Configuration Manual for implementing Dynamic Resource Algorithm and ARIMA Time-Series Analysis

MSc Research Project
Cloud Computing

Saifali Sayyed
Student ID: x18178294

School of Computing
National College of Ireland

Supervisor: Prof. Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Saifali Sayyed
Student ID:	x18178294
Programme:	MSc in Cloud Computing
Year:	2020
Module:	MSc Research Project
Supervisor:	Prof. Vikas Sahni
Submission Due Date:	17/08/2020
Project Title:	Optimisation of Resource Allocation and Prediction Analysis in Serverless Computing using Dynamic Resource Algorithm
Word Count:	1067
Page Count:	10

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:	16th August 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 for implementing Dynamic Resource Algorithm and ARIMA Time-Series Analysis

Saifali Sayyed
x18178294

16th August 2020

1 Introduction

For implementing this project Python and Bash programming languages have been leveraged. Libraries such as pandas, NumPy, stats model, ARIMA, and tools such as Prometheus, Docker, Kubernetes, Jupyter Notebook, AWS CloudWatch, and OpenFaaS have been used. This configuration manual will also help to execute the overall project step by step.

2 Public Cloud Platform

As this project is deployed on the AWS EC2 virtual machine[1]. A core understanding of the AWS Cloud Platform is needed for creating the AWS EC2 Instance. Below are the configuration details for the AWS EC2 instance creation.

Table 1 : AWS EC2 Configuration Details

Steps	**Configurations**	**Specifications**
Step 1	Choose AMI	Ubuntu Server 16.04 LTS (HVM)
Step 2	Choose Instance Type	t2.large (2 vCpus, 8GB Memory)
Step 3	Instance Details	Public VPC, Administrator IAM Role Attached
Step 4	Storage	8GB General Purpose SSD
Step 5	Security Groups	All Traffic

[1]https://docs.aws.amazon.com/efs/latest/ug/gs-step-one-create-ec2-resources.html

3 Required Languages

The proposed approach is implemented using Bash Script, Python 2.7 and Python 3.8 language. By default AWS EC2 Ubuntu 16.04 Instance will be having Bash, Python 2.7 and Python 3.8 version already installed. If you are using different Public Cloud Platform or Python 3.8 is not installed you can install it by using below command.

See the following command :

```
$ sudo apt-get install python3
```

Next, pip and pip 3 software package manager has to be install which will require for installing the Python Libraries. Use below command for installing pip and pip3.

Install Pip:

```
$ sudo apt install python-pip
```

Upgrade Pip:

```
$ pip install --upgrade pip
```

Install Pip3:

```
$ sudo apt install python3-pip
```

4 Installation of Platforms and Tools

As this project has leveraged multiple tools and platforms. In this section, these tools are installed in a logical sequence.

1. **Install Docker and Start Service**

   Command for installing latest Docker version:

   ```
   $ sudo apt-get install docker.io
   ```

   Add root user to docker group:

   ```
   $ sudo gpasswd -a root docker
   ```

   Start Docker Service:

   ```
   $ sudo systemctl enable --now docker
   ```

2. **Install Kubectl**

   Confirm the Kubectl Installation:

```
$ snap install kubectl --classic
```

   Confirm the Kubectl Installation:

```
$ kubectl version --client
```

3. **Install Minikube**

   Before installing Minikube, the virtualization has to confirmed is it supported or not on the underlying Ubuntu Machine. Use the below command for confirming the virtualization, if no output is displayed that means virtualization is supported.

   Confirm Virtualization using below command:

```
$ grep -E --color 'vmx|svm' /proc/cpuinfo
```

   Next, **install** the Minikube using direct download.[2]

   Install Minikube:

```
$ curl -Lo minikube https://storage.googleapis.com/
   ↪ minikube/releases/latest/minikube-linux-amd64 \
  && chmod +x minikube

$ sudo mkdir -p /usr/local/bin/

$ sudo install minikube /usr/local/bin/
```

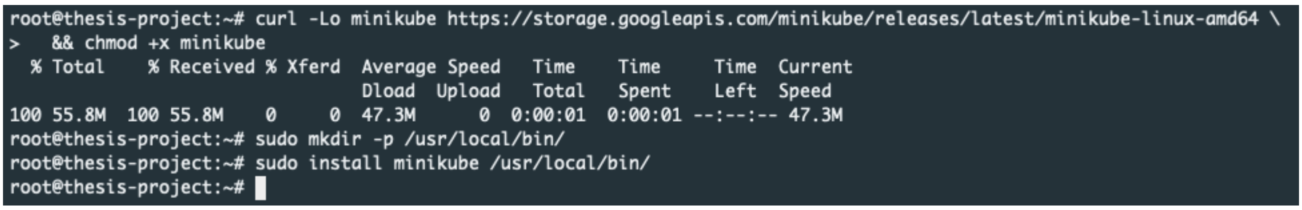


Figure 1: Minikube Installation

   Minikube requires conntrack to be installed before starting the cluster. Use the below command for it.

