CS-524 Course Project

CS 524 A - Introduction to Cloud Computing

Spring 2022

By,

Omkar Sinha

CWID: 10468301

Dept. Computer Science

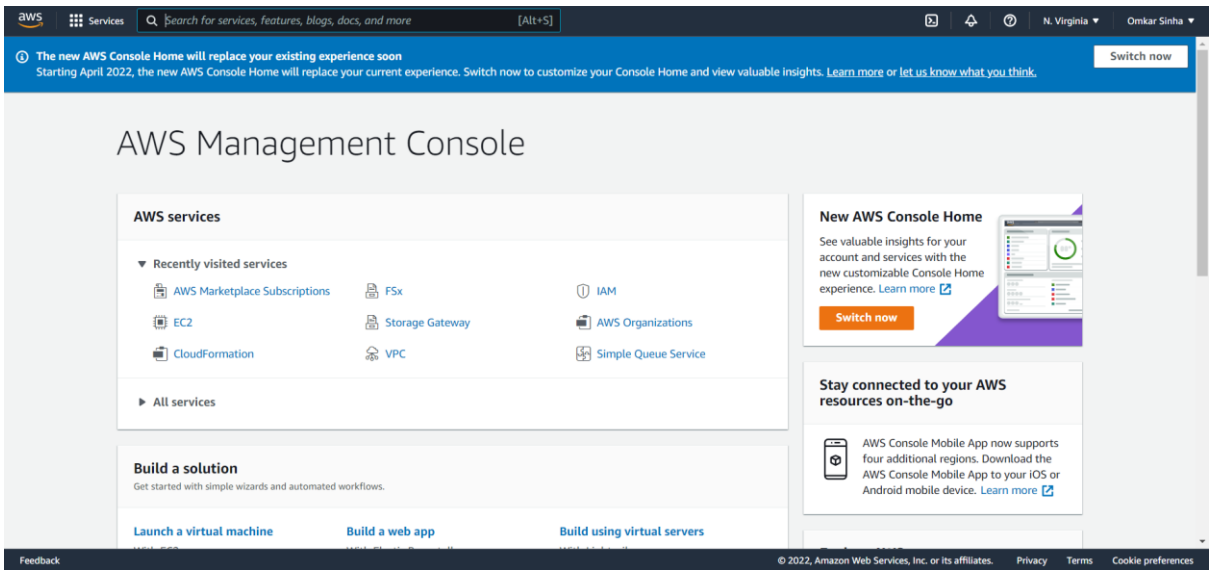
Stevens Institute of Technology

Under Guidance of,

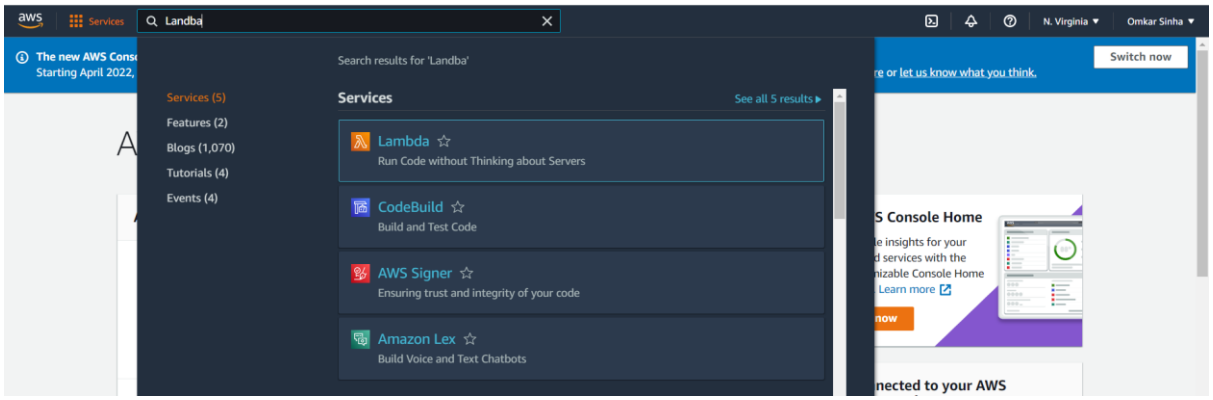
Prof. Dr. Igor Faynberg

PART A: AWS

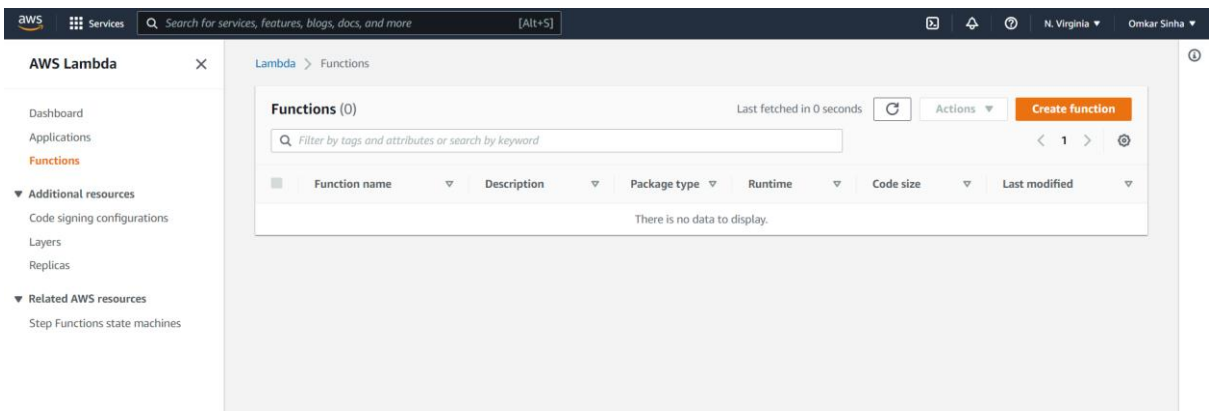
1) Firstly, we login to our AWS Account, as below:



2) We open the services and choose Lambda:

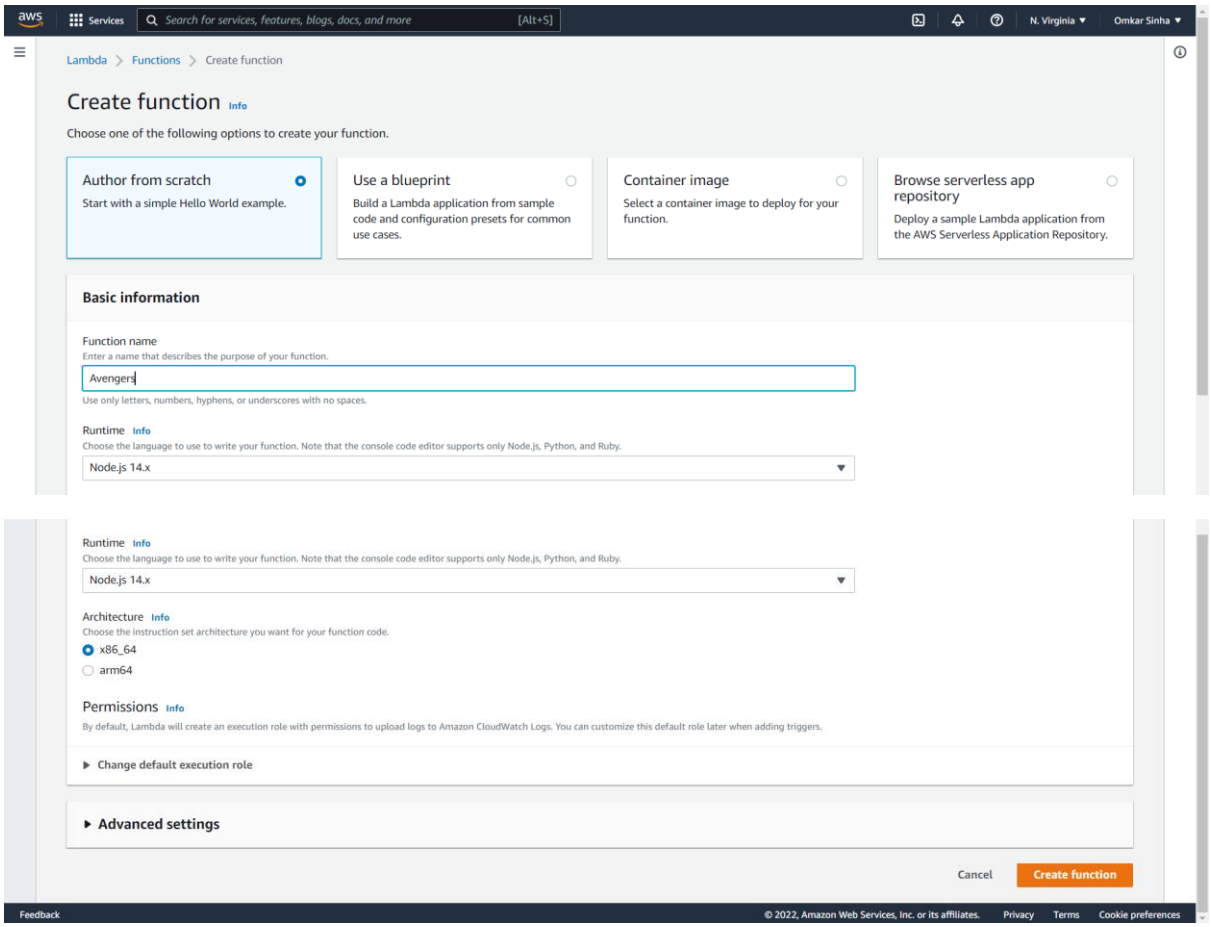


3) We see the AWS Lambda page (currently no functions are created) :-

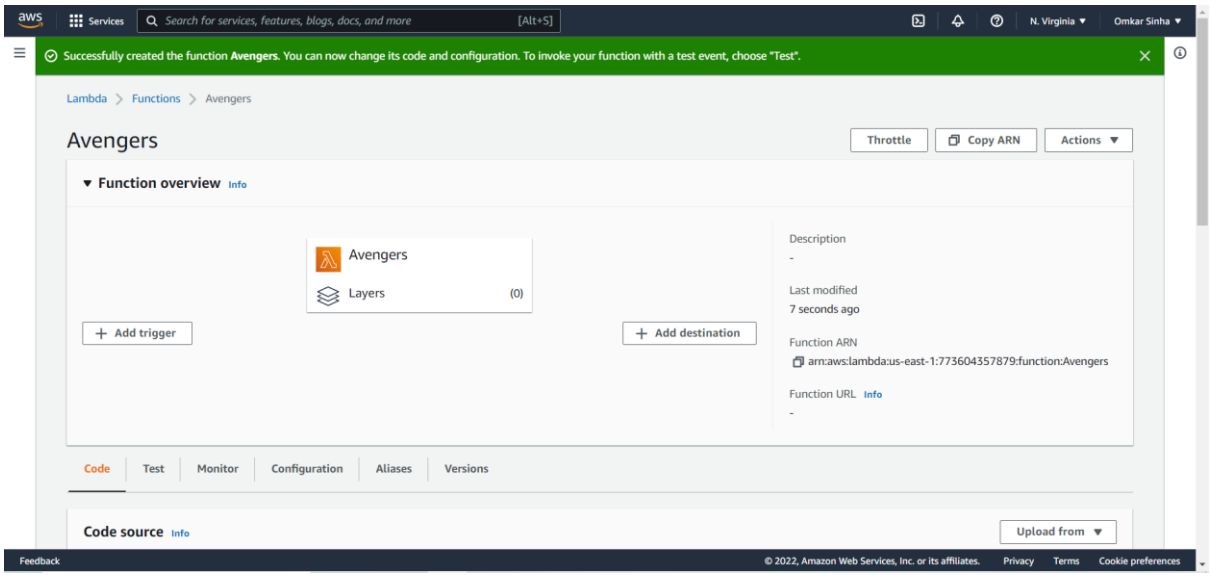


4) We choose “create function” to create a new function “Avengers”:

