Frankfurt University of Applied Sciences

Faculty 2 – Computer Science and Engineering
M.Sc. Program – High Integrity Systems

Cloud Computing

Crypto Currencies Prices Tracking Microservices Using Apache OpenWhisk

Authors
Group 10
Lam Phuoc Huy 1104785
Saifullah Saifullah 1339916
Marcel Sahillioglu 1367742

Supervisor
Prof. Dr. Christian Baun

January 28, 2021

Contents

1 Introduction

In this project, we created a Crypto Currencies Prices Tracker Microservices using Apache OpenWhisk [1], an open-source, distributed Serverless platform. This platform enables users to execute user-defined functions without worrying about operational issues [2]. For instance, the management of resources such as the availability of infrastructure and other computing components are not the responsibility of the user. There are multiple platforms available for the implementation of the Function-as-a-Service (FaaS) like AWS Lamda, Google Cloud Function, Microsoft Azure Functions, and Oracle Cloud Functions. In this project, we decided to use IBM Cloud Functions platform to serve as a deployment infrastructure for our services.

The remainder of the report is organized as follows. The next section gives a glance at our project goal and objectives. In Section 3, we take a look at Apache OpenWhisk and Serverless Computing technology. A complete setup and installation guide to deploying Apache Openwhisk locally is addressed in Section 4. IBM Cloud and IBM Cloud Command-Line Interface (CLI) will be shown in Section 5. After that, our Microservices will be covered in Section 6. Finally, conclusions and future research ideas are given in Section 7.

2 Project Goal and Objectives

The purpose of this semester project is to comprehend the theoretical and practical knowledge regarding cloud computing. Every group has to opt-in for a cloud-related topic, design and deploy then present their learning outcomes by writing a comprehensive report. The goal of this project is to study the available options and infrastructures to deploy cloud-related software. After deployment, we analyze all the components and explain how to use these technologies.

As mentioned previously, we have selected to work with Apache Openwhisk. First, we install the "Standalone" Apache OpenWhisk stack locally on our computer to learn about FaaS as well as OpenWhisk syntax. We will explain more about this in Section 4. Our plan is to write a function that can help us track the price of the top 5 crypto currencies daily then email these prices to us. After development, we have decided to deploy our function on the IBM Cloud Function platform, as it also based on Apache OpenWhisk technology.

The responsibility for the project can be distributed as:

- Lam Phuoc Huy: Team leader, in charge of function development and documentation.
- Marcel Sahillioglu: Programmer, in charge of integration and deployment on IBM Cloud Function, documentation.
- Saifullah Saifullah: Research, testing, documentation.

List of tools and software our team used during the development: **Operating System**: Ubuntu 20.04 LTS; **Functions platform**: Apache OpenWhisk; **Deployment platform**: IBM Cloud Functions; **Communication**: Whatsapp; **Code management**: Github; **Diagram creation**: draw.io.

3 Apache OpenWhisk and Serverless Computing

3.1 Serverless Computing

Serverless computing or just serverless is a method of running applications without concerns about the provisioning of computer servers or any of the compute resources. It is a partial realization of an event-driven idea, in which applications are defined by Actions and the Events that are used to trigger them [3]. However, behind the scenes, compute servers are most definitely involved. Serverless is similar to the wireless Internet where the wires exist, but they are not visible to end-users. Functions designed and deployed on serverless platforms provide the complete definition of the actions through simple function abstractions and also build the logic for event processing. IBM strongly focused on these concepts in their OpenWhisk platform, where functions could easily be defined in terms of the event, trigger and actions [3].

In addition, serverless computing needs compute resources to run applications, but the server management and capacity planning decisions are completely abstracted from developers and users. In this type of computing, we generally write applications/functions that focus on one particular task. We then upload that application on the cloud provider, which gets invoked via different events, such as HTTP requests, webhooks, etc. All major cloud providers like AWS, Google Cloud Platform, or Microsoft Azure provide serverless offerings.

3.2 Apache OpenWhisk

It is an open-source, distributed server-less platform that executes functions in response to events at any scale. It manages the infrastructure, servers, and scaling using Docker containers so you can focus on building amazing and efficient applications [4].

The OpenWhisk platform supports a programming model in which developers write functional logic generally referred to as Actions, in any supported programming language, which can be dynamically scheduled and run in response to associated events known as triggers from external sources or from HTTP requests. The project includes a REST API-based CLI along with other tools to support packaging, catalog services, and many popular container deployment options. Some of the main features of this platform which are listed by the official document are review below:

- **Deploys anywhere**: As this platform uses container-based components, Openwhisk supports many choices for deployment both locally and within Cloud infrastructures.
- **Write Functions in any Programming Language**: It supports a wide list of programming languages such as NodeJS, Java, Scala, PHP, Python, and many more. If there is any platform or language which is not supported, one can easily create and customize own executables that can run on the Docker runtime by using Docker SDK.
- **Integration Support**: It is very easy to integrate developed Actions with many popular services using packages that are provided either as independently developed projects or part of default catalog. For instance, packages offer integration with general services such as Kafka message queues, databases, mobile applications, messaging services or Really Simple Syndication (RSS) feeds.
- **Combining Functions into rich Composition**: Functions written in different programming languages can run as a packaging code with Docker. It is feasible to invoke code syn-

chronously, asynchronously, or on a schedule. It is recommended to use parameter binding to avoid hardcoding service credentials in one's code.

- **Scaling and Optimal Utilization**: Run developed Action thousands of times in a fraction of a second, or once a week. The instances scale easily to meet demand as needed and stop when not in usage. It provides optimal utilization of resources, so one could pay bills when the service is in use, for idle resources, there is no need to pay any penny.

4 Setup and Installation Standalone OpenWhisk

It is important to note that after doing some research, our team concluded that it is easier to set up the "Standalone" version of OpenWhisk on Linux. Our team chooses Ubuntu version 20.04 LTS because it is one of the top Linux Distribution in recent times. If you want to install OpenWhisk locally on Windows 10 we suggest that you use Windows Subsystem for Linux (WSL) [5]. Our setup is based on instructions found on OpenWhisk GitHub Repository [6] and OpenWhisk Documentation [4].

Alternatively, it is also possible to study OpenWhisk technology using IBM Cloud Functions [7]. IBM Cloud Functions provide a web-based version of OpenWhisk with modern, easy to use Graphical User Interface (GUI). However, our team believes that having a local installation of OpenWhisk helps you to understand more about how to set up a development environment, and getting used to working with CLI will help you in the long run.

4.1 Requirements

The easiest way to start learning OpenWhisk is to install the "Standalone" OpenWhisk stack. This is a full-featured OpenWhisk stack running as a Java process for convenience. Serverless functions run within Docker containers. You will need Docker, Java, and Node.js available on your machine.

Open the Terminal and check the availability of Docker, Java, and Node.js on your computer using these commands below. When typing in Terminal you need to remember that it is both space and case sensitive so pay close attention to avoid typo. If you are using Ubuntu 20.04 LTS, both Java and Node.js should have already been installed on your system.

```
java -version
node -v
docker version
```

To set up Docker Engine on Ubuntu, we install using the repository method following the instructions here [8]. First, make sure there are no old versions of Docker. Older versions of Docker were called docker, docker.io, or docker-engine. If these are installed, uninstall them:

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

4.1.1 Set Up The Repository

1. Update the apt package index and install packages to allow apt to use a repository over HTTPS:

```
sudo apt-get update
sudo apt-get install \
apt-transport-https \
ca-certificates \
curl \
gnupg-agent \
software-properties-common
```

2. Add Docker's official GPG key:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Verify that you now have the key with the fingerprint 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88, by searching for the last 8 characters of the fingerprint.

```
sudo apt-key fingerprint 0EBFCD88
```

3. Use the following command to set up the stable repository.

```
sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"
```

4.1.2 Install Docker Engine

1. Update the apt package index, and install the latest version of Docker Engine and containerd, or go to the next step to install a specific version:

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

2. To install a specific version of Docker Engine, list the available versions in the repo, then select and install:
a. List the versions available in your repo:

```
apt-cache madison docker-ce
```

b. Install a specific version using the version string from the second column, for example, 5:18.09.1 3-0 ubuntu-xenial

```
sudo apt-get install docker-ce=<VERSION_STRING> \
docker-ce-cli=<VERSION_STRING> containerd.io
```

3. Verify that Docker Engine is installed correctly by running the hello-world image.

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

4.2 OpenWhisk Quick Start

After all the requirements are met, use these commands to download and run OpenWhisk on your system:

```
git clone https://github.com/apache/openwhisk.git
cd openwhisk
sudo ./gradlew core:standalone:bootRun
```

From our experience, the running process will be stuck at 96% EXECUTING. This is not an error, it just means that for some reason OpenWhisk process can't automatically open browser to a functions Playground. The Playground allows you create and run functions directly from your browser. Manually open a browser of your choice and access the link below:

```
http://172.17.0.1:3232/playground/ui/index.html
```

To make use of all OpenWhisk features, you will need the OpenWhisk command line tool called wsk which you can download from here [9]. Please download the correct version for you system. For example, if your OS is 64 bits then please choose the file that end with -linux-amd64.tgz. After the download is finished, you can extract it and move the wsk file to openwhisk/bin. Please refer to the CLI configuration [10] for additional details. Typically you configure the CLI for Standalone OpenWhisk as follows:

wsk property set –apihost 'http://172.17.0.1:3233' –auth '23bc46b1-71f6-4ed5-8c54-816aa4f8c502:123zO3xZCLrMN6v2BKK1dXYFpXlPk ccOFqm12CdAsMgRU4VrNZ9lyGVCGuMDGIwP'

5 IBM Cloud Functions and IBM Cloud Functions CLI

5.1 Intro to IBM Cloud

Even though the aim of our project was to work with Apache Openwhisk we decided to use IBM Cloud Functions instead since it is based on Apache Openwhisk and the functionality is compatible. IBM Cloud [11] itself is a platform which offers a lot of cloud services. Among these are for example IBM Cloud Functions, Cloud Foundry, Kubernetes, OpenShift, etc. Due to the fact that we want to create a function as a service, we only consider IBM Cloud Functions in this project.

5.2 Getting Started with IBM Cloud Functions

In order to work with IBM Cloud one needs to have an IBM Cloud Account [12]. For students of a university setting up an account is for free and very easy.

Figure 1 displays the homescreen of the IBM Cloud Functions environment after creating an account.

The most important menu tabs are the introduction, Actions, triggers, APIs and monitoring. In order to create a function as a service we need to create a new action. An action does not differ from a simple function. Actions can be coded in Node.js, Python, Ruby, Swift, PHP or Go. It needs to have an individual name and has to be assigned to an enclosing package. This comes in handy if the

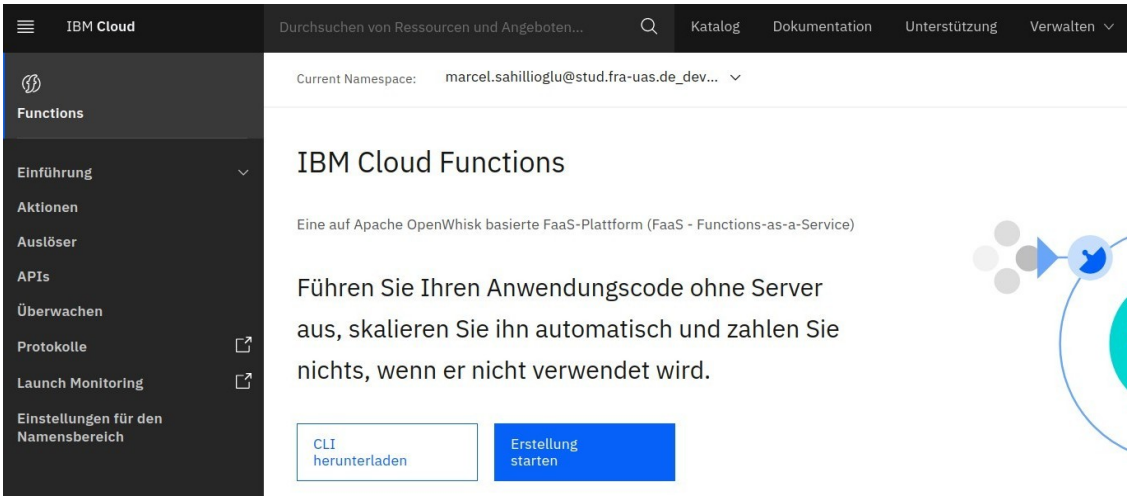


Figure 1: Homescreen IBM Cloud Functions

action needs to invoke another one. By default, the selected enclosing package is the *Default Package*.

After those steps the development environment should be visible like depicted in figure 2. In this example the Action's name is *test*. The name of the enclosing package is *CloudComp*. The chosen programming language for this example is Node.js 12.

5.3 IBM Cloud Functions CLI

To install IBM Cloud CLI on your system, please take a look at this tutorial [13]. First, open the Terminal and run this command:

curl -sL https://raw.githubusercontent.com/IBM-Cloud/ibm-cloud-developer-tools/master/linux-installer/idt-installer | bash

To verify that the CLI and Developer Tools were installed successfully, run the help command:

```
ibmcloud dev help
```

After that we need to log in to IBM Cloud:

```
ibmcloud login -a cloud.ibm.com
```

We then install the Cloud Functions plug-in:

```
ibmcloud plugin install cloud-functions
```

To target a name space:

```
ibmcloud target --cf
```

Finally to test if everything working correctly:

```
ibmcloud fn list
```

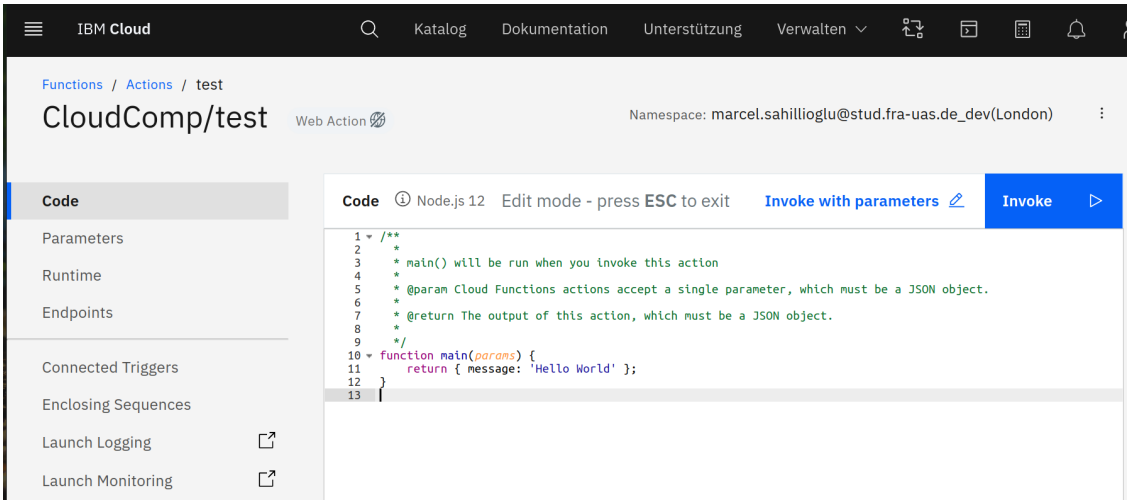


Figure 2: Development Environment

5.4 Working with the IBM Cloud Functions CLI

Everything the user can do within the browser user interface of the IBM Cloud Functions, e.g. creating actions, invoking them, etc. can also be done with the IBM Cloud CLI. The following subsections will cover a few examples which were relevant to our project. For more details please check the IBM Cloud documentation [14] . There are a lot of possibilities that would be too extensive for this report.

Every time after logging into the IBM account via the CLI one needs to specify the *target space* and *target organization*. Otherwise, it will not be possible to invoke an action. Logging into the account and setting *target space* and *target organization* interactively was already explained in section 5.3.

5.4.1 Actions with the CLI

In order to list all actions already created within the IBM account use the following command:

```
ibmcloud fn action list
```

The output should look as displayed in figure 3. If the command is run without *action* in it, like in section 5.3, everything that was created within the IBM Cloud will be listed. This means all packages, actions, triggers, rules, etc.

To invoke an existing action use the following command withe the name of the enclosing package it was created in and the name of the action:

```
ibmcloud fn action invoke ENCLOSING_PACKAGE/ACTION_NAME
```

It is possible to create, delete, invoke, list and update an action as well as get information about it. For more information run the command:

```
marcel@marcel-Inspiron-7370:~$ ibmcloud fn action list
actions
/marcel.sahillioglu@stud.fra-uas.de_dev/CloudComp/Invoke_2              private nodejs:12
/marcel.sahillioglu@stud.fra-uas.de_dev/CloudComp/SendMail              private nodejs:12
/marcel.sahillioglu@stud.fra-uas.de_dev/CloudComp/CryptoTrack           private python:3.7
/marcel.sahillioglu@stud.fra-uas.de_dev/CloudComp/test                  private nodejs:12
/marcel.sahillioglu@stud.fra-uas.de_dev/monitoring_httpValue/monitor_value private nodejs:10
/marcel.sahillioglu@stud.fra-uas.de_dev/hello-world-ibmcf/helloworld    private nodejs:10
/marcel.sahillioglu@stud.fra-uas.de_dev/get-http-resource/location      private nodejs:10
```

Figure 3: Action List

```
marcel@marcel-Inspiron-7370:~$ ibmcloud fn trigger list
triggers
/marcel.sahillioglu@stud.fra-uas.de_dev/trigger                         private
/marcel.sahillioglu@stud.fra-uas.de_dev/12AM trigger                    private
/marcel.sahillioglu@stud.fra-uas.de_dev/15min trigger every day         private
/marcel.sahillioglu@stud.fra-uas.de_dev/15min trig                      private
/marcel.sahillioglu@stud.fra-uas.de_dev/15min trigger                   private
/marcel.sahillioglu@stud.fra-uas.de_dev/value_trigger                   private
marcel@marcel-Inspiron-7370:~$
```

Figure 4: Trigger List

```
wsk action
```

while being logged in.

5.4.2 Triggers with the CLI

The trigger is the other important feature within our project. Using the CLI working with triggers is very similar to working with actions. It is possible to create, delete, fire, get and update a trigger. Furthermore, all triggers can be listed by the following command which will lead to the output depicted in figure 4.

```
ibmcloud fn trigger list
```

To fire a trigger run the following command. Make sure that if the name of your trigger includes spaces you need to put the name of the trigger in quotation marks. Output displayed in figure 5.

```
ibmcloud fn trigger fire '12AM trigger'
```

```
marcel@marcel-Inspiron-7370:~$ ibmcloud fn trigger fire '12AM trigger'
ok: triggered /_/12AM trigger with id 8f482051e9254a15882051e9258a156d
```

Figure 5: Fired Trigger

Figure 6: *Connected Triggers*

6 Crypto Currencies Prices Tracker Microservices

6.1 Our Microservices Introduction

The aim of our program is to periodically check the exchange rates of five different crypto currencies through a REST API and send us an email with those values. Before that we experimented with the exchange rate of one crypto currency, namely Bitcoin, by comparing it to a defined threshold. If this threshold was surpassed we sent us the mail with the value. To spice things up a little we decided to further develop the program by sending us the five currencies as mentioned before. The explanation of the program will refer to the version sending five exchange rates. The changes to the previous version will not be explained in detail.

We therefor implemented three actions. The first one gets invoked via a periodic trigger which fires at a specific time. It then invokes the second action that gets the exchange rates via the API and converts them into the data type we want. The values will be passed back to the first action. Lastly the third action will be called by the first one with those parameters in order to send the email.

6.2 Project Actions

6.2.1 Trigger

Attaching a periodic trigger to an action is very straighforward. IBM Cloud Functions provides multiple kinds of triggers. Like displayed in figure 2 one can simply press the tab "Connected Triggers" and let the action be triggered by different events. An action can be triggered whenever objects in a certain bucket within a Cloud Object Storage are updated, a certain Cloudant database is updated or a push notification is sent to your mobile app just to give a few examples.
In this project we focused on the periodic trigger. The shortest time frame in which two consecutive triggers can fire periodically is every minute. One can choose on which days, hours and minutes the trigger should invoke the action. Once connected to the action it is possible to enable and disable the trigger (figure 6).

6.2.2 1st Action *Invoke_2*

Our first periodically invoked action with the name *Invoke_2* invokes the other actions and either gets or passes parameters. It is displayed in figure 7 and its' runtime is Node.js.

To work with OpenWhisk related functions we need to include the OpenWhisk module and create an instance like in line