National College of Ireland

# An efficient Serverless Architecture to support fine-grained coordination and state sharing for Stateful Applications

MSc Research Project
Cloud Computing

Ankit Kumar Singh
Student ID: x19205121

School of Computing
National College of Ireland

Supervisor: Sean Heeney

# Configuration Manual

Ankit Kumar Singh
19205121
MSc in Cloud Computing

16th August 2021

## 1 Introduction

### 1.1 Purpose of this document

Configuration Manual is prepared to specify the Research project requirements described in the National college of Ireland project Handbook. The main aim to prepare the configuration manual is to illustrate the configuration steps and prerequisites to successfully setup and deploy the research project " An efficient Serverless Architecture to support fine-grained coordination and state sharing for Stateful Applications".

### 1.2 Document Structure

| Section | Purpose |
|---|---|
| **General Information** | This section provides the basic information required to setup serverless and other cloud services |
| **Development Environment Prerequisites** | This section explains the steps required to setup deployment environment for proposed architecture |
| **Solution Deployment Procedure** | This section illustrates the solution deployment steps on the serverless platform for k-means machine learning algorithm and distributed shared object layer on EC2 instance. |
| **Validation** | This section explains the various validations steps to validate the proposed approach |

## 2 General Information

### 2.1 Objective

The main motive of the proposed architecture is to provide an efficient serverless architecture to support coordination and gine grained state sharing between cloud functions for stateful applications. To fulfill the requirements for fine-grained state sharing and synchronization, it will be necessary to create and use distributed shared memory layer to store intermediate data. Also, replication of shared objects is required to provide durability and fault tolerance.
The following research question has been addressed to fulfill the objective
*"Can the serverless architecture workflow for stateful applications be enhanced using distributed shared memory to reduce network latency for fine-grained coordination and state management between functions on different serverless platforms e.g. AWS Lambda or Openstack Qinling?"*

## 2.2 Architecture Requirements

This section describes the Amazon web services required to build the proposed architecture.

### 2.2.1 AWS Virtual private network

The Amazon virtual private network is created to isolate the network from the external world and to provide internet access to the lambda function. The Distributed shared object layer server and client both reside in the same VPC with the same subnet group IP address. VPC requires some additional configurations like security groups, route table, and NAT gateway [1].

### 2.2.2 AWS Elastic Compute Cloud

Amazon Ec2 is used to deploy the distributed shared object layer to store mutable objects and perform an operation on them remotely [2].

### 2.2.3 AWS Simple storage service

Amazon simple storage service is used to store initial data of the application [3].

### 2.2.4 AWS lambda

Amazon lambda is serverless compute service. It is used to deploy the Ml application. The functions are treated as cloud threads in this serverless setup [4].

### 2.2.5 AWS Cloud Watch

Amazon cloud watch is used to monitor the performance and resource utilization while performing the experiments. It is also used to extract logs to know the current status of the executing functions [5].

# 3 Development Environment Prerequisites

## 3.1 Code Repository

Refer to the Zip file of the source code submitted to the National College of Ireland on Moodle.

## 3.2 Required Programming languages

The proposed architecture follows object oriented approach, any object oriented programming language can be used to write the lambda function. In my case, I have used java to write the lambda function. Before creating functions make sure to have Java version 8 installed on your machine.

## 3.3 Create IAM Role

Before configuring lambda, create the Identity and Access management role with the following permissions as shown in the figure 1.

- AWSLambdaFullAccess
- AmazonS3FullAccess
- CloudWatchLogsFullAccess
- AWSLambdaVPCAccessExecutionRole
- CloudWatchEventsFullAccess
- AmazonEC2FULLAccess

[1] https://aws.amazon.com/vpc/
[2] https://aws.amazon.com/ec2/
[3] https://aws.amazon.com/s3/
[4] https://aws.amazon.com/lambda/
[5] https://aws.amazon.com/cloudwatch/