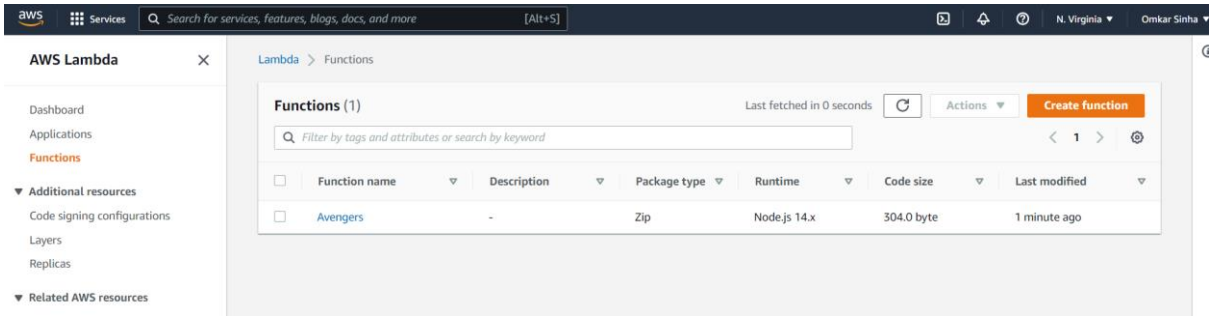
(i) We put the function name as “Avengers”, keeping other details same:



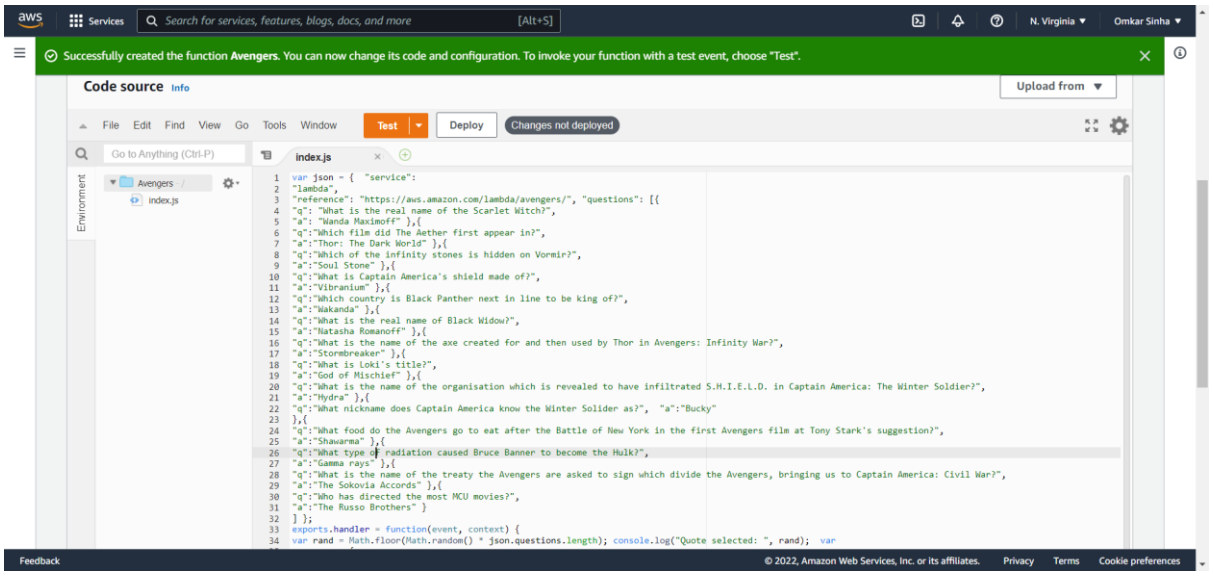
5) We see that “Avengers” Lambda function is created:



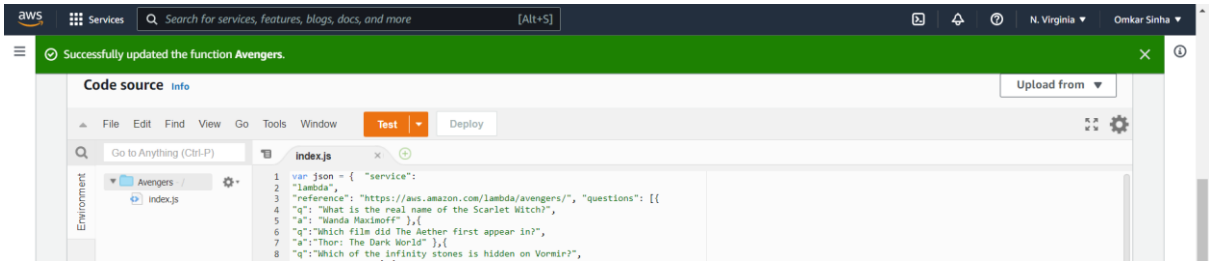
We see one function listed in Lambda -> Functions:



6) Now we proceed to add our given code source to the “Avengers” function. We copy and paste the code into the source code window as follows:

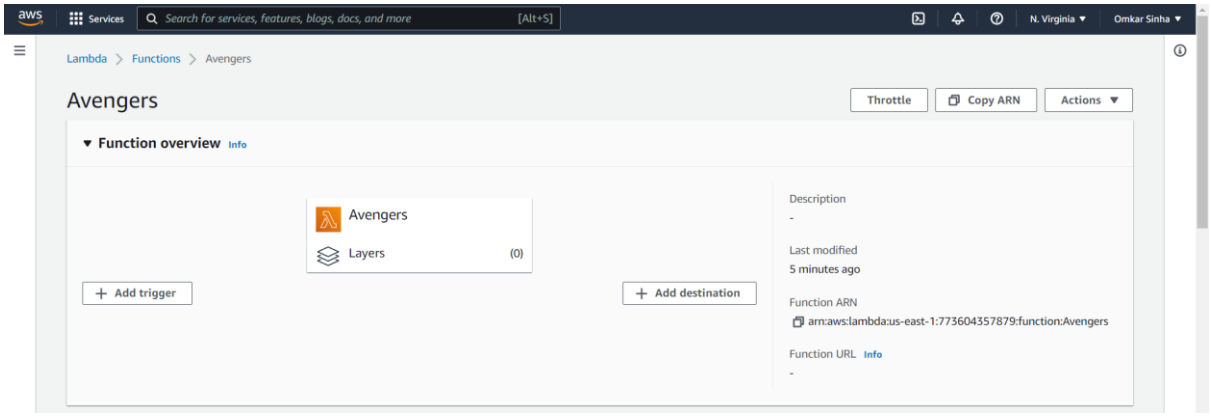


After adding the code and making sure that syntax is proper, we choose “Deploy”:

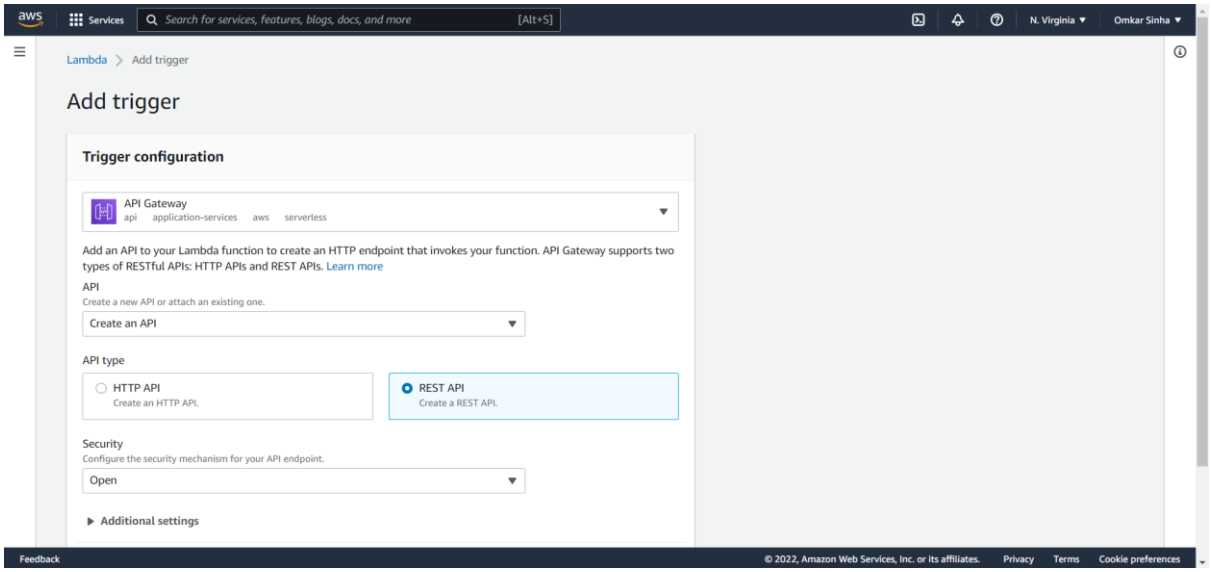


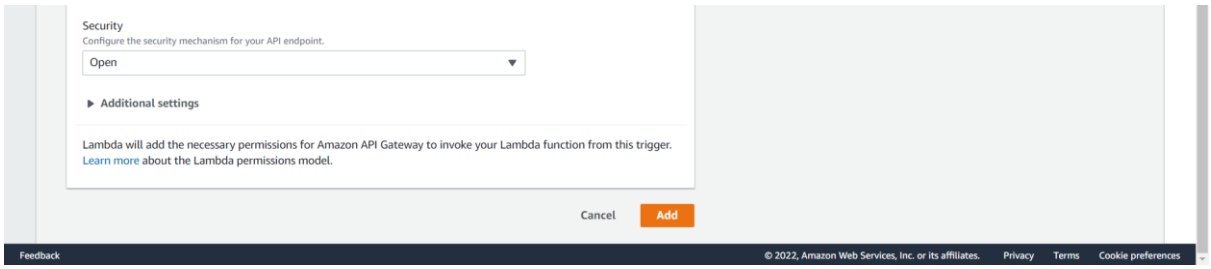
7) Now we proceed to add trigger to our Lambda function.

