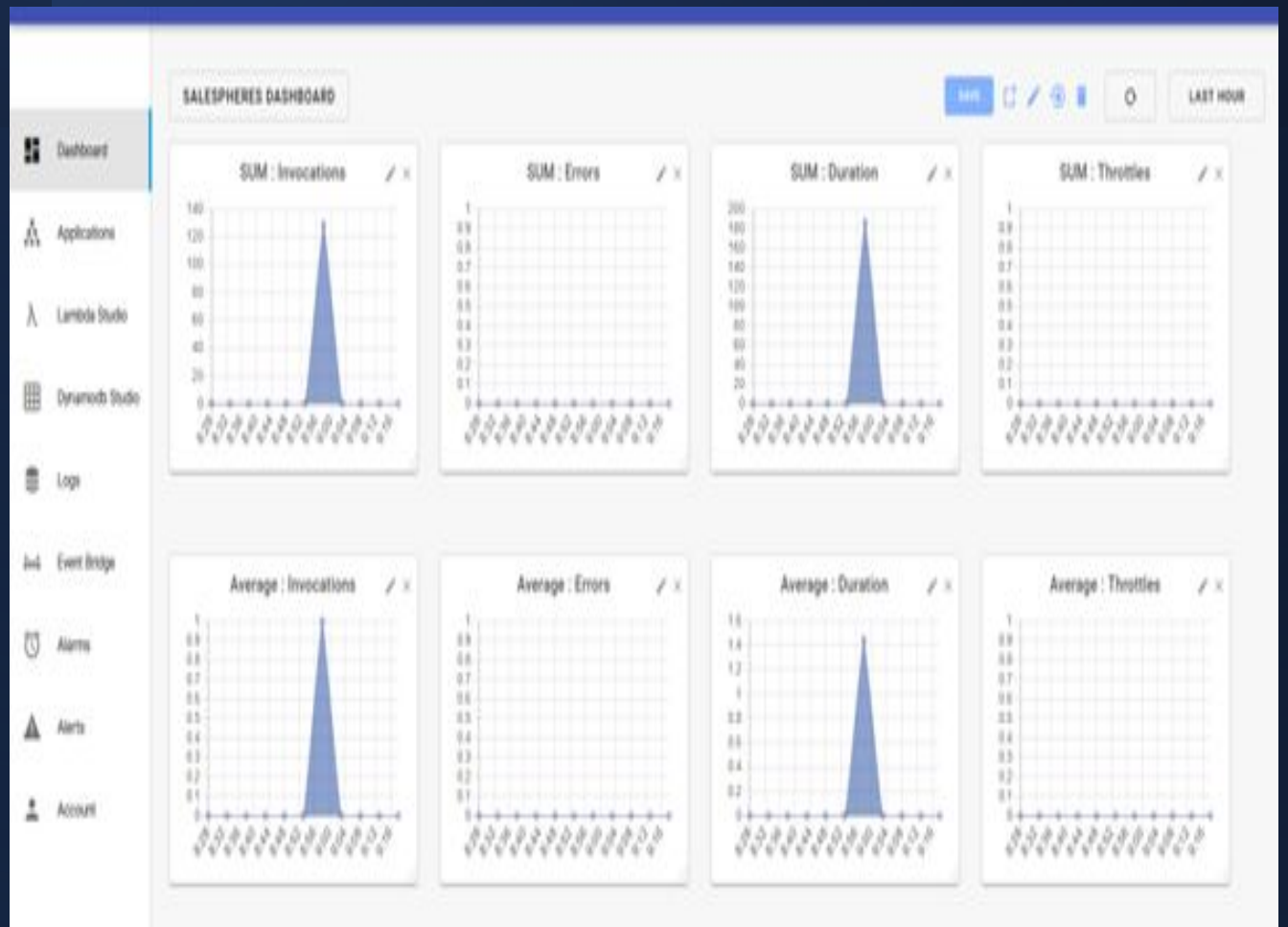
Completion on ready to view Centralize Dashboard