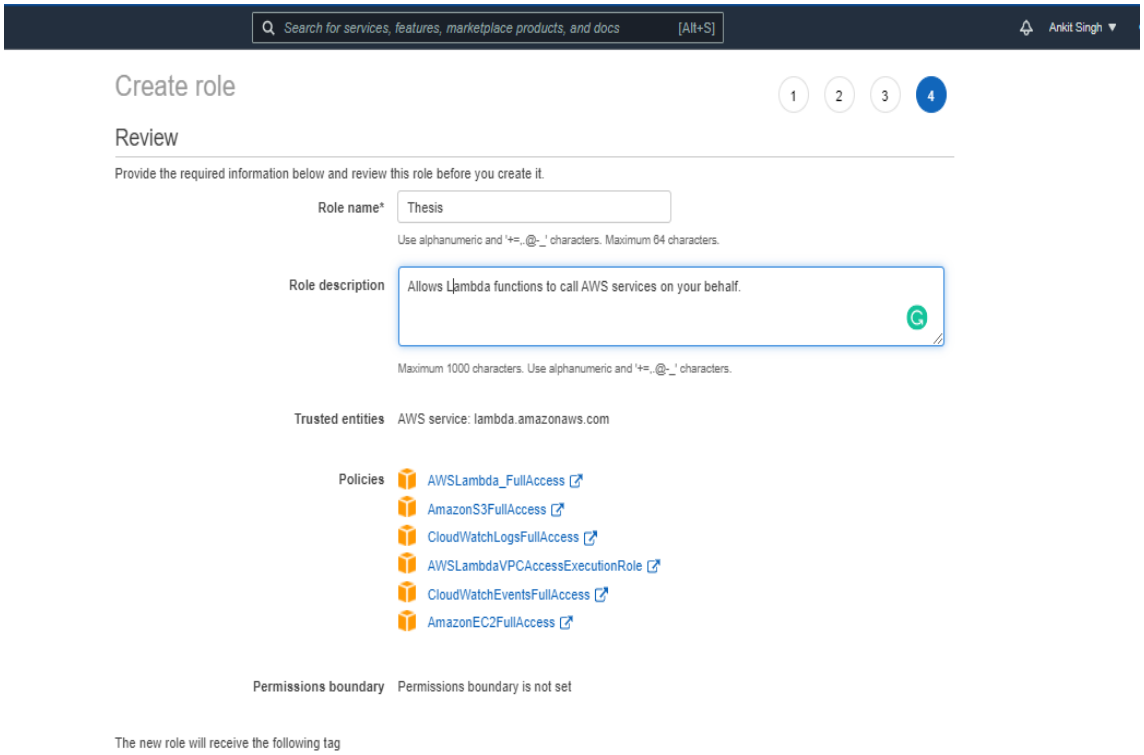



Figure 1: IAM role created to access the lambda and other services.

## 3.4 Configure VPC

On the VPC console navigate to your virtual private cloud and create VPC. enter the following information once you select the create VPC with single subnet as shown in figure 3

- IP CIDR Block
- VPC name
- Public Subnet
- Availability zone
- Subnet name

## 3.5 Create EC2

After creating the VPC, create an EC2 instance with an r5.2xlarge machine having 8 cores. Select the VPC groups created above and choose the Ip address from the subnets created as shown in the figure.

## 3.6 Create S3

Create S3 bucket to store the initial data of the application and also assign S3 endpoint into route table of VPC so that lambda can access the S3 bucket as shown in the figure.

## 3.7 Create lambda function

To create lambda function navigate to AWS lambda dashboard follow the following steps-

- Click on create a function to create lambda function as shown in figure2
- Now select the blueprint option and click next as shown in figure 4
- Enter the function name, subnet, and other permissions.