   Install Conntrack:

```
$ sudo apt-get install conntrack
```

   Once it is installed use the below command for starting the Minikube cluster first time only once.

[2]https://kubernetes.io/docs/tasks/tools/install-minikube/install-minikube-using-a-package

Start Minikube Cluster:

```
$ minikube start --vm-driver=none
```

4. **Install OpenFaaS-Cli**

   Install the OpenFaaS Cli using the curl command, this utility is already installed in the AWS EC2 Ubuntu 16.04 machine.

   Install OpenFaaS Cli:

```
$ curl -sL https://cli.openfaas.com | sudo sh
```

5. **Deploying OpenFaaS platform**

   Next, the OpenFaaS deployment has to be done. This will create an OpenFaaS application stack.

   Clone the Repository:

```
$ git clone https://github.com/openfaas/faas-netes
```

Creation of OpenFaaS Namespaces:

```
$ kubectl apply -f https://raw.githubusercontent.com/
   ↪ openfaas/faas-netes/master/namespaces.yml
```

Figure 2: OpenFaaS Namespaces

Creating OpenFaaS API Gateway Password:

```
$ PASSWORD=password123

$ kubectl -n openfaas create secret generic basic-auth \
--from-literal=basic-auth-user=admin \
--from-literal=basic-auth-password="$PASSWORD"
```

Deployment of OpenFaaS Stack:

```
$ cd faas-netes && \
kubectl apply -f ./yaml
```

Login to OpenFaaS Gateway:

```
$ export OPENFAAS_URL=http://127.0.0.1:31112
$ echo -n $PASSWORD | faas-cli login --password-stdin
```

Once, all the platforms and tools are installed, the OpenFaaS Dashboard can be access now. For accessing the OpenFaaS Dashboard use the below url in Google Chrome browser. Figure 3 depict the OpenFaaS Dashboard

`http://public_ip_instance:31112/`

Credentials for login:
username: admin
password: password123

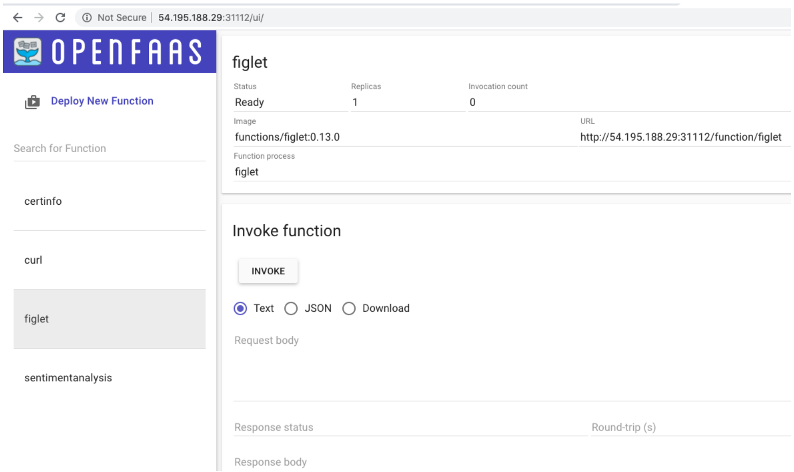


Figure 3: OpenFaaS Dashboard

6. **Create the Figlet Function:**

   The figlet function can be created using the OpenFaaS Dashboard.

   Step 1: Deploy New Function
   Step 2: Select Store
   Step 3: Select Figlet
   Step 4: Create Figlet Function

7. **Install AWS-CLI**

   For sending the ARIMA predicted values to the AWS CloudWatch, AWS-CLI[3] is required which can be installed using below command.

   Install AWS-CLI:

```
$ sudo apt install aws-cli
```

8. **Jupyter Notebook**

   For executing the ARIMA time-series analysis code, jupyter notebook[4] has to be installed. Use below command for installing Jupyter Notebook.

