National College of Ireland

Configuration Manual

MSc Research Project
MSc in Cloud Computing

Surya Kumar Govindan
Student ID: 19103883

School of Computing
National College of Ireland

Supervisor: Manuel Tova-Izquierdo

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Surya Kumar Govindan
Student ID:	19103883
Programme:	MSc in Cloud Computing
Year:	2020
Module:	MSc Research Project
Supervisor:	Manuel Tova-Izquierdo
Submission Due Date:	17/08/2020
Project Title:	Configuration Manual
Word Count:	925
Page Count:	8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

Signature:	
Date:	26th September 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Surya Kumar Govindan
19103883

1 Introduction

This document details the necessary steps involved to set up, configure and use the proposed Deep Learning based Serverless (DLS) framework. Additionally, steps to execute a Python-based function on both the DLS framework and Apache OpenWhisk have been detailed and instructions on how to validate and compare these results have been explained as well. The tables below show the details of implementation components and the purpose of each artefact used in the implementation and evaluation.

Table 1: Components and corresponding Products needed for DLS Framework

Component	**Product**
Virtual Machine	Amazon Web Services (EC2)
Operating System	Ubuntu 18.04.4 LTS
Serverless platform	Apache OpenWhisk 0.9.0
Cache manager	Redis 4.0.9
Containerization	Docker* 19.03.12
Docker image	Python:rc-alpine3.12
Programming language	Python 3, Bash
Database	CouchDB* 2.3.1

Table 2: Details of artefacts and their purpose

Artefact	**Purpose**
evaluation/*	Data and visualization file used for evaluation experiments
wsk_comp/helloWorld.zip	Test Python script with 'sql' library to execute on OpenWhisk
wsk_comp/whisk	Script to trigger a function using Apache OpenWhisk
dataset.csv	Original dataset generated for 10 days to gather logs
forecast.csv	File with times at which new containers are to be started
formatted_dataset.csv	Cleaned dataset for training the deep learning model
function	Custom Python script of DLS framework
helloWorld.py	Test Python script with 'sql' library
install.sh	DLS & OpenWhisk installation script
model.py	Python script to train the deep-learning model

2 Configuration

This section details the steps to be followed to configure the DLS framework and Apache OpenWhisk on two individual Virtual Machines respectively.

2.1 Ordering VM

For the implementation, as mentioned in last section, we will be using two AWS EC2 't2.medium' instances. For running the framework without any memory hiccups, a minimum of 't2.medium' size is recommended. Any size below this creates performance issues in accommodating all the components.

Step 1

Login to AWS Management Console Amazon Web Services [n.d.] and create the first EC2 't2.medium' instance and the configuration of the same should look as shown below in Figures 1 and 2 below. The below shown values are the recommended configuration that are to be changed in the AWS EC2 instance launch wizard. Rest all values can be chosen as per AWS defaults. The user will have to choose an existing key pair or create a new key pair to logon to these servers via ssh using tools like putty (recommended) or any compatible bash terminal.

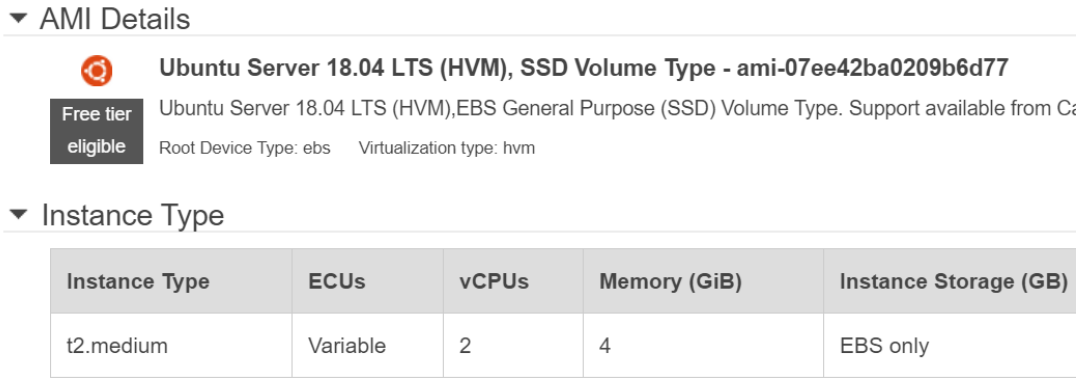

Figure 1: AWS EC2 instance launch review showing image and resources info

Step 2

Two such EC2 instances as shown in Step 1 are to be created. Once done, on the first machine (Machine 1), ensure that the 'git' tool is installed as shown below in the Figure 3, if it is not already installed. NOTE - Most Ubuntu images will come with 'git' installed already.