(i) We select “Add Trigger” from the Avengers function home-screen as shown below and choose “API Gateway” from the drop-down menu:

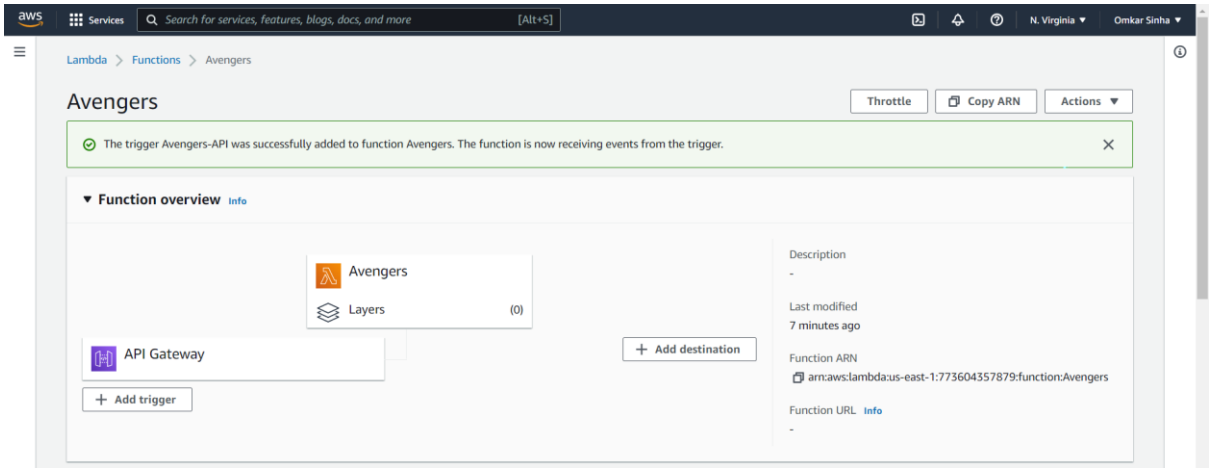


(ii) We choose “Rest API” as the API type and security as “Open”, leaving other details unchanged, and then select “Add”:

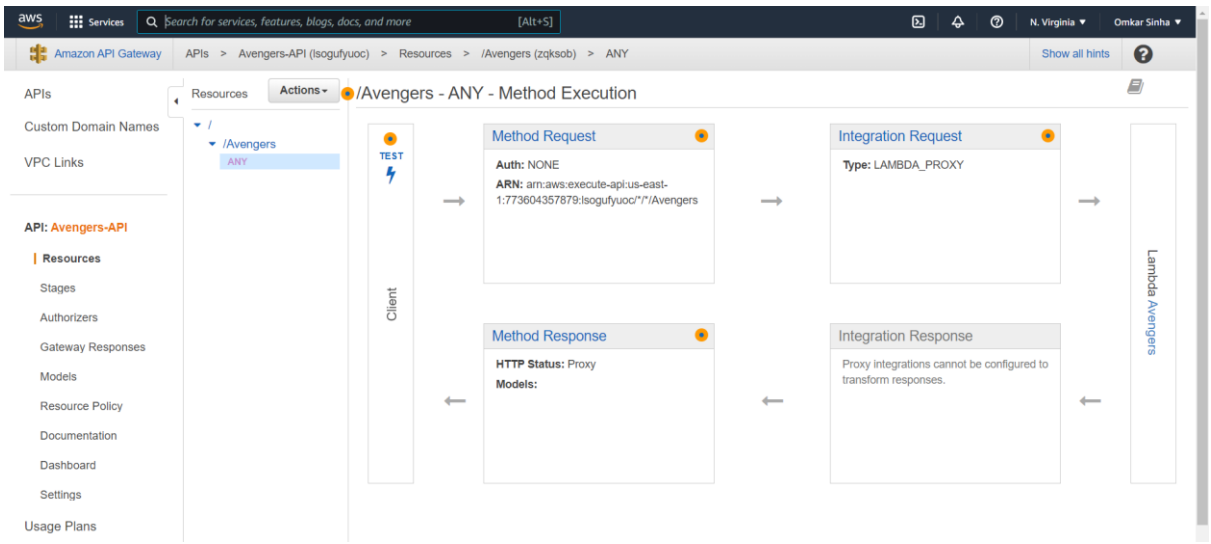




(iii) We see a prompt message declaring that the trigger is successfully created:

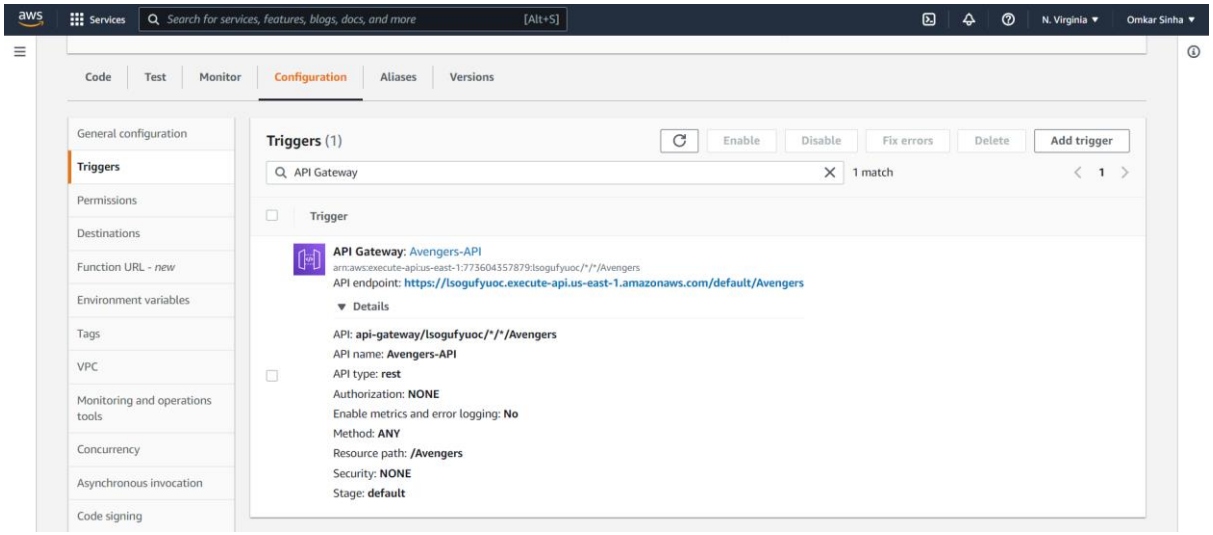


(iv) We can view the method execution for Avengers function in the API gateway as follows:

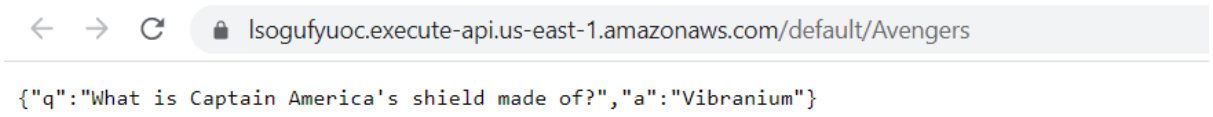


8) Now, we proceed to test our Lambda function by copying our API Gateway endpoint URL and pasting it into the browser:

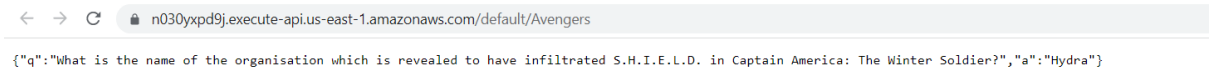
(i) We firstly select the API Gateway from above Avengers function overview, and are presented with the following trigger details:



(ii) We now test our Lambda function by copying the API endpoint URL from above and paste it into the browser. We see the following question and answer pair in JSON format from marvel series that we included in the code. This proves our function is working:



(iii) We again run the same URL and get another random question and answer:



9) We check CloudWatch logs now after running the URL first time:

(i) We choose “Monitor” from the panel as shown below. Then we check the Metrics toggle. Here we see the following metrics:

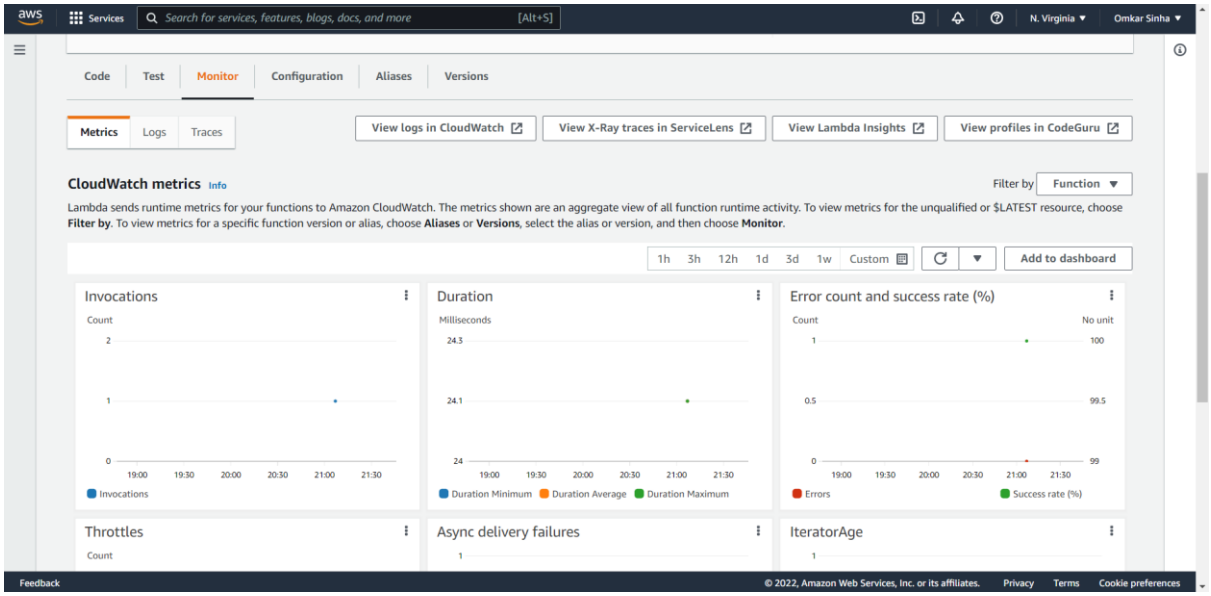
a) **Invocations:** We see the 1 invocation shortly after 9PM

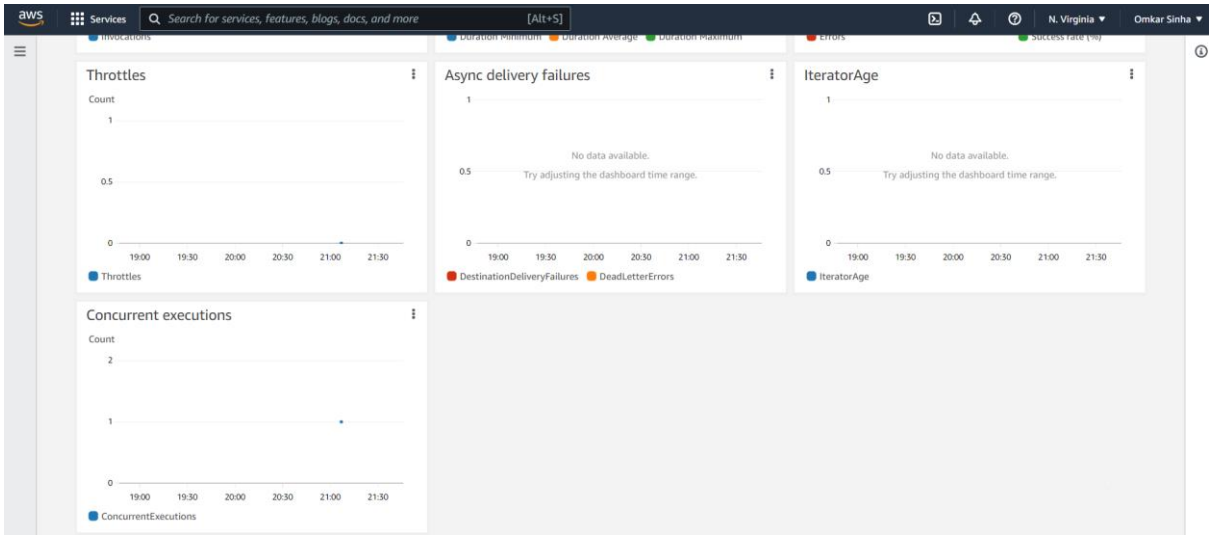
b) **Duration:** In duration graph we find that our invocations are plotted against time. We notice that our invocation took 24.1 milliseconds.

c) **Error Count and success rate:** We see that our success rate is 100% and error rate 0%, for the time interval shortly after 9PM – as represented by green and red dots, respectively.

d) **Concurrent execution:** Concurrent execution has a count of 1 because we had only 1 invocation.