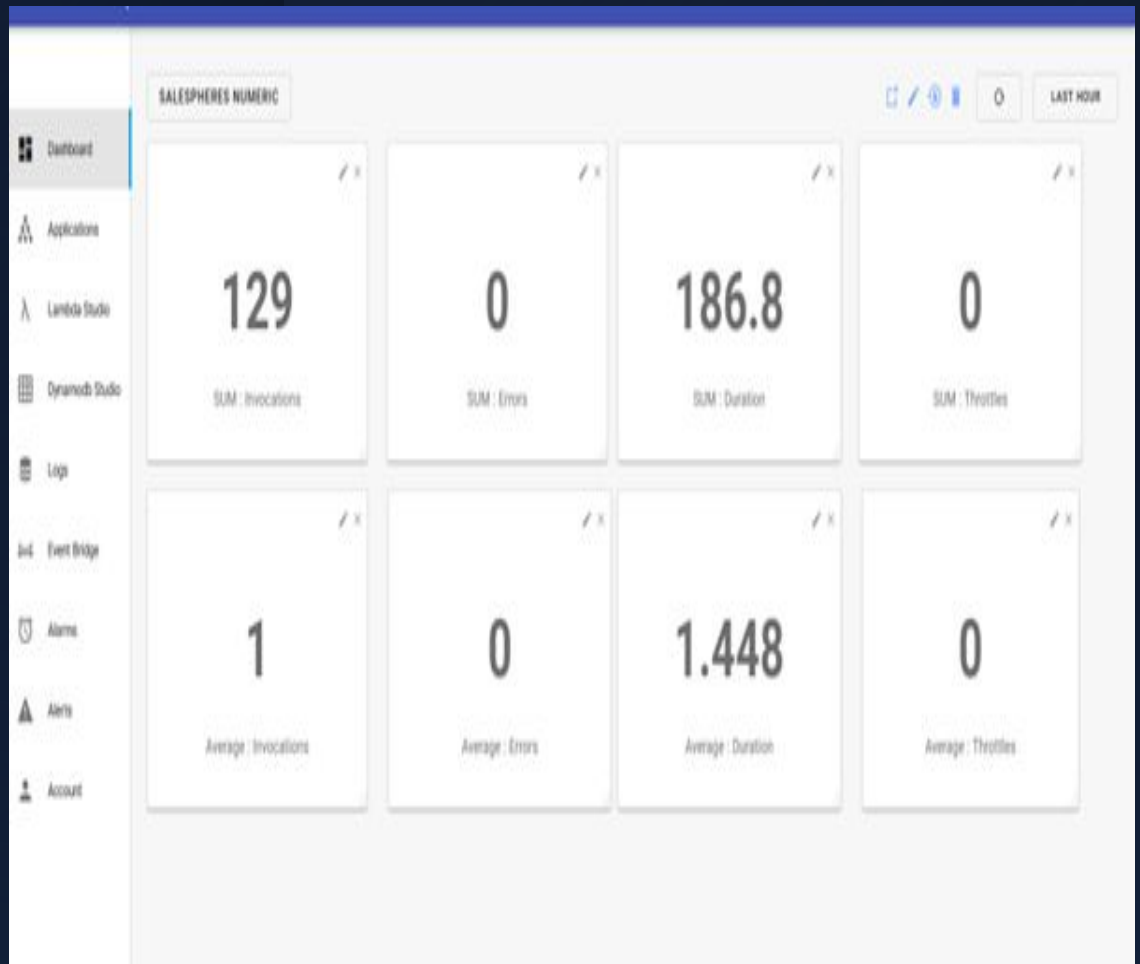
Next is the illustration of a graph widget based on hourly data of Sum and Average from salesphere-dev-api and SalesSphere-dev-error