Step 3

Use the command 'git clone https://github.com/suryakumargovindan/dls.git' to clone the code of the DLS framework from GitHub. Output of the same is shown in the Figure 4. This will create a directory named 'dls' in the current working directory and the installation scripts of the framework will be present inside it, as shown in the Figure 5.

Security group name	launch-wizard-477
Description	launch-wizard-477 created 2020-08-14T16:10:37.027+01:00

Type	Protocol	Port Range	Source
SSH	TCP	22	0.0.0.0/0
All TCP	TCP	0 - 65535	0.0.0.0/0

Instance Details

Storage

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS
Root	/dev/sda1	snap-0fc98bbeb73c2fb3b	30	gp2	100 / 3000

Figure 2: AWS EC2 instance launch review showing security group and storage

```
ubuntu@ip-172-30-0-177:~$ sudo apt install git
Reading package lists... Done
Building dependency tree
Reading state information... Done
git is already the newest version (1:2.17.1-1ubuntu0.7).
```

Figure 3: Check whether git is installed on the VM

```
ubuntu@ip-172-30-0-177:~$ git clone https://github.com/suryakumargovindan/dls.git
Cloning into 'dls'...
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 13 (delta 0), reused 10 (delta 0), pack-reused 0
Unpacking objects: 100% (13/13), done.
```

Figure 4: Command showing cloning the given GitHub repository

2.2 Run installation scripts

Once the GitHub repository is configured successfully, switch to the 'dls' directory and the installation of DLS framework could be started with the command - 'sudo nohup bash install.sh > installation.log &', as shown in the Figure 5. This will start installing the prerequisites Apache OpenWhisk [n.d.a], Dale Lane [n.d.] (Docker, Docker-Compose, Python, Redis etc.) for DLS framework automatically in the background and the logs of this can be checked in the file 'installation.log'. Once triggered, the directory should look as shown in the Figure 6.

```
ubuntu@ip-172-30-0-177:~/dls$ ll
total 36
drwxrwxr-x  3 ubuntu ubuntu  4096 Aug 11 18:31 ./
drwxr-xr-x 11 ubuntu ubuntu  4096 Aug 11 18:31 ../
drwxrwxr-x  8 ubuntu ubuntu  4096 Aug 11 18:17 .git/
-rw-rw-r--  1 ubuntu ubuntu    49 Aug 11 18:16 README.md
-rwxrwxr-x  1 ubuntu ubuntu 10298 Aug 11 18:20 function*
-rwxrwxr-x  1 ubuntu ubuntu    67 Aug 11 18:16 helloWorld.py*
-rwxrwxr-x  1 ubuntu ubuntu  3876 Aug 11 18:31 install.sh*
ubuntu@ip-172-30-0-177:~/dls$ sudo nohup bash install.sh > installation.log \&
```

Figure 5: Starting the installation of DLS framework

```
ubuntu@ip-172-30-0-177:~/dls$ ll
total 40
drwxrwxr-x  3 ubuntu ubuntu  4096 Aug 11 18:40 ./
drwxr-xr-x 11 ubuntu ubuntu  4096 Aug 11 18:39 ../
drwxrwxr-x  8 ubuntu ubuntu  4096 Aug 11 18:17 .git/
-rw-rw-r--  1 ubuntu ubuntu    49 Aug 11 18:16 README.md
-rwxrwxr-x  1 ubuntu ubuntu 10298 Aug 11 18:20 function*
-rwxrwxr-x  1 ubuntu ubuntu    67 Aug 11 18:16 helloWorld.py*
-rwxrwxr-x  1 ubuntu ubuntu  3876 Aug 11 18:31 install.sh*
-rw-rw-r--  1 ubuntu ubuntu   659 Aug 11 18:39 installation.log
-rw-rw-r--  1 ubuntu ubuntu     0 Aug 11 18:40 whisk_install.log
```

Figure 6: The current working directory with logs of installation

2.3 Verify installation

Check whether the installation has been successful by using the DLS framework's 'function' script to execute a Python based simple 'helloWorld' program. The output of the same could be seen in the Figure 7. A similar output will show that all the components have been properly configured and installed.