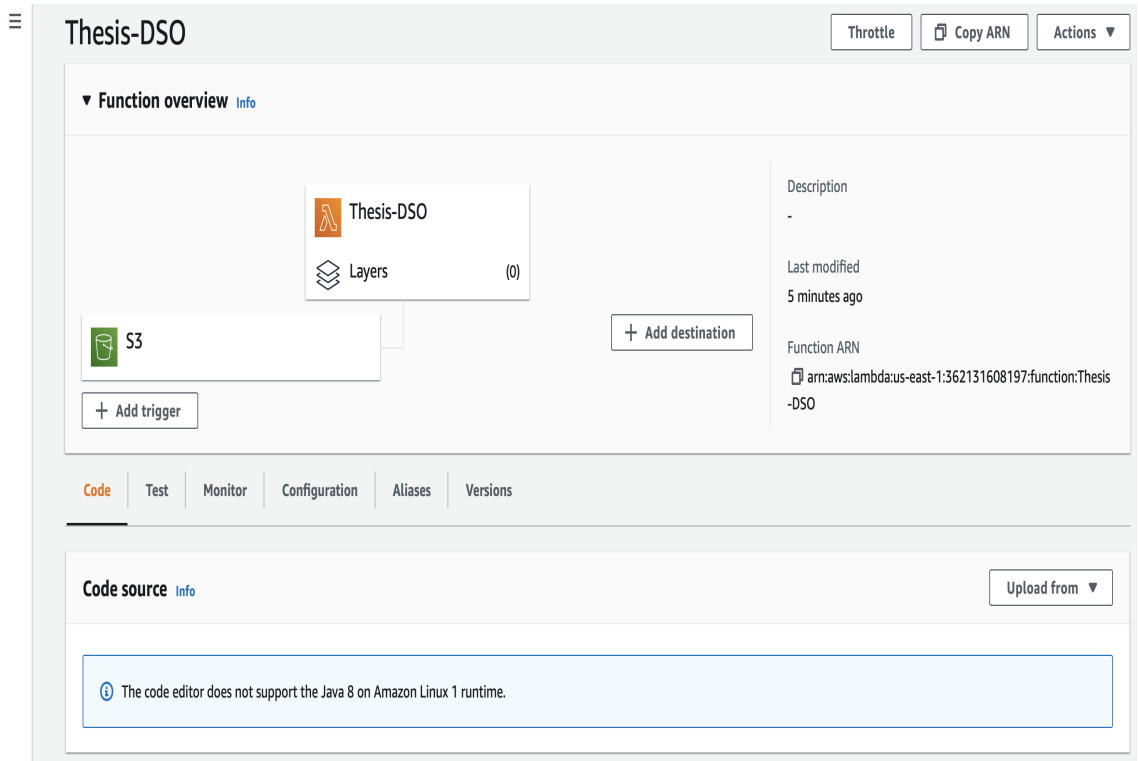



Figure 2: AWS Lambda configuration interface

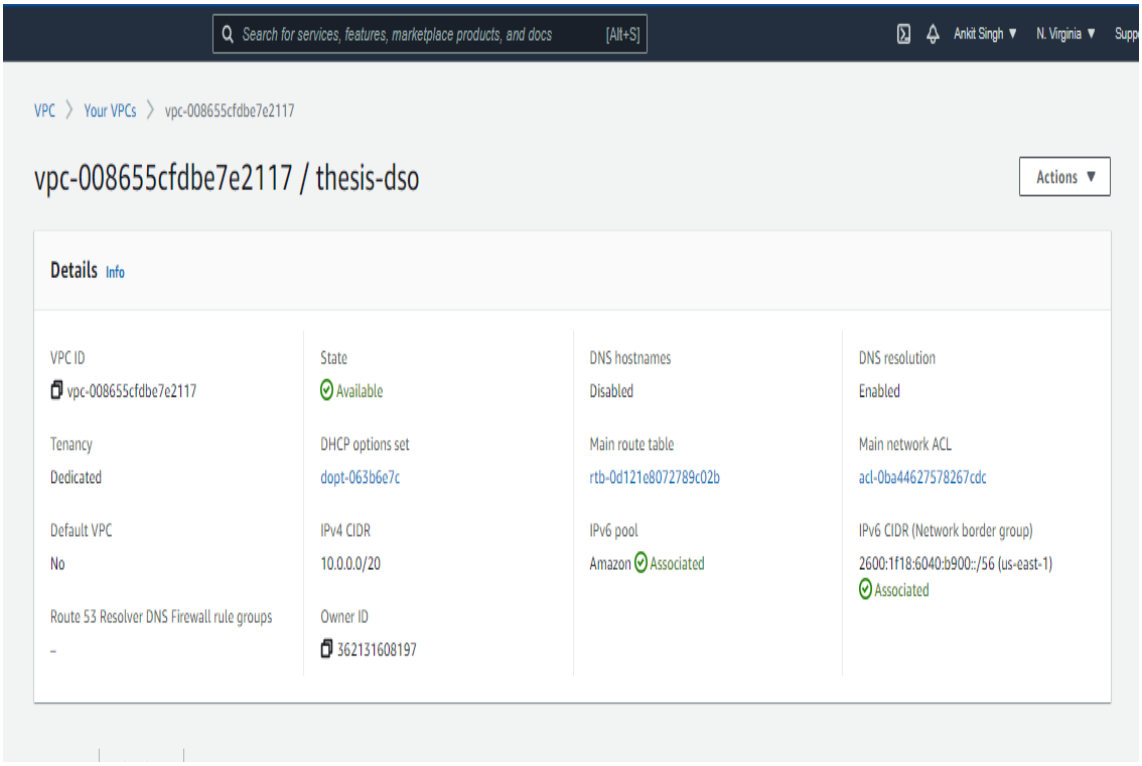



Figure 3: Virtual private network for application

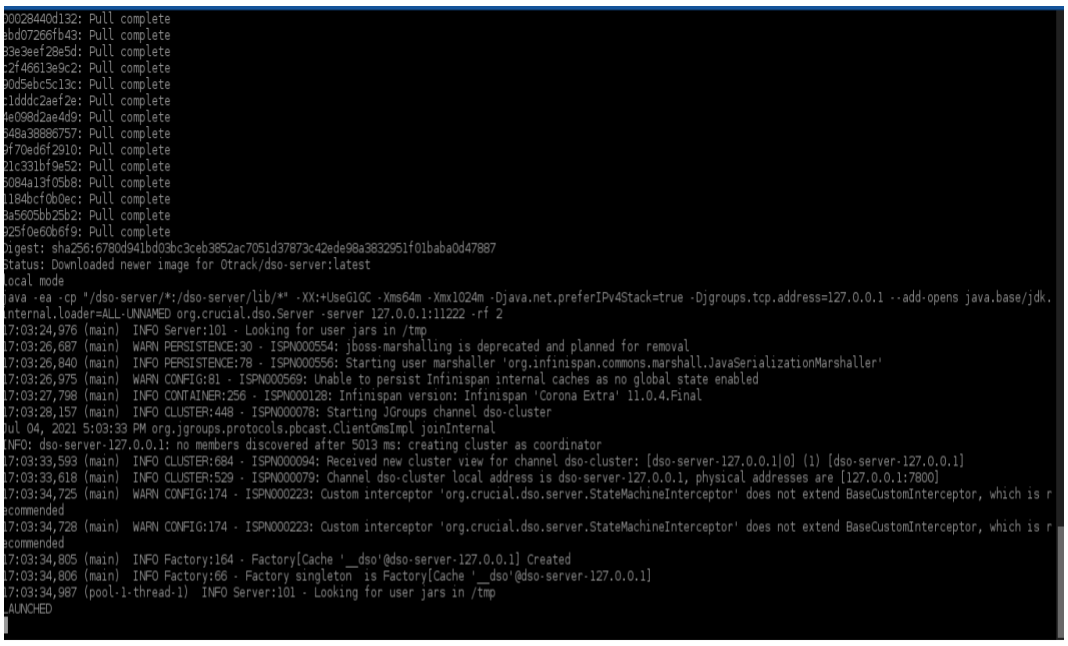
- Configure the network and add a trigger to lambda. To trigger lambda function whenever there is an update.

# 4 Solution Deployment Procedure

## 4.1 Distributed Shared object layer

Following are the steps to create the DSO layer on EC2 as shown in figure 4

- Start the Ec2 machine with r5.2xlarge instance following the development environment.
- Install Docker on the Ec2 machine using command - sudo yum install docker
- pull the github repository using the following command on EC2 machine terminal.
  - git clone - URL of the repository
  - ./ location of the test case script file following -create to create server
  - ./ Write the same location following -delete to delete the server
- To give DSO layer access to S3, you need it change these 3 parameters in XML file
  - location = "your Bucket name"
  - access_key = "Your key"
  - secret_access_key = "Your secret key"




i-07900a28e5d16f6df (trail DSO)

Figure 4: k-means, machine learning implementation on proposed architecture

### 4.1.1 k-means machine learning algorithm

K-means initial data is stored in s3. The k-means code treats the data set as a set of points in comma separated values and partition them into several small parts.

Following are the prerequisites steps to deploy k-means on AWS lambda.

- Check DSO client installed and running in local Maven repo.
- Configure VPC with several subnets assigned

- Put S3 endpoint in VPC route table to allow lambda to access S3
- Define IAM role full with access to Lambda, cloud watch, S3, and lambda VPC access.

