National College of Ireland

Using Redis for persistent storage in serverless architecture to maintain state management - Configuration Manual

MSc Research Project
Cloud Computing

Ankit Kumar
Student ID: 20149158

School of Computing
National College of Ireland

Supervisor: Adriana Chis

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ankit Kumar
Student ID:	20149158
Programme:	Cloud Computing
Year:	2021
Module:	MSc Research Project
Supervisor:	Adriana Chis
Submission Due Date:	31/01/2022
Project Title:	Using Redis for persistent storage in serverless architecture to maintain state management - Configuration Manual
Word Count:	XXX
Page Count:	6

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.

Signature:	
Date:	31st January 2022

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):	

Using Redis for persistent storage in serverless architecture to maintain state management - Configuration Manual

Ankit Kumar
20149158

1 Introduction

The configuration manual represents an overview of the machine specification that has been used in our experiments along with a detailed explanation of the technologies and libraries. We will discuss how the core concept has been developed using our desired programming language.

1.1 Hardware Specifications

Table 1: Hardware specification

Instance name (flavour)	m6gd.xlarge
vCPU	8
RAM	16
Network Bandwidth	Upto 6 Gbps

1.2 Software Specifications

1.2.1 NodeJS

The idea can be developed using NodeJS or Golang because the community packages and documentation for Golang are comparatively new than NodeJS there are some open issues in the implemented wrappers and libraries, so our choice of language will be NodeJS[1]. In the experiments, we are using two versions of NodeJS. For hosting and core module development, we have used NodeJS 16 and for the serverless function runtime environment, we have used NodeJS 12. Both 12 and 16 version of NodeJS comes under Long-term Support (LTS), therefore it will be a safer choice to choose the above two versions.

1.2.2 Libraries and Frameworks

- OpenWhisk[2]: An open-source framework for self-deployed serverless architecture.

[1]NodeJS: `https://nodejs.org/en`
[2]OpenWhisk: `https://openwhisk.apache.org`

- Redis[3]: Open-source in-memory data structure store that has various saving strategies and replication is built-in by default.
- Docker-cli-js[4]: A CLI tool for managing docker containers via NodeJS.
- Express[5]: Back-end web application for NodeJS.
- HandlebarsJS[6]: Render library for NodeJS that can be used in conjunction with Express.

2 Server Setup

Setting up the code is fairly easy because NodeJS provides `ackage.json that can be used to instal` with just a single command. For refernce the file should look like Figure 1

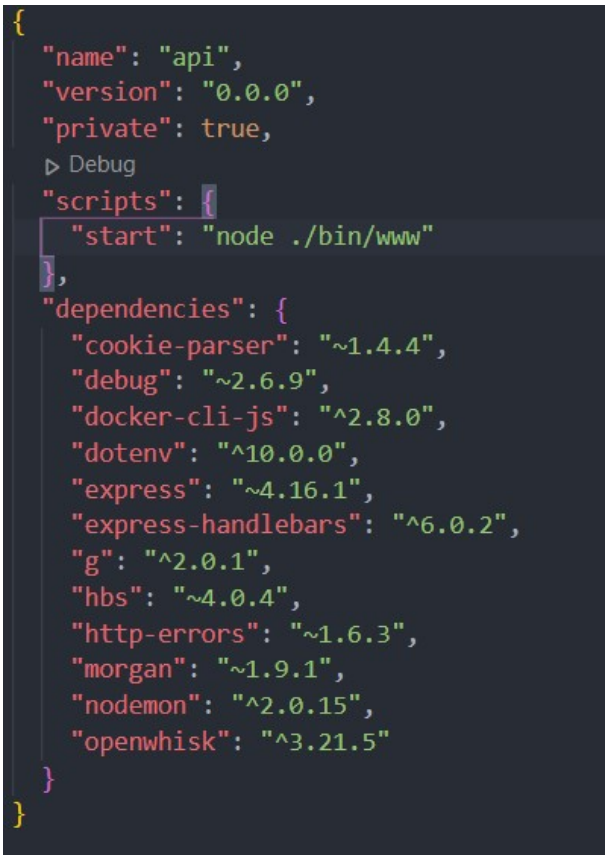

Figure 1: package.json file

Make sure that the system has NodeJS and NPM installed. If not, we can install it by the following commands:

```
sudo apt install nodejs
```