Once verified, repeat the steps defined in sections (2.1) and (2.2), to create the machine 2 and install OpenWhisk components on it. The same script 'install.sh' will also perform the installation of Apache OpenWhisk and the logs for the same could be checked in file 'whisk_install.log'. Below pictures show how to validate the installation of Apache OpenWhisk components. To verify whether the OpenWhisk components are installed and running, enter the command 'sudo dps | grep -i openwhisk' which shows the list of corresponding Docker containers running for OpenWhisk. 'dps' is a manual command created and installed by the DLS framework from section (2.2). This can be seen in the Figure 8 and there should be 10 OpenWhisk related containers running.

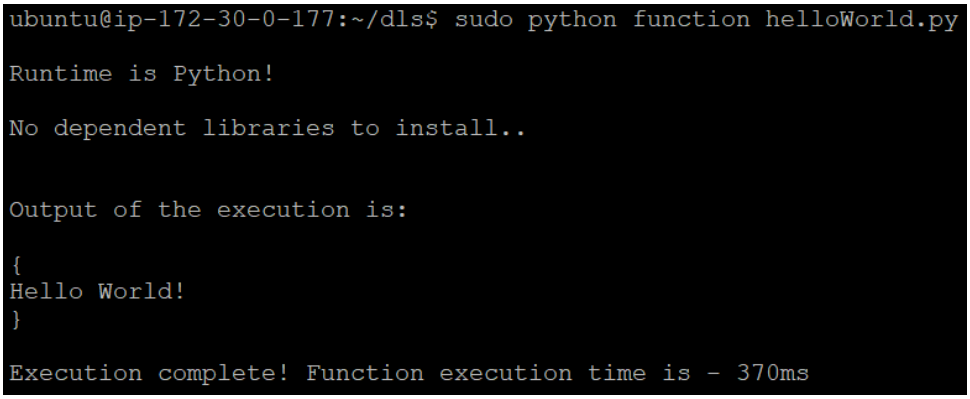

Figure 7: Verifying the successful installation of DLS

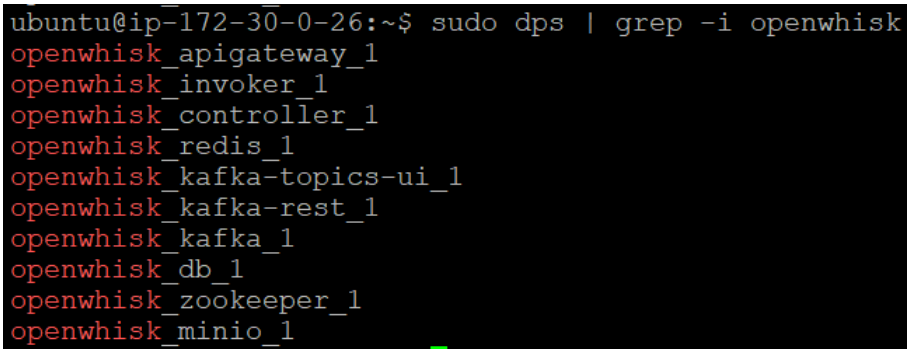

Figure 8: List of OpenWhisk components running on Machine 2

Once OpenWhisk components are verified, validate that the OpenWhisk engine components are properly read by the OpenWhisk client (wsk) Apache OpenWhisk [n.d.b] by executing the command as root - 'wsk -i property get', as shown in Figure 9.

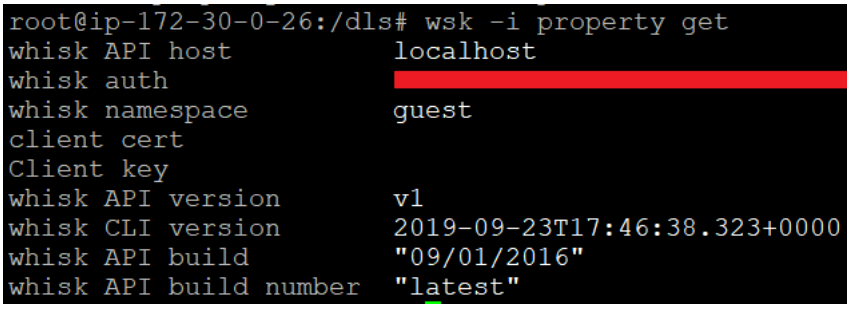

Figure 9: Verification of OpenWhisk engine and client

3 Validation

In this section steps to call a sample Python script with 'sql' as the dependent library is given. The sample script for the same are given along with the GitHub repository.

3.1 Function execution in DLS

Execute the command 'sudo python function helloWorld.py' to run the Python script helloWorld.py using the DLS framework's 'function' command, shown in the Figure 10. The logs of the execution can be found in Figure 11 in the file 'functions_list.csv'

```
ubuntu@ip-172-30-0-228:~/dls$ sudo python function helloWorld.py

Runtime is Python!

Downloading libraries as it's not found in cache...

Installing required libraries...

Collecting sql
  Downloading sql-0.4.0.tar.gz (3.6 kB)
Building wheels for collected packages: sql
  Building wheel for sql (setup.py): started
  Building wheel for sql (setup.py): finished with status 'done'
  Created wheel for sql: filename=sql-0.4.0-py3-none-any.whl size=
  Stored in directory: /root/.cache/pip/wheels/6d/84/b8/cfeff5ca9
Successfully built sql
Installing collected packages: sql
Successfully installed sql-0.4.0


Output of the execution is:

{
Hello World!
}

Execution complete! Function execution time is - 155ms
```

Figure 10: Executing a Python script using DLS framework's 'function' command

```
ubuntu@ip-172-30-0-228:~/dls$ cat functions_list.csv
Timestamp, Function, Duration, Type
2020-08-13 17:13:51,helloWorld.py,3087,cold
```

Figure 11: Logs of script execution using 'function' of DLS framework

3.2 Function execution in OpenWhisk

Switch to the directory 'whisk' and then execute the command 'sudo python function helloWorld.py' to run the Python ZIP file helloWorld.zip using Apache OpenWhisk's 'wsk' cli tool, as shown in the Figure 12. When Python scripts have dependent libraries, they have to be packaged in a ZIP file and then are to be run by creating and invoking actions OpenWhisk [n.d.] using 'wsk'.

The customized 'whisk' command created for this automates these tasks. The output of the execution using OpenWhisk can be found in Figure 13 in the file 'whisk_list.csv'. The activation list of Apache OpenWhisk could be found in the Figure 14.

```
ubuntu@ip-172-30-0-228:~/dls/wsk_comp$ sudo python whisk helloWorld.zip
ok: created action helloWorld
ok: invoked /_/helloWorld with id ee5204e8b03b43c49204e8b03b93c431
{
    "activationId": "ee5204e8b03b43c49204e8b03b93c431",
    "annotations": [
        {
            "key": "path",
            "value": "guest/helloWorld"
```

Figure 12: Executing Python script on Apache OpenWhisk with custom 'whisk' command

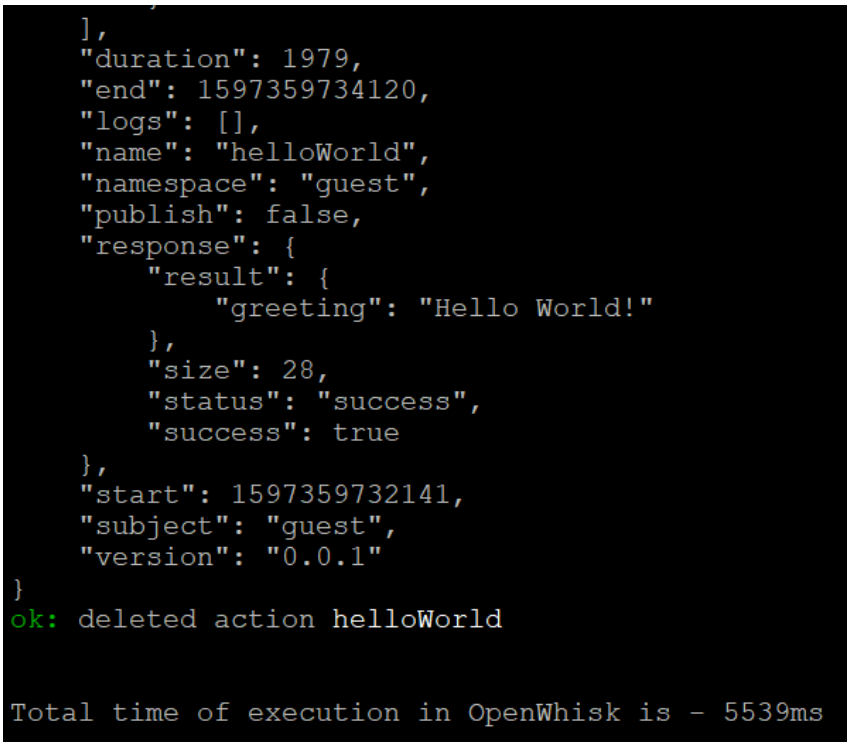

Figure 13: Output of executing Python script in Apache OpenWhisk

```
ubuntu@ip-172-30-0-228:~/dls/wsk_comp$ wsk -i activation list
Datetime            Activation ID                  Kind      Start Duration   Status  Entity
2020-08-13 17:15:53 4fa6e1c52ae14fa1a6e1c52ae11fa121 python:3  cold  1.301s     success guest/helloWorld
2020-08-13 16:47:19 2e4220a616c84cc38220a616c8fcc3ed python:3  cold  1.721s     success guest/helloWorld
2020-08-13 16:29:51 c2fe9eb83ae6480bbe9eb83ae6980b27 python:3  warm  5ms        success guest/helloWorld
2020-08-13 16:20:45 16ec2e752fac4fdeac2e752fac9fdefc python:3  cold  1.577s     success guest/helloWorld
```

Figure 14: Log of function activations in Apache OpenWhisk

3.3 Validation of results

From the files 'functions_list.csv' and 'whisk_list.csv', compare the total duration taken for executing the same script on DLS (on machine 1) and Apache OpenWhisk (on machine 2), right from the call of the script till completion of execution. A similar comparison could be seen from the figures 15 & 16.

```
ubuntu@ip-172-30-0-177:~/dls$ cat functions_list.csv
Timestamp, Function, Duration, Type
2020-08-13 23:17:45,helloWorld.py,2276,cold
```

Figure 15: Log of test script execution using 'function' command for DLS

```
ubuntu@ip-172-30-0-228:~/dls/wsk_comp$ cat whisk_list.csv
Timestamp, Function, Duration
2020-08-13 23:17:46,helloWorld.zip,5226
```

Figure 16: Log of test script execution using 'whisk' command for Apache OpenWhisk

3.4 Forecast

To use the forecast option of the DLS framework's to schedule containers at given times, run the command 'sudo python function forecast helloWorld.py', which will schedule the containers to start at given times from file 'forecast.csv', as shown in figure 17.

```
ubuntu@ip-172-30-0-228:~/dls$ sudo python function forecast
One container(s) scheduled to start at 00:29:57
One container(s) scheduled to start at 00:44:57
One container(s) scheduled to start at 00:59:57
One container(s) scheduled to start at 01:14:57
One container(s) scheduled to start at 01:29:57
One container(s) scheduled to start at 01:44:57
One container(s) scheduled to start at 01:59:57
One container(s) scheduled to start at 02:14:57
One container(s) scheduled to start at 02:29:57
One container(s) scheduled to start at 02:44:57
```

Figure 17: Output of using 'forecast' feature of DLS framework

References

Amazon Web Services [n.d.]. AWS EC2 Instance Launch Wizard, https://eu-west-1.console.aws.amazon.com/ec2/v2/home?region=eu-west-1#LaunchInstanceWizard:.

Apache OpenWhisk [n.d.a]. Installing Apache OpenWhisk, https://github.com/apache/openwhisk#quick-start.

Apache OpenWhisk [n.d.b]. OpenWhisk Client Releases, https://github.com/apache/openwhisk-cli/releases.

Dale Lane [n.d.]. Getting started with OpenWhisk and Kafka , https://dalelane.co.uk/blog/?p=3741.

OpenWhisk [n.d.]. OpenWhisk Documentation, https://openwhisk.apache.org/documentation.html#actions-creating-and-invoking.