[3]https://aws.amazon.com/cli/
[4]https://jupyter.org/

Install Jupyter Notebook:

```
$ pip install notebook
```

Start Jupyter Notebook:

```
$ jupyter notebook --allow-root
```

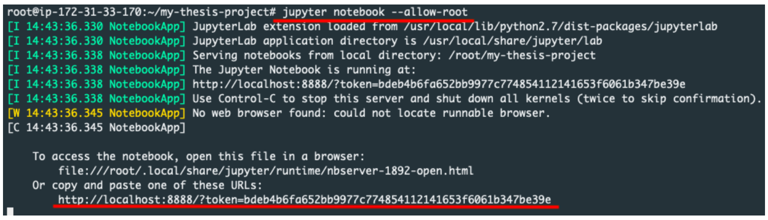


Figure 4: Start Jupyter Notebook

The figure 4, depicts the output after executing jupyter notebook command which shows the URL and port to access the dashboard, here the port is 8888. Jupyter Notebook Dashboard will not open using the AWS EC2 instance public-ip. For accessing the jupyter dashboard, ssh-tunnel has to be created between our machine and remote machine which is in our case is AWS EC2 Instance. Use the below command to create it.

Creating SSH Tunnel:

```
$ ssh -i keyname.pem -L ourport:localhost:8888
    ↪ ubuntu@publicip
```

Now use the below URL for accessing the jupyter dashboard in chrome browser.

`localhost:8888`

Figure 5 depicts the jupyter dashboard.

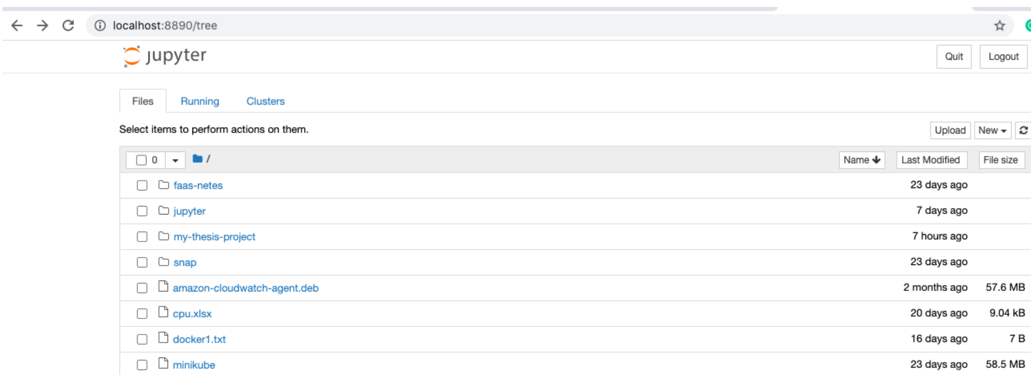


Figure 5: Jupyter Notebook Dashboard

9. **Install Parallel**

   The proposed approach is executed using the parallel ubuntu utility, in which two commands can be executed paralelly[5]. Use the below command to install it.

   Install Parallel:

```
$ sudo apt install parallel
```

10. **Install AWS CloudWatch Agent on Ubuntu**

    For sending the ARIMA predictions value to AWS CloudWatch. AWS CloudWatch Agent[6] has to be installed. Use the below command to install it.

    Download AWS CloudWatch Agent:

```
$ wget https://s3.region.amazonaws.com/amazoncloudwatch-
    ↪ agent-region/ubuntu/arm64/latest/amazon-cloudwatch
    ↪ -agent.deb
```

    Install AWS CloudWatch Agent:

```
$ sudo dpkg -i -E ./amazon-cloudwatch-agent.deb
```

11. **Deploy Grafana Pod**

    In proposed approach, Prometheus is already integrated with the OpenFaaS API Gateway. For retrieving the Prometheus metrics from the it and visualizing it on the the Grafana Dashboard perform the below steps.

    Deploy Grafana Pod in OpenFaaS namespace:

```
 $ kubectl -n openfaas run \
--image=stefanprodan/faas-grafana:4.6.3 \
--port=3000 \
grafana
```

    Expose Grafana NodePort:

```
 $ kubectl -n openfaas expose pod grafana \
--type=NodePort \
--name=grafana
```

[5] https://www.gnu.org/software/parallel/

[6] https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/download-cloudwatch-agent-commandline.html

NodePort Address:

```
$ GRAFANA_PORT=$(kubectl -n openfaas get svc grafana -o
  ↪ jsonpath="{.spec.ports[0].nodePort}")

$ GRAFANA_URL=http://IP_ADDRESS:$GRAFANA_PORT/dashboard/
  ↪ db/openfaas

$ echo $GRAFANA_PORT
```

Above *echo* command will display the Grafana port and browse the below URL in the chrome browser to access the Grafana Dashboard.

username: admin
password: admin

`http://IP-ADDRESS:GRAFANA-PORT/dashboard/db/openfaas`

5 Installing Required Libraries

In this section, the required Python Libraries are installed which are implemented in proposed approach.