Application Monitoring Dashboard

Step 14:
Completion on ready to view centralize Dashboard

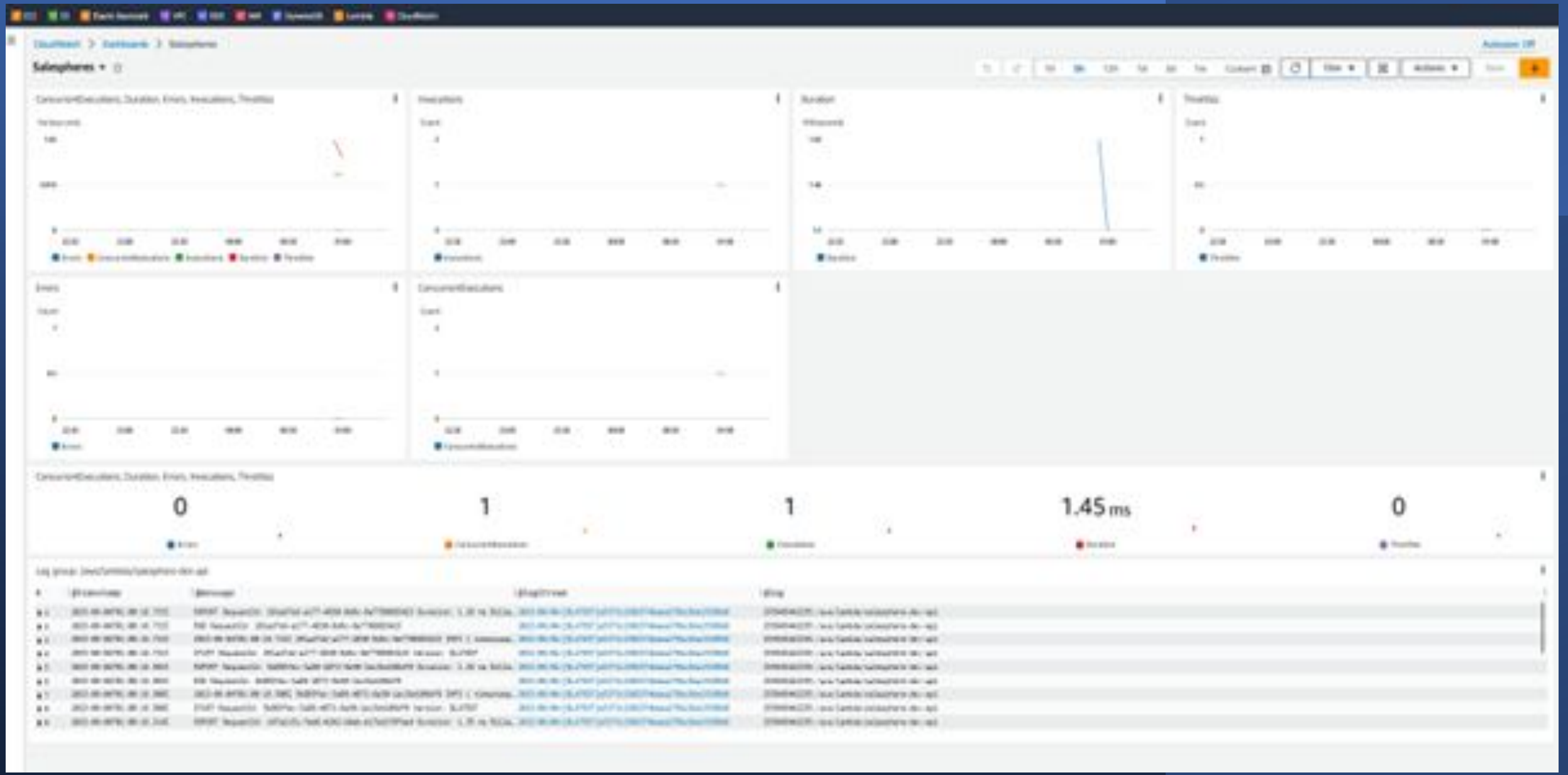
This is the illustration of a detail widget based on SUM and Average hourly on salesphere-dev-api and SalesSphere-dev-error.
SALESPHERES DASHBOARD





FAQ

- What if Sensedeep is not available or outage?
- 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


Salesphores
0
1
1
1.45 ms
0

Invocations, Duration, Errors , ConcurrentExecution, Throttles

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.

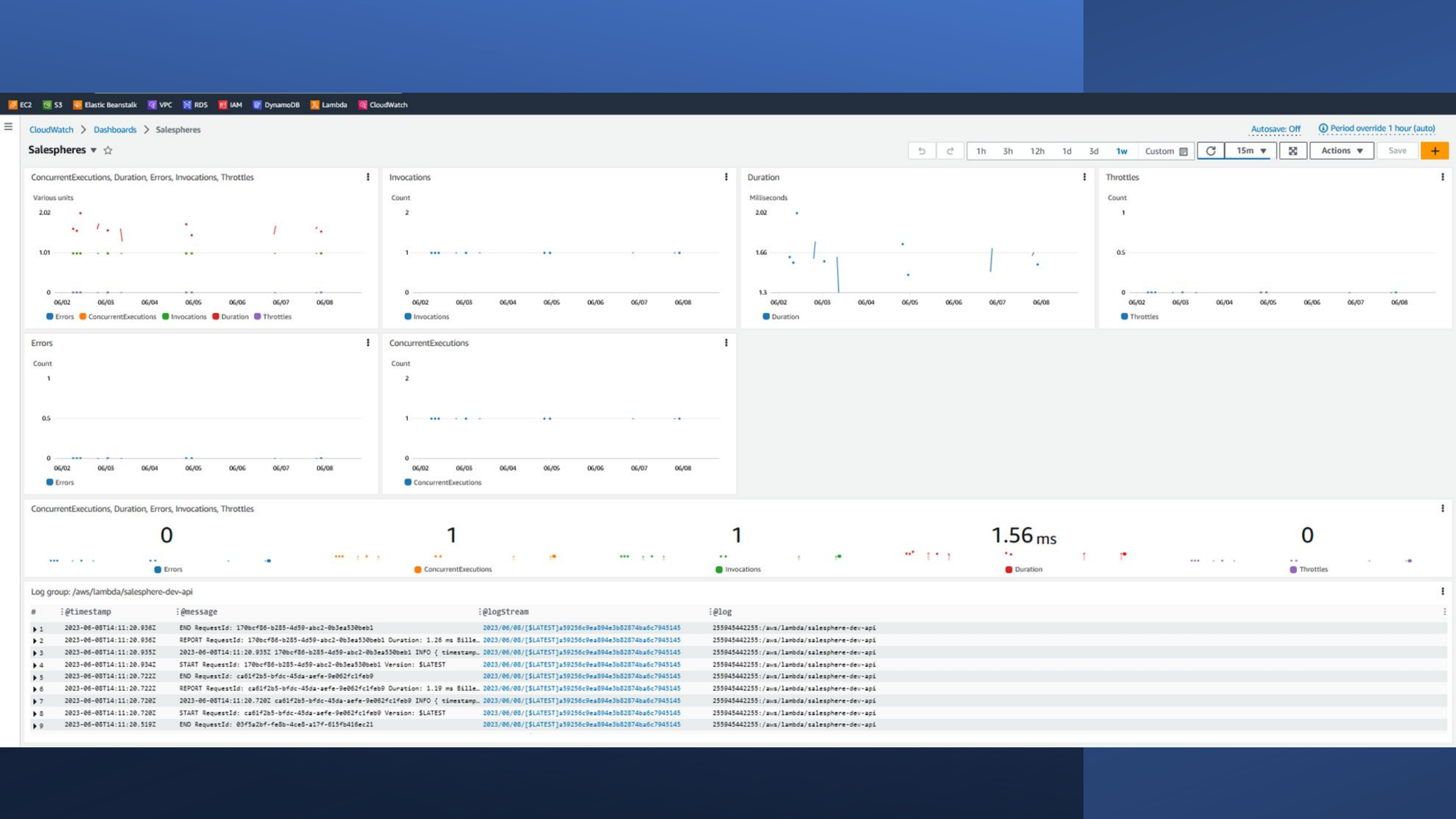
Errors - The number of invocations that result in a function error.

ConcurrentExecutions - 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.

Would like to get the logs in real-time. How would I access?

- You can access on the define Dashboard name


EC2 S3 Elastic Beanstalk VPC RDS IAM DynamoDB Lambda CloudWatch
CloudWatch > Dashboards > Salespheres
Autosave: Off
Period override 1 hour (auto)
Salespheres
1h 3h 12h 1d 3d 1w Custom
15m
Actions
Save
ConcurrentExecutions, Duration, Errors, Invocations, Throttles
Various units
2.02
1.01
0
06/02 06/03 06/04 06/05 06/06 06/07 06/08
Errors ConcurrentExecutions Invocations Duration Throttles
Invocations
Count
2
1
0
Invocations
Duration
Milliseconds
2.02
1.66
1.3
Duration
Throttles
Count
1
0.5
0
Throttles
Errors
Count
1
0.5
0
Errors
ConcurrentExecutions
Count
2
1
0
ConcurrentExecutions
ConcurrentExecutions, Duration, Errors, Invocations, Throttles
0
Errors
1
ConcurrentExecutions
1
Invocations
1.56 ms
Duration
0
Throttles
Log group: /aws/lambda/salesphere-dev-api
@timestamp @message @logStream @log
1 2023-06-08T14:11:20.936Z END RequestId: 170bcf86-b285-4d59-abc2-0b3ea530beb1 2023/06/08/[$LATEST]a59256c9ea894e3b82874ba6c7945145 255945442255:/aws/lambda/salesphere-dev-api
2 2023-06-08T14:11:20.936Z REPORT RequestId: 170bcf86-b285-4d59-abc2-0b3ea530beb1 Duration: 1.26 ms Bille… 2023/06/08/[$LATEST]a59256c9ea894e3b82874ba6c7945145 255945442255:/aws/lambda/salesphere-dev-api
3 2023-06-08T14:11:20.935Z 2023-06-08T14:11:20.935Z 170bcf86-b285-4d59-abc2-0b3ea530beb1 INFO { timestamp… 2023/06/08/[$LATEST]a59256c9ea894e3b82874ba6c7945145 255945442255:/aws/lambda/salesphere-dev-api
4 2023-06-08T14:11:20.934Z START RequestId: 170bcf86-b285-4d59-abc2-0b3ea530beb1 Version: $LATEST 2023/06/08/[$LATEST]a59256c9ea894e3b82874ba6c7945145 255945442255:/aws/lambda/salesphere-dev-api
5 2023-06-08T14:11:20.722Z END RequestId: ca61f2b5-bfdc-45da-aefe-9e062fc1feb9 2023/06/08/[$LATEST]a59256c9ea894e3b82874ba6c7945145 255945442255:/aws/lambda/salesphere-dev-api
6 2023-06-08T14:11:20.722Z REPORT RequestId: ca61f2b5-bfdc-45da-aefe-9e062fc1feb9 Duration: 1.19 ms Bille… 2023/06/08/[$LATEST]a59256c9ea894e3b82874ba6c7945145 255945442255:/aws/lambda/salesphere-dev-api
7 2023-06-08T14:11:20.720Z 2023-06-08T14:11:20.720Z ca61f2b5-bfdc-45da-aefe-9e062fc1feb9 INFO { timestamp… 2023/06/08/[$LATEST]a59256c9ea894e3b82874ba6c7945145 255945442255:/aws/lambda/salesphere-dev-api
8 2023-06-08T14:11:20.720Z START RequestId: ca61f2b5-bfdc-45da-aefe-9e062fc1feb9 Version: $LATEST 2023/06/08/[$LATEST]a59256c9ea894e3b82874ba6c7945145 255945442255:/aws/lambda/salesphere-dev-api
9 2023-06-08T14:11:20.519Z END RequestId: 03f5a2bf-fe8b-4ce8-a17f-615fb416ec21 2023/06/08/[$LATEST]a59256c9ea894e3b82874ba6c7945145 255945442255:/aws/lambda/salesphere-dev-api

Future plan?

- After a period of evaluation, we are glad to propose the positive results updates to our management. We are glad the project SaleSpheres version 1.0 ready to launch in production
- Next plan of version 2.0 will be on the following,

a) Area of API Gateway integration

b) Consideration the enhancement the features on the SenseDeep application with logs

c) Monitoring on other services like EC2.


Thank You!