[3]Redis: `https://redis.io`
[4]Docker CLI (NodeJS): `https://github.com/Quobject/docker-cli-js`
[5]ExpressJS: `https://expressjs.com`
[6]HandlebarsJS: `https://handlebarsjs.com`

```
sudo apt install npm
```

Extract the core API code in a directory from the zip file: api.zip. Navigate to the api directory. Run the command:

```
npm install
```

Now that we have our core set-up, we need to make sure that OpenWhisk and Docker is deployed and running. Run the following commands to install the dependencies for both Docker and OpenWhisk:

```
sudo apt update -y && sudo apt upgrade -y
sudo apt install git python-pip python-setuptools /
    build-essential libssl-dev libffi-dev python-dev /
    software-properties-common
sudo pip install ansible
```

2.1 For Docker

Follow the steps below or follow the official documentation for Docker installation in Ubuntu[7]

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg lsb-release
curl -fsSL https://download.docker.com/linux/ubuntu/gpg |\
sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

echo \
  "deb [arch=$(dpkg --print-architecture) \
  signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] \
  https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Verify that Docker is up and running:

```
sudo docker run hello-world
```

2.2 For OpenWhisk

- Clone the repository: https://github.com/apache/openwhisk-devtools.git
- Navigate into openwhisk-devtools/docker-compose directory

[7]Docker installation Ubuntu: `https://docs.docker.com/engine/install/ubuntu`

- Before configuring OpenWhisk, we need to install two specific libraries e.g. zip and net-tools, otherwise, it will throw an error while building the NodeJS environment.
  - sudo apt install zip net-tools

Now after the dependencies are installed, we can start OpenWhisk with default settings using the command:

```
sudo make quick-start
```

The above command will build the necessary docker containers for the framework to work properly.

When the make command has finished the tasks, it will show an output with WSK_CONFIG_FILE variable. We need to copy and paste the line in our current shell to let the system know where to get the apihost and api_key. We can cat the .wskprops file and copy the variables as we will be needing those two for our UI inputs.

Along with WSK_CONFIG_FILE, the shell will also render the IP address or domain name where the OpenWhisk APIs are currently running. Copy it, as we will be needing that as well for the UI input.

2.3 Webserver

Navigate to api directory which we configured in [section], and run the command:

```
npm start
```

The command will host the application on localhost of the server on port 8001, if the post is already taken, we can change the port by modifying the PORT variable in the .env file on the root directory. The server should have a public domain where we could access the webserver on the specified port and the UI should look like Figure 2.

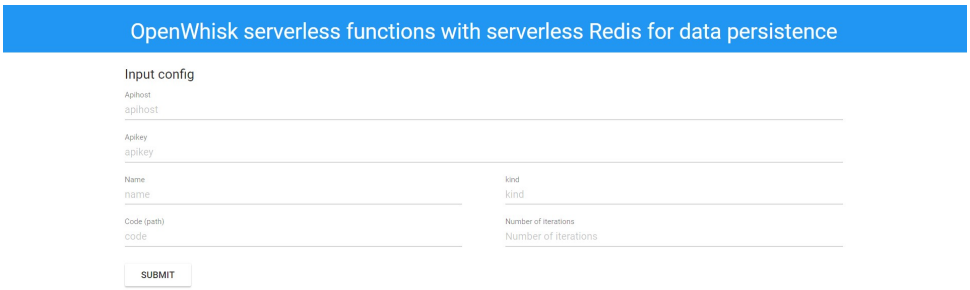

Figure 2: Webserver input page

The inputs on the UI are mentioned below:

- Apihost: IP or domain name of the default host of OpenWhisk API.