Both Client and server should be running in the same VPC group with r5.2xlarge instance.

Before deploying the package to AWS lambda makes few changes to pom.xml

- AWS Account Id
- IAM role name
- Lambda function name
- S3 bucket name to temporarily store files before deploying in lambda.
- region used for the lambda function
- VPC security group ID

# 5 Validation

Firstly, the proposed architecture is validated using micro benchmarks and then further its performance is compared against the Amazon spark. Figure 5 shows the throughput for simple and complex operation . Figure 7 shows the comparison of spark with DSO layer.

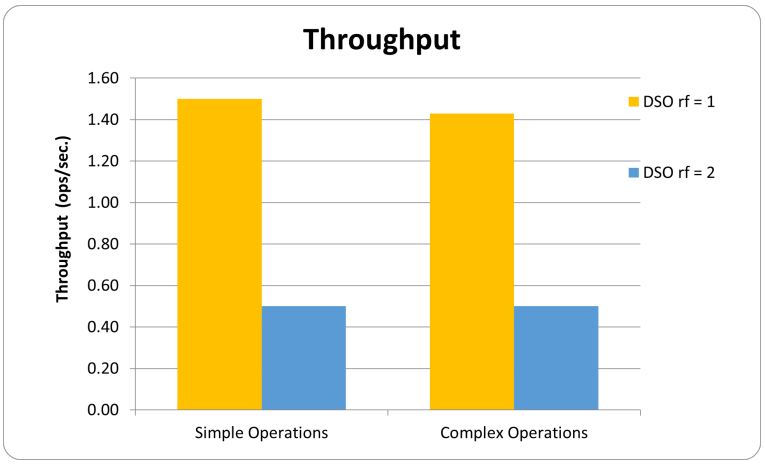



Figure 5: Throughput of the proposed architecture with and without replication (ops/sec in $10^5$)

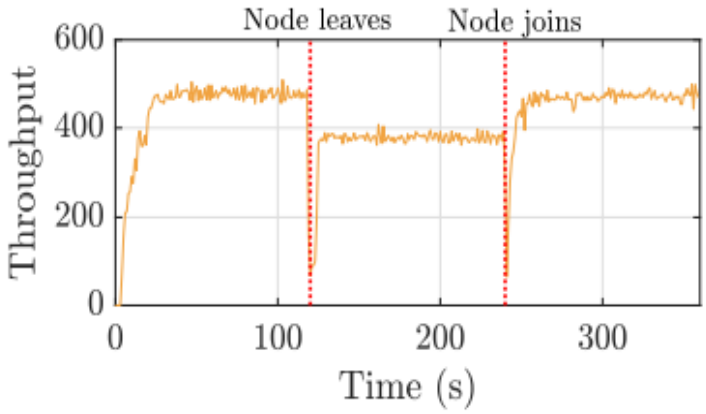



Figure 6: Throughput of the model when k-means algorithm executing.

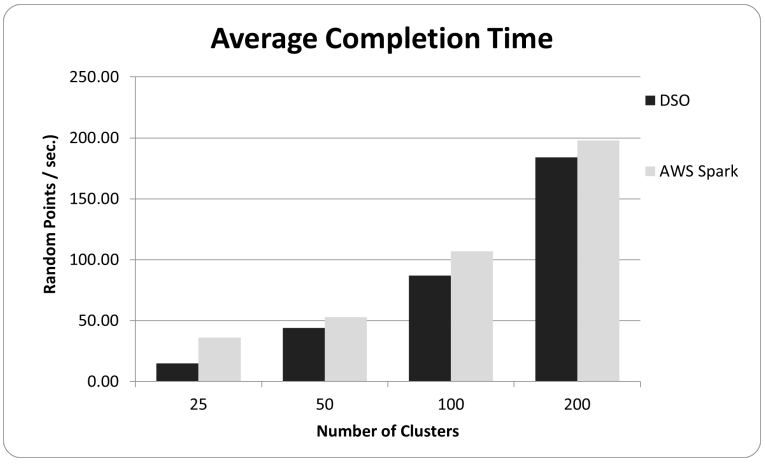



Figure 7: Average time to complete 10 iteration for k-means algorithm with different cluster size