1. **Install Pandas:**

```
$ pip install pandas
```

Install Matplotlib:

```
$ python3 -m pip install -U matplotlib
```

If there are any error regarding upgrade the matplotlib, use the below command.

Upgrade Matplotlib:

```
$ python3 -m pip install   upgrade   matplotlib
```

Install Numpy:

```
$ pip install numpy
```

Install Statsmodel:

```
$ pip install statsmodels
```

Install IPyKernel:

```
$ python3 -m pip install ipykernel
```

6 Evaluation Steps

6.1 Experiment 1

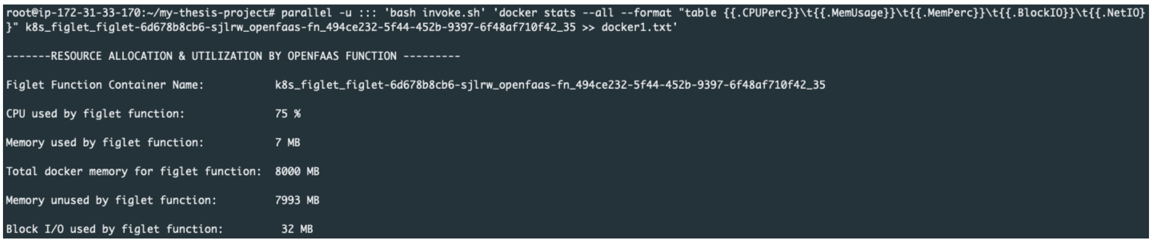
6.1.1 Evaluation of Dynamic Resource Algorithm

1. Using Default Docker Resource Allocation

   Execute invoke.sh script and docker stats command for storing the monitoring metrics in docker1.txt file using Parallel utility.

   Execute invoke script:

```
$ parallel -u ::: 'bash invoke.sh' 'docker stats --all
    ↪ --format "table {{.CPUPerc}}\t{{.MemUsage}}\t{{.
    ↪ MemPerc}}\t{{.BlockIO}}\t{{.NetIO}}"
    ↪ function_container_name >> docker1.txt'
```



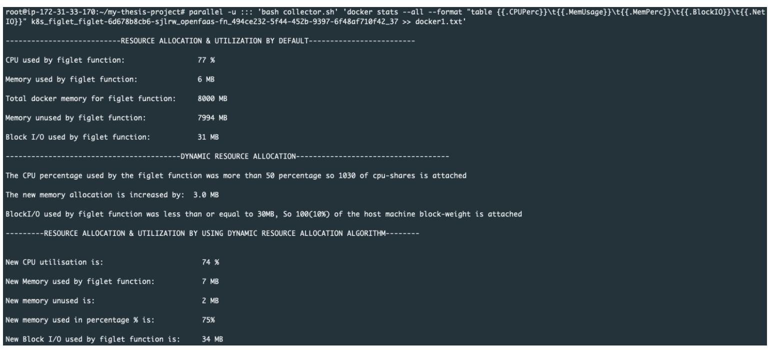
```
root@ip-172-31-33-170:~/my-thesis-project# parallel -u ::: 'bash invoke.sh' 'docker stats --all --format "table {{.CPUPerc}}\t{{.MemUsage}}\t{{.MemPerc}}\t{{.BlockIO}}\t{{.NetIO}
}" k8s_figlet_figlet-6d678b8cb6-sjlrw_openfaas-fn_494ce232-5f44-452b-9397-6f48af710f42_35 >> docker1.txt'

-------RESOURCE ALLOCATION & UTILIZATION BY OPENFAAS FUNCTION ---------

Figlet Function Container Name:          k8s_figlet_figlet-6d678b8cb6-sjlrw_openfaas-fn_494ce232-5f44-452b-9397-6f48af710f42_35

CPU used by figlet function:             75 %

Memory used by figlet function:          7 MB

Total docker memory for figlet function: 8000 MB

Memory unused by figlet function:        7993 MB

Block I/O used by figlet function:        32 MB
```

Figure 6: Default Docker Resource Allocation

2. Using Dynamic Resource Algorithm

   Execute collector script:

```
$ parallel -u ::: 'bash collector.sh' 'docker stats --
    ↪ all --format "table {{.CPUPerc}}\t{{.MemUsage}}\t
    ↪ {{.MemPerc}}\t{{.BlockIO}}\t{{.NetIO}}"
    ↪ function_container_name >> docker1.txt'
```



```
root@ip-172-31-33-170:~/my-thesis-project# parallel -u ::: 'bash collector.sh' 'docker stats --all --format "table {{.CPUPerc}}\t{{.MemUsage}}\t{{.MemPerc}}\t{{.BlockIO}}\t{{.Net
IO}}" k8s_figlet_figlet-6d678b8cb6-sjlrw_openfaas-fn_494ce232-5f44-452b-9397-6f48af710f42_37 >> docker1.txt'

--------------------------RESOURCE ALLOCATION & UTILIZATION BY DEFAULT------------------------

CPU used by figlet function:             77 %

Memory used by figlet function:          6 MB

Total docker memory for figlet function: 8000 MB

Memory unused by figlet function:        7994 MB

Block I/O used by figlet function:       31 MB

----------------------------------------DYNAMIC RESOURCE ALLOCATION-----------------------------------

The CPU percentage used by the figlet function was more than 50 percentage so 1030 of cpu-shares is attached

The new memory allocation is increased by:  3.0 MB

BlockI/O used by figlet function was less than or equal to 30MB, So 100(10%) of the host machine block-weight is attached

---------RESOURCE ALLOCATION & UTILIZATION BY USING DYNAMIC RESOURCE ALLOCATION ALGORITHM--------


New CPU utilisation is:                   74 %

New Memory used by figlet function:       7 MB

New memory unused is:                     2 MB

New memory used in percentage % is:       75%

New Block I/O used by figlet function is: 34 MB
```

Figure 7: Dynamic Resource Algorithm Output

6.1.2 Evaluation of ARIMA Time-Series Model

1. Standard Time-Series Model

   Start the jupyter notebook and navigate to jupyter directory. Open std-time-forecasting.ipynb file and execute it using the header run button. Figure 8 depict the standard time-series analysis model.

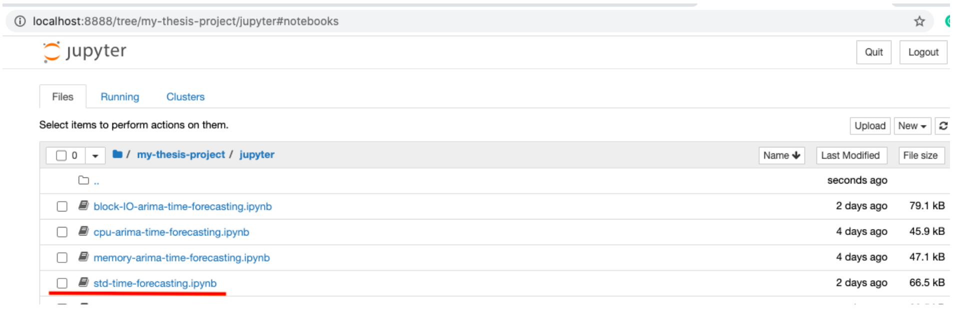


Figure 8: Standard Time-Series Model

2. ARIMA Time-Series Model

   Start the jupyter notebook and navigate to jupyter directory. Open cpu-arima-time-forecasting.ipynb, memory-arima-time-forecasting.ipynb and block-IO-arima-time-forecasting.ipynb and execute it using the header run button. Figure 9 depict the ARIMA time-series analysis model.

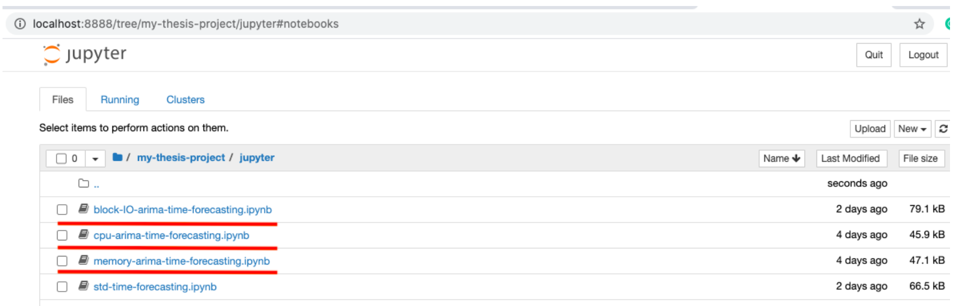


Figure 9: ARIMA Time-Series Model

6.2 Experiment 2

In this experiment, change the figlet function invoke count using the collector.sh script. Open collector.sh script and navigate to line number 4. Change the invoke variable count from 500 to 2000 and execute then execute the collector.sh script using the parallel command specified in Experiment 1.