- Api_key: We would require a key provided by the OpenWhisk credentials manager, we can get the key from .wskprops file.
- Name: It will be an identifier for the action, we can get activation details using it. The activation details will hold the action output, duration, start time, end time, etc.
- Kind: Depending on the programming language, we need to provide a runtime so that OpenWhisk could deploy the action on a dependent Docker image.
- Code: Relative or absolute path of the zip file for the code.
- numberOfInvocation: This defines how many actions we would like to invoke parallelly.

After adding the parameters and submitting, the output will give us the details of our actions, which should look like Figure 3.

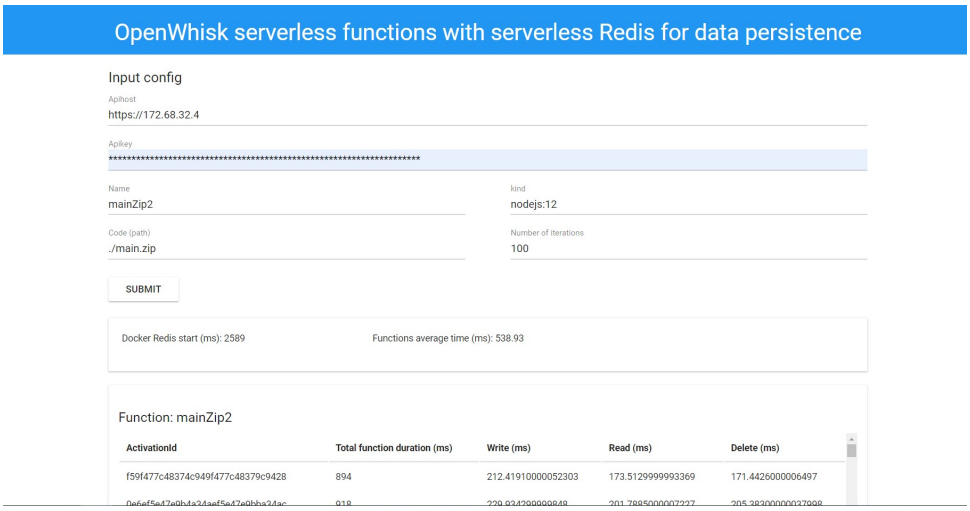

Figure 3: Webserver output

3 Code Explanation

Our core module exports a start function that will accept the parameters from the UI. The function has different tasks that return a promise, so that we could wait for the executions of each task to be finished before it goes on to the next. We need to have everything synchrously because we don't want to fire our actions before Redis container startup, else they will not be able to connect and start throwing errors. The flow of the execution is mentioned below with code snippet Figure 4:

- Initiate OpenWhisk configurations
- Create action with the given name parameter
- Start Redis container

```
async function start(params) {

    initOW(params)
    // await createAction(params)

    let startRedisTime = performance.now()
    await startRedis()
    dockerTime = performance.now() - startRedisTime

    // Invoking actions as per numberOfInvocation
    for (let index = 0; index < params.numberOfInvocation; index++) {
        let id = await invokeAction(params)
        activationID.push(id['body']['activationId'])
    }

    // Stop Redis offset
    setTimeout(() => {
        stopRedis()
    }, 5000)
}

    // Get the activations from their respective activation IDs
    for (let index = 0; index < activationID.length; index++) {
        let id = activationID[index] = await getActivation({
            name: activationID[index]
        })
        activations.push(id)
    }

    // Get the average time taken to complete the activations
    let activationAvg = await getActivationAverage(activations)

    return {
        numberOfInvocation: params.numberOfInvocation,
        activationAvg,
        dockerTime,
        activations
```

Figure 4: Code reference

- Loop through the numberOfInvocation parameter and fire the actions according to the number
- With a defined offset time, stop Redis
- Loop through activationID and get the results for each actions
- The function will return:
  - Activation duration average (activationAvg)
  - Docker startup time (dockerTime)
  - Activations JSON (activations)

The three actions mentioned in the report are api, setAction, and getAction are available in the zip file with their respective names. Just unzip the the files and change the Redis URL as per the setup.