Other metrics like **Throttles, Async delivery failures** and **IteratorAge** have no data.

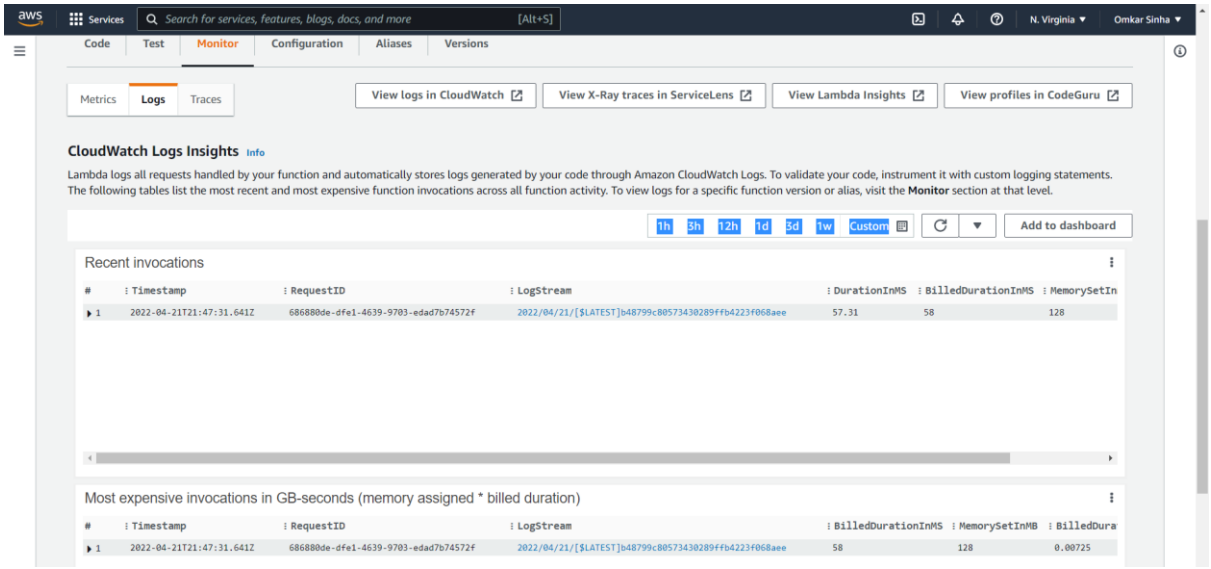




(ii) Now, we choose the Log toggle and see the following details(after our second execution at 21:47):

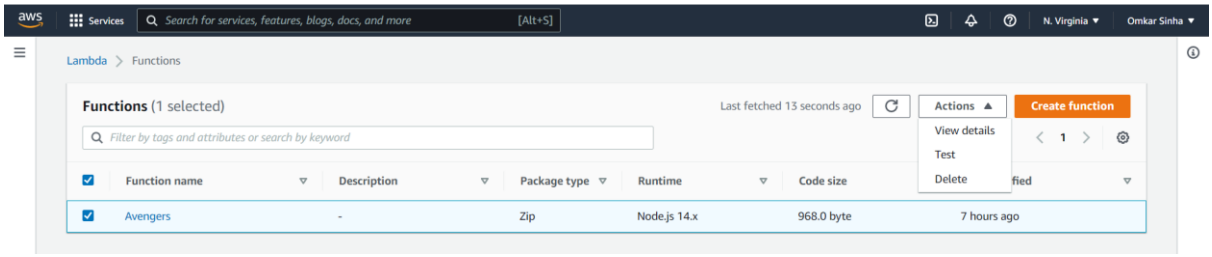
a) **Recent invocations:** We see details of our latest invocation including its timestamp, RequestID, log stream, duration, etc.

b) **Most Expensive invocation**: This gives billing details for invocation starting with most expensive. Our 2nd invocation is the most expensive in this case and its timestamp, request ID, logstream, billed duration, memory used(in MB) and other details are given.

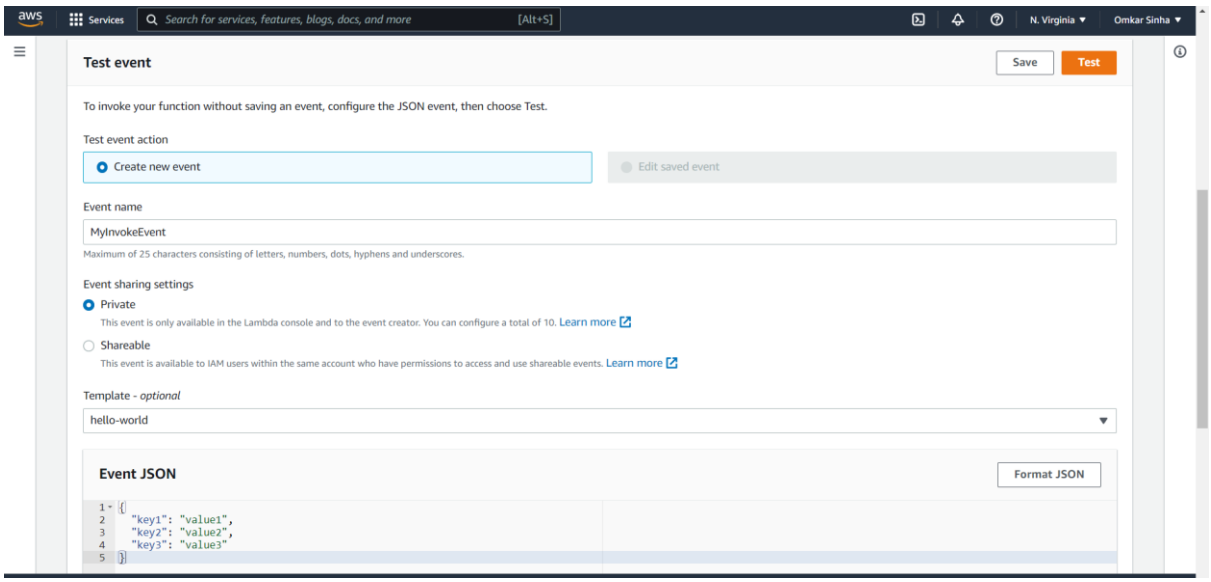


10) Now, we invoke the Lambda function and view CloudWatch logs for this function.

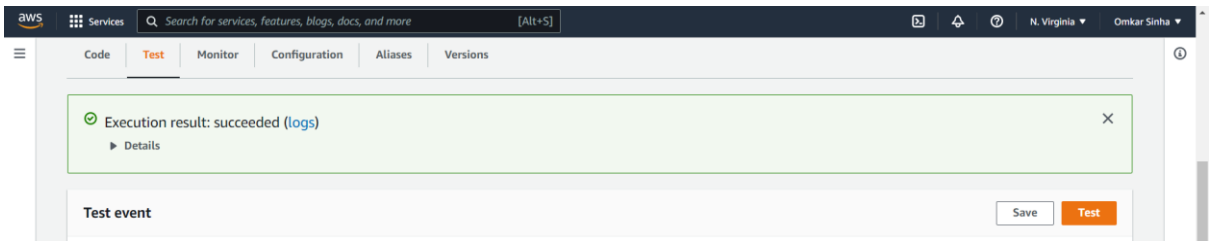
(i) Firstly, we need to create a test event. We choose “test event” from Actions as follows:



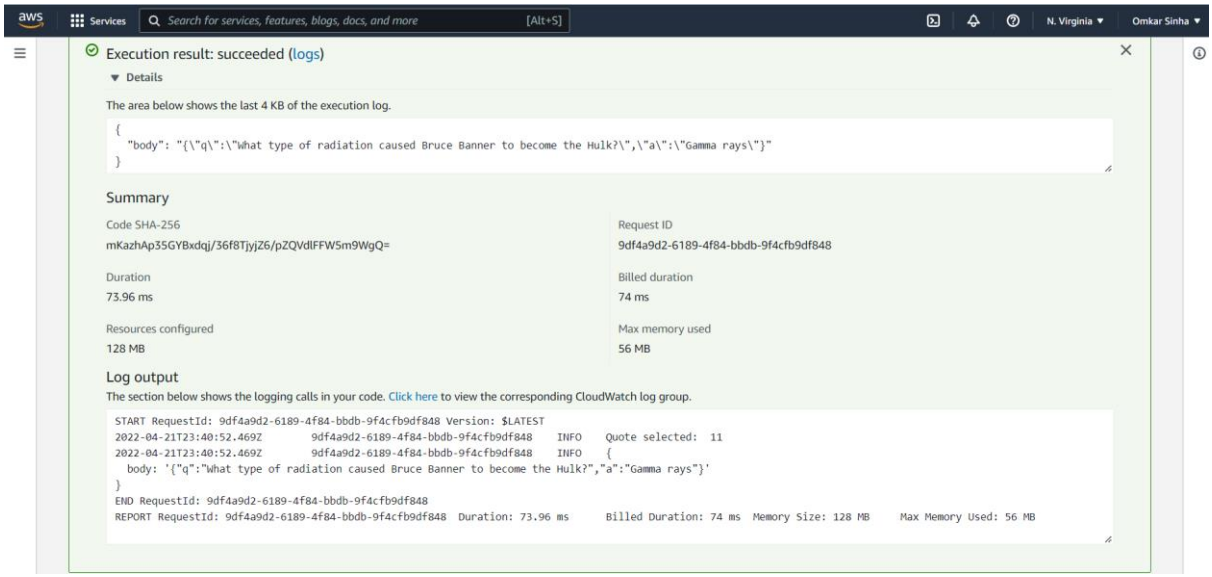
(ii) We choose “Create new event” and name it “MyInvokeEvent”, while keeping rest of the details unchanged:



(iii) Then we select “Test”. We see the following prompt announcing “Execution result: succeeded”.

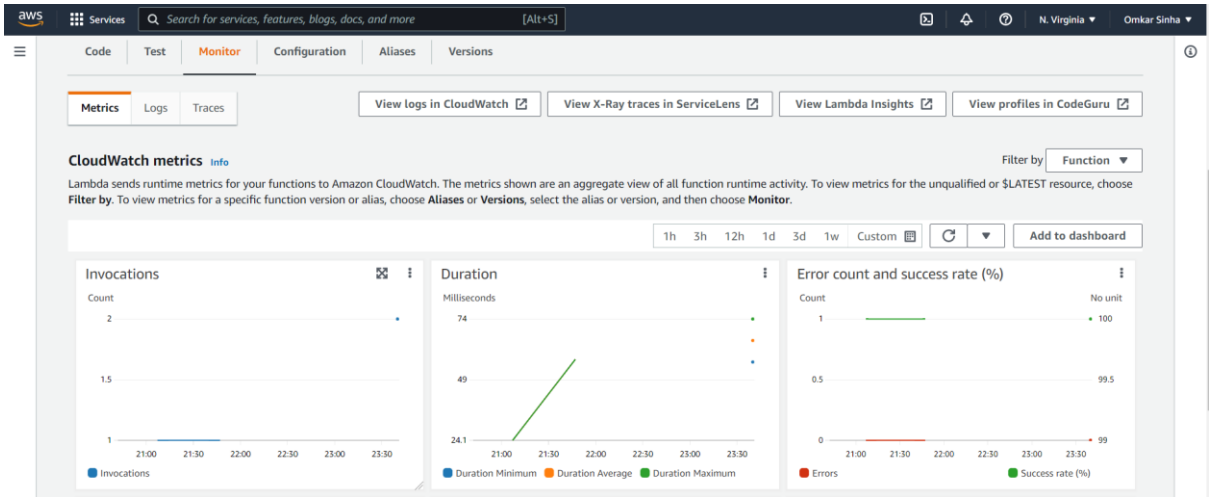


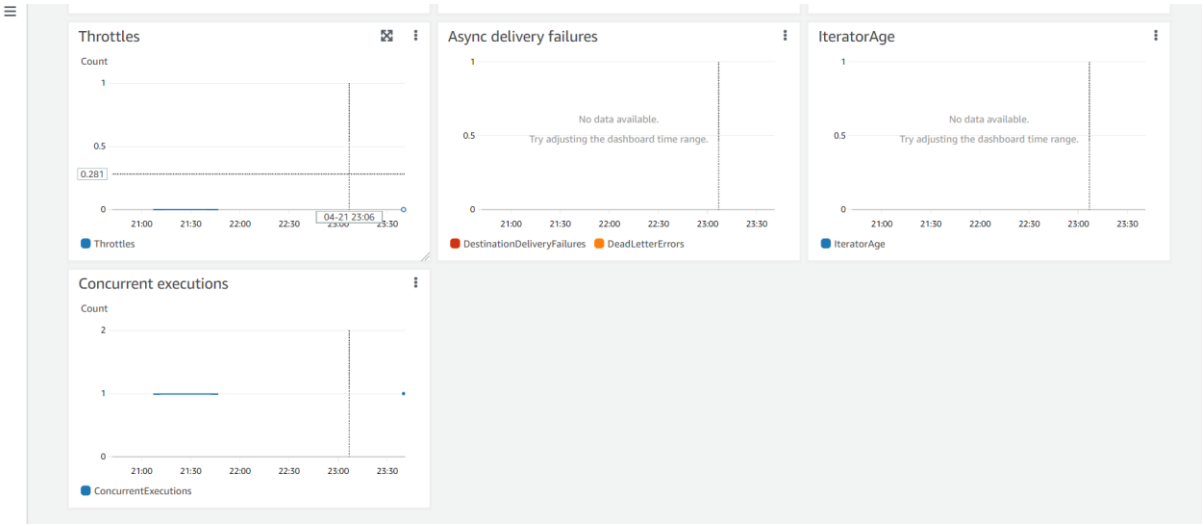
(iv) We expand the details to see the following. The body of the log output consist of the question-answer pair related to the marvel series: (We execute the function twice)



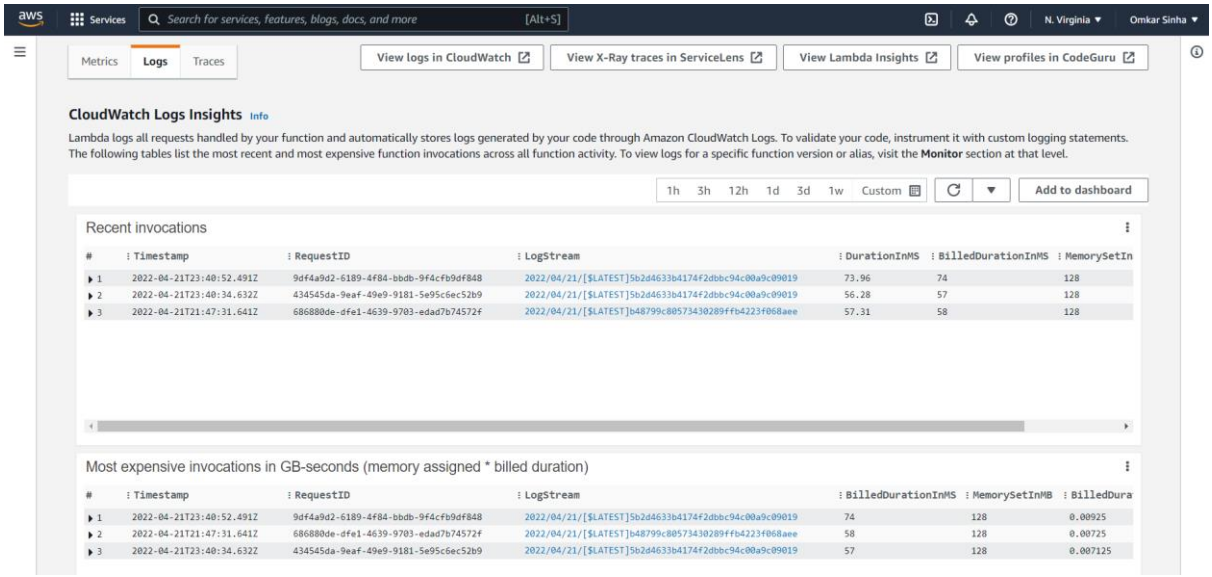
11) We view the cloudWatch details.

(i) We see the following metrics as follows for our invocations:

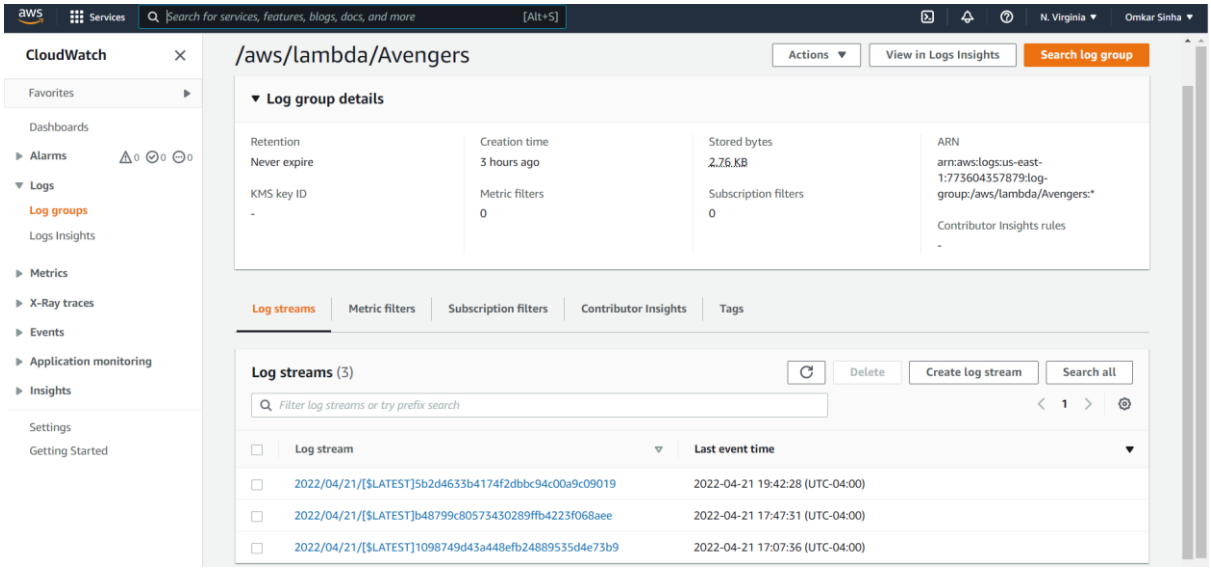




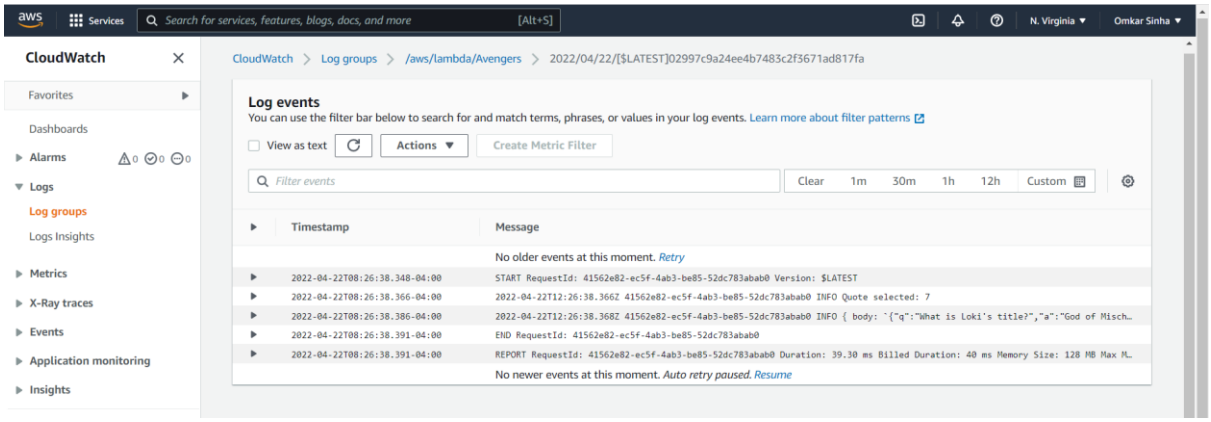
(ii) We also find the following logs



(iii) We open CloudWatch and find following log details(here local time-zones are shown):



(iv) We click on the log streams to view log events. We can see details of timestamps, ID, start and end time and all the information sent of the requests made using our API:

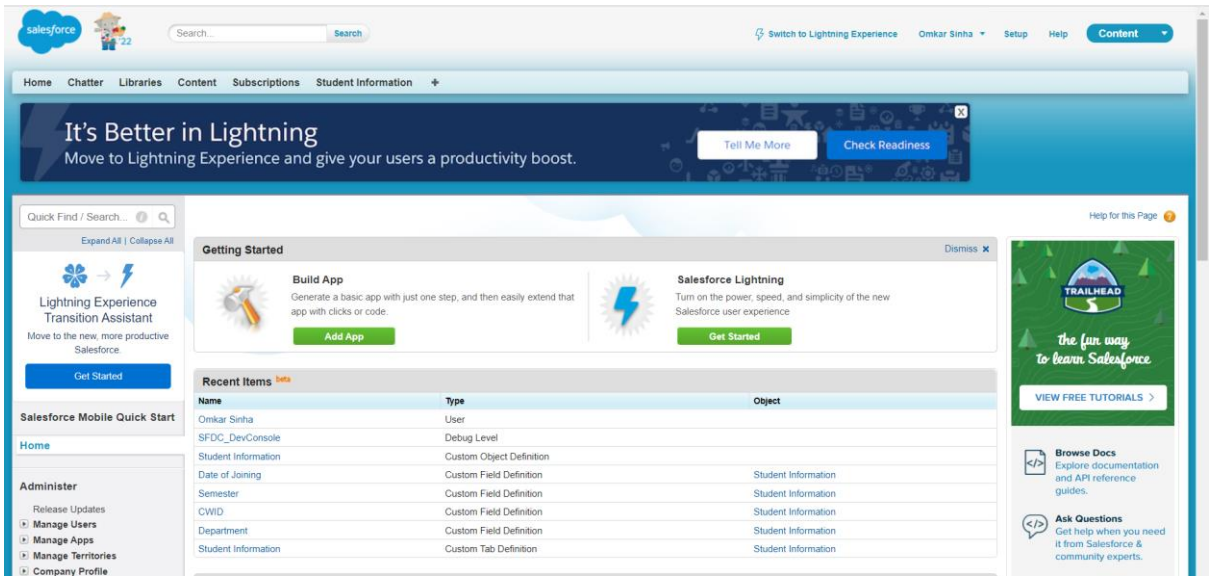


12) **Conclusion of the report:**

We can conclude from this exercise that the microservices architecture creates a neat division of responsibility between various services. The Lambda acts as serverless computing platform where we upload our code and attach an API gateway as the trigger. We create this API gateway as a REST API to both receive and serve up json inputs/outputs upon HTTP call. The output json follows the REST model. This is also a strong demonstration of AWS Lambda's serverless platform which provides a clear separation between the infrastructure and applications that run on the platform.

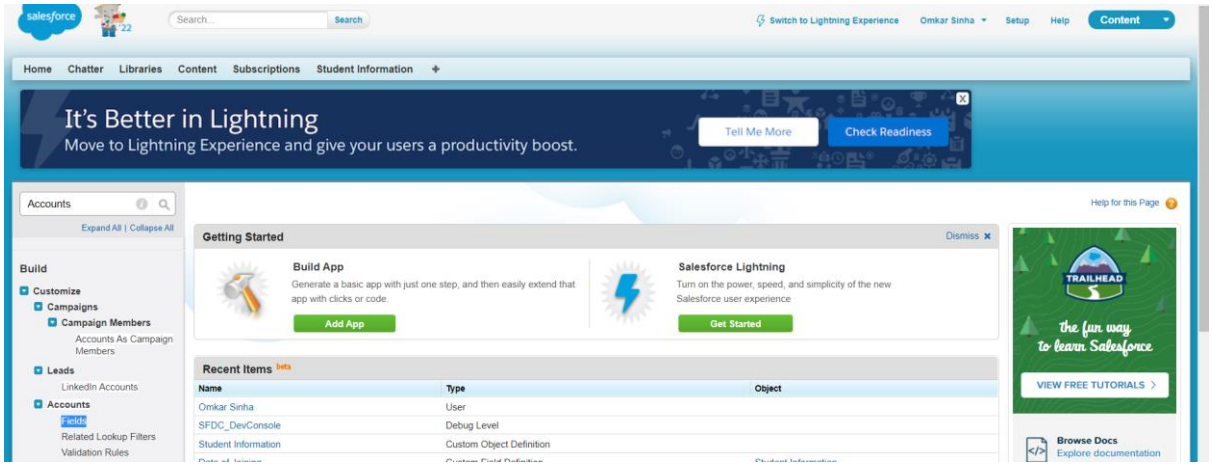
PART B: Salesforce

1) We login to our Salesforce account and go to setup:

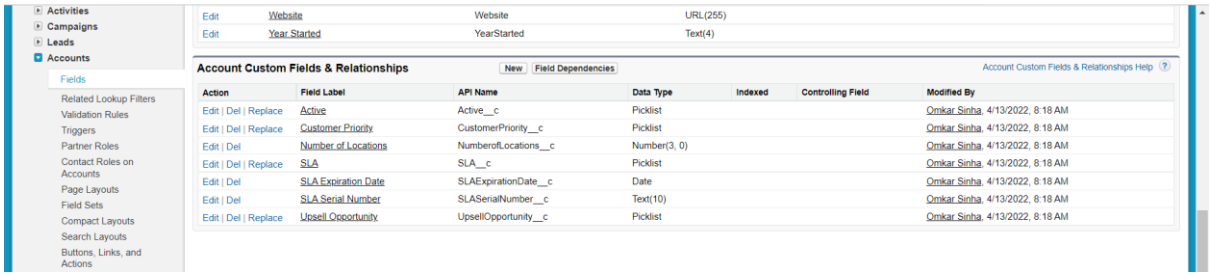


2) Creating custom field on Account Object:

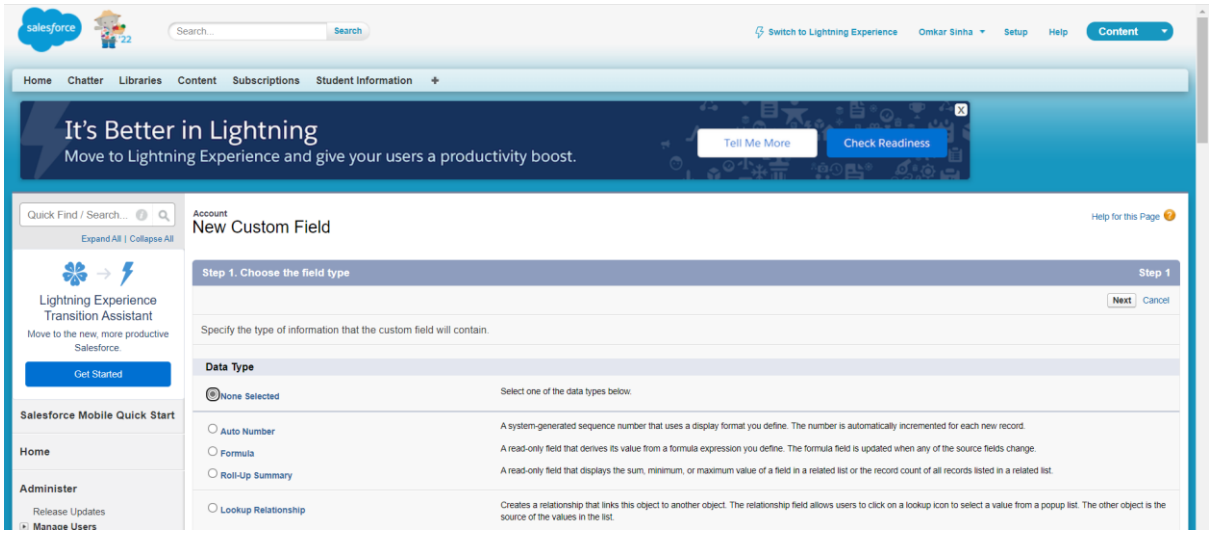
(i) Now, we will create custom field name: Field Update, on the Account object. We search “Accounts” in quick find bar and go to fields. We get all fields:



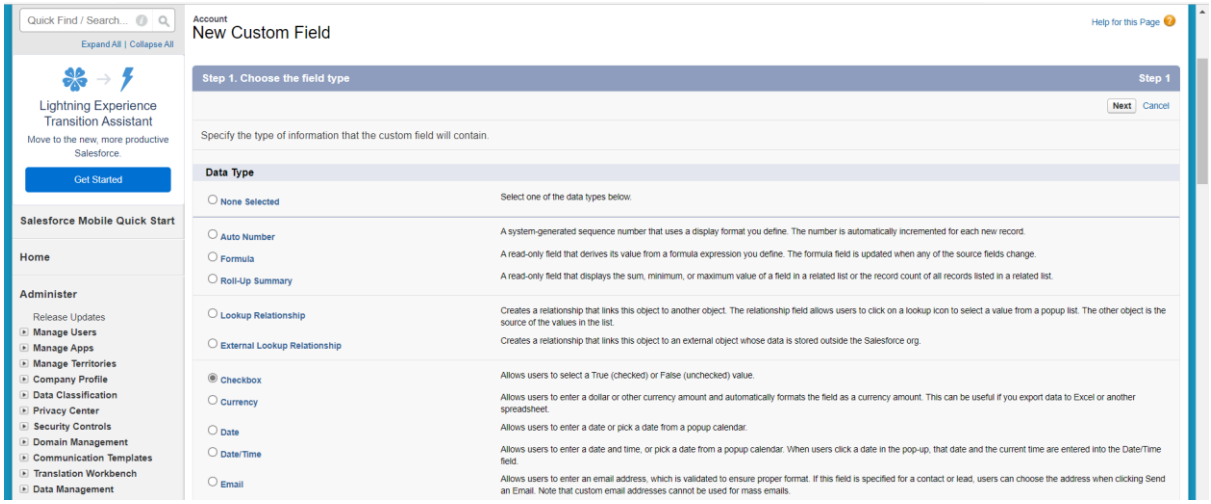
We scroll down to “Account Custom Fields & Relationships” and click on “New”:



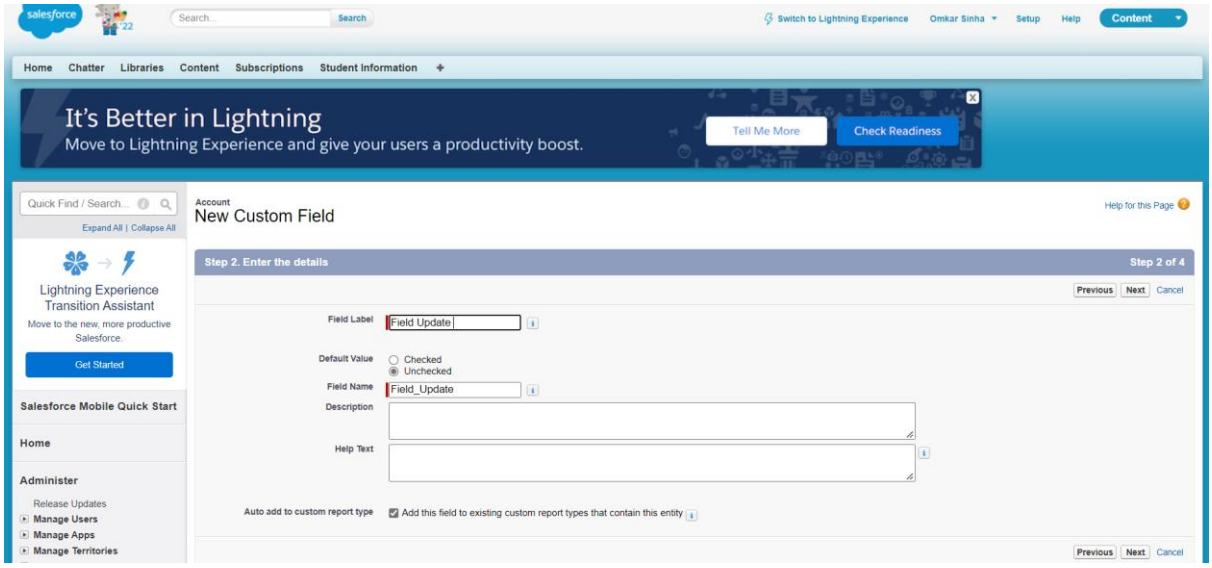
(ii) We find dialog screen for new custom field details:



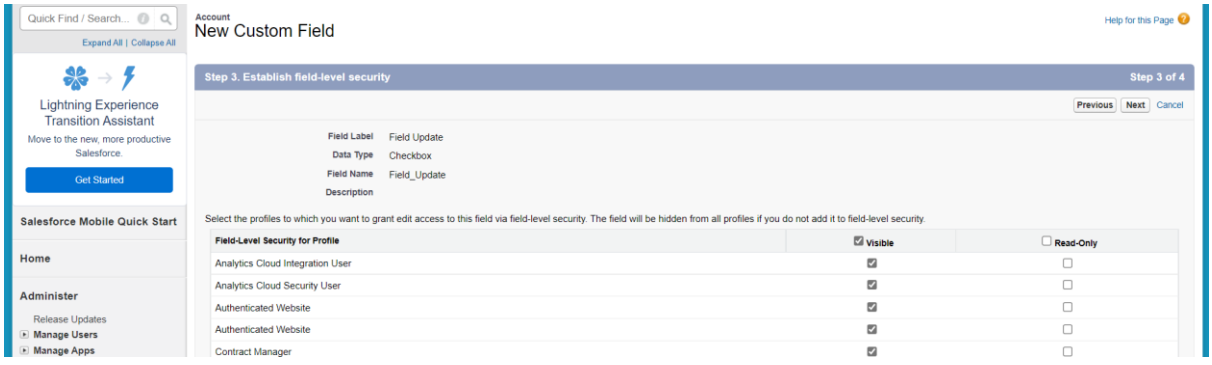
First is data type. We choose checkbox, and go to next:



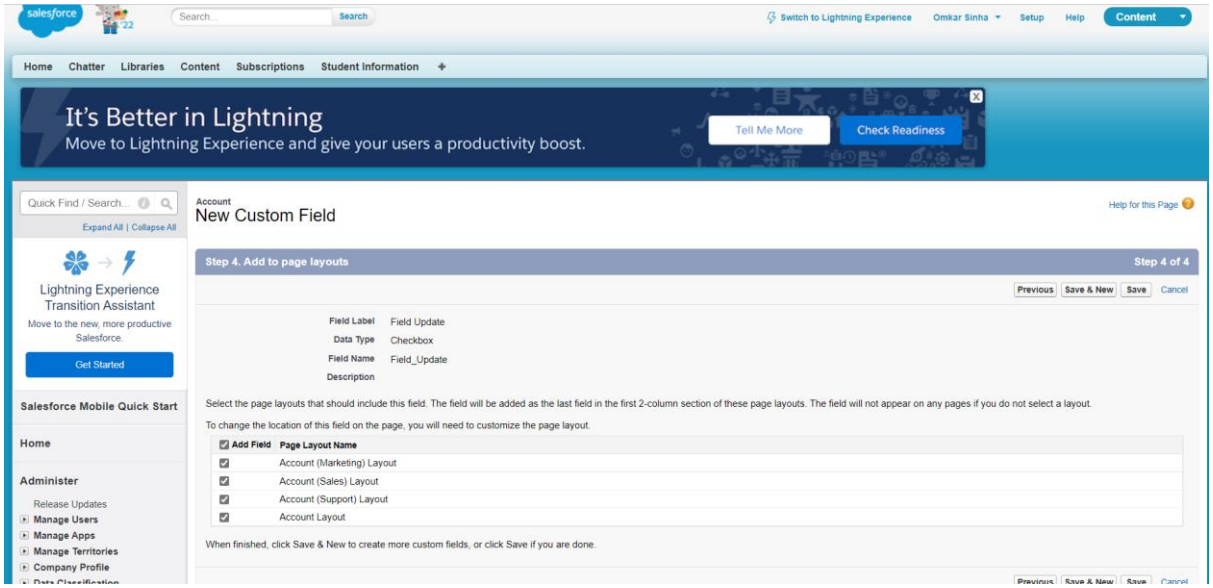
(iii) We enter the other details for the field, ad go to next:



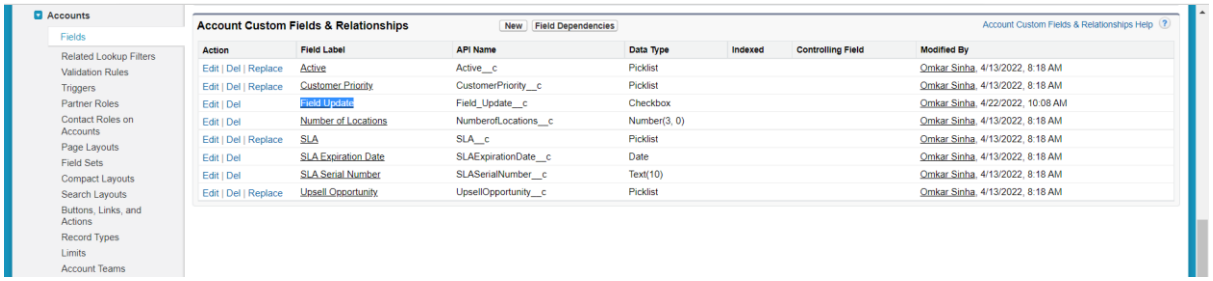
(iv) We check "visibility" and go to next:



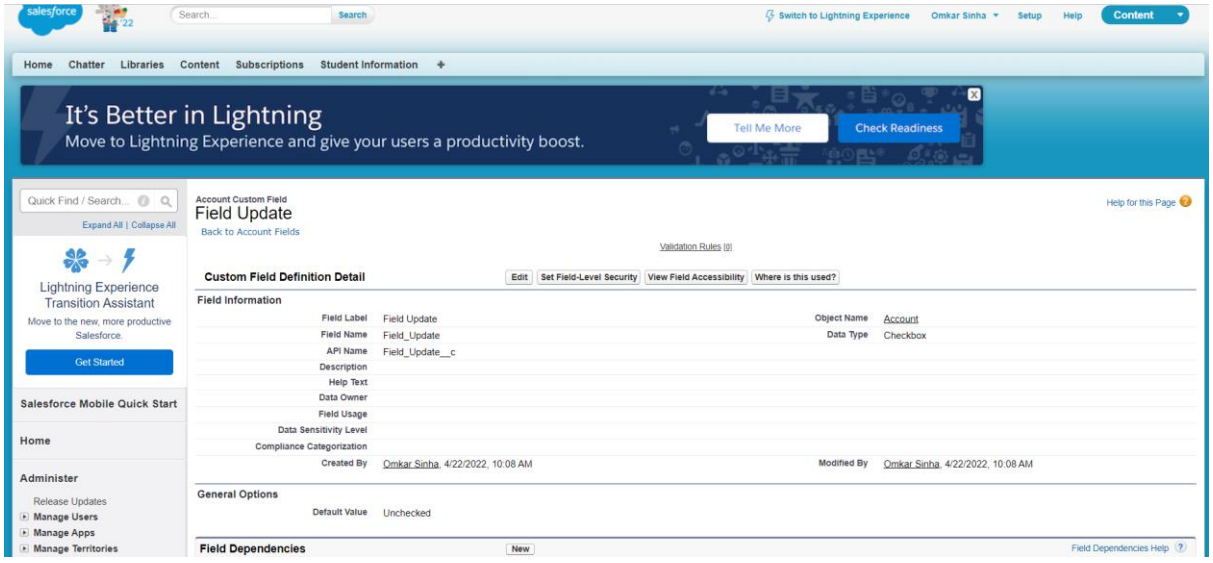
(v) In add to page layout, we select all the checkboxes and then select Save:



(vi) We see "Field Update" added to "Account Custom Fields & Relationships":

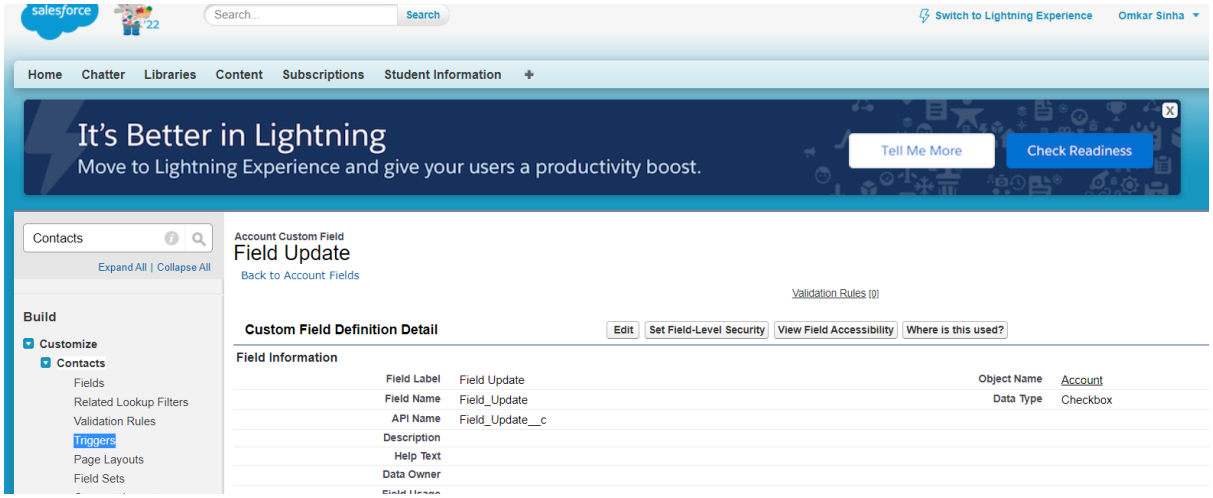


(vii) We select "Field Update" and look at its details as follows:

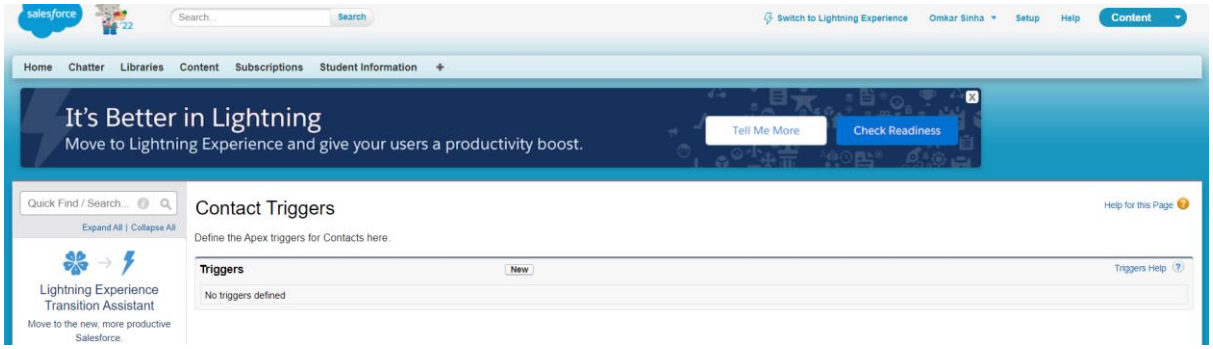


3) Creating Apex trigger on Contact Object:

(i) Now, we will create trigger on the Account object. We search "Accounts" in quick find bar and go to triggers:



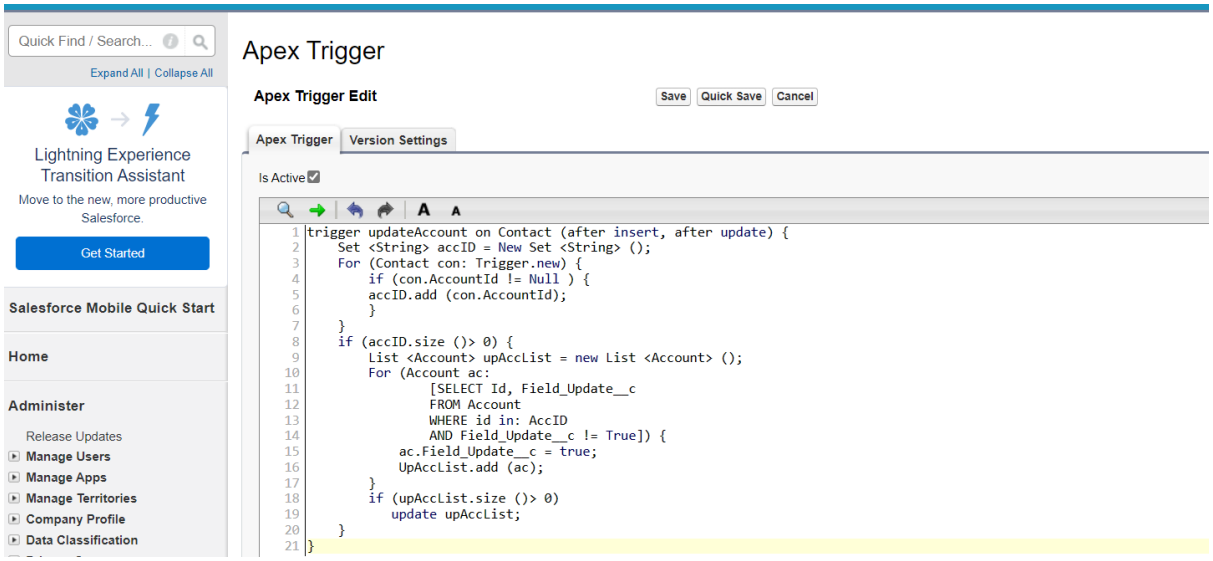
(ii) We choose "New" to create new trigger:



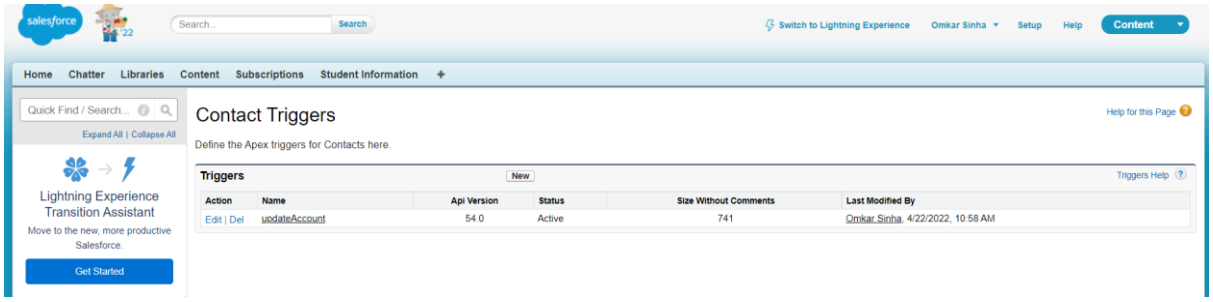
(iii) We paste following code for apex trigger:

```
trigger updateAccount on Contact (after insert, after update) {
   Set <String> accID = New Set <String> ();
   For (Contact con: Trigger.new) {
      if (con.AccountId != Null ) {
      accID.add (con.AccountId);
      }
   }
   if (accID.size ()> 0) {
      List <Account> upAccList = new List <Account> ();
      For (Account ac:
            [SELECT Id, Field_Update__c
            FROM Account
            WHERE id in: AccID
            AND Field_Update__c != True]) {
```

```
        ac.Field_Update__c = true;
        UpAccList.add (ac);
      }
     if (upAccList.size ()> 0)
       update upAccList;
   }
}
```



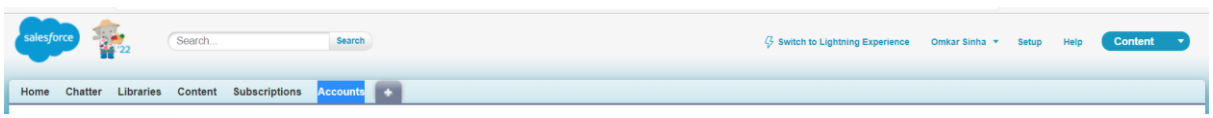
We check that “Is Active” is selected and then save it. We see trigger is created:



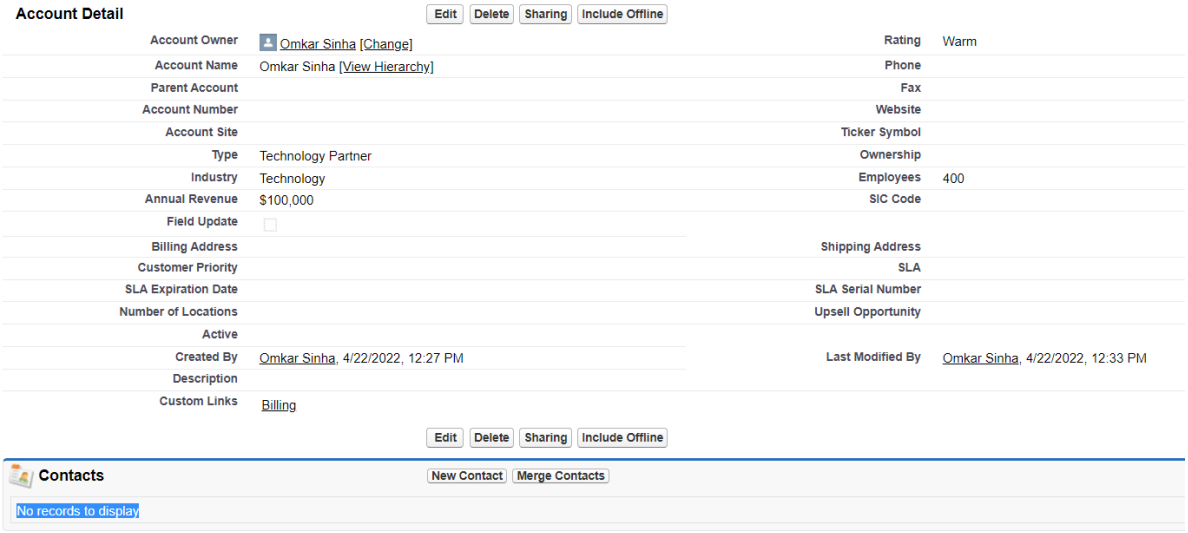
3) Creating a Contact record:

Now we will be creating Contact records from the Account object.

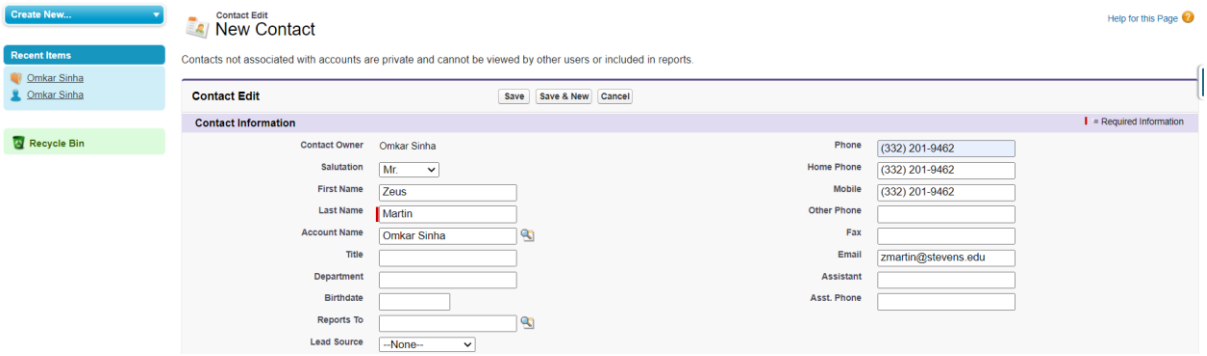
(i) Firstly, we click on the Accounts object:

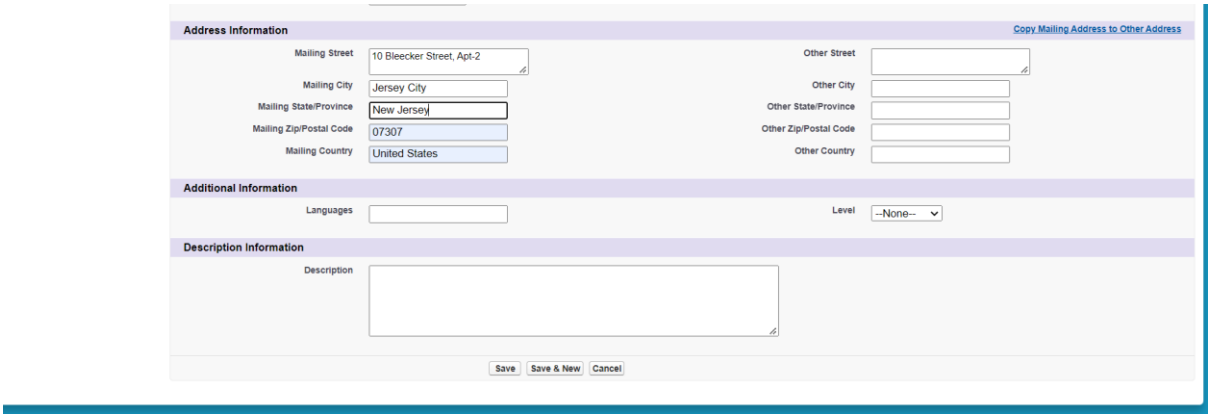


(ii) We see our pre-existing record for Account. We notice that the “Contacts” object has no records.



(iii) We click on “New Contact”, to create a new contact. We fill out the details of a particular contact “Zeus Martin” in this case and select save:





(iv) We now again go to our original “Omkar Sinha” account. We see that “Field Update” field on the Account Object is Checked and “Zeus Martin” record is added in the Contacts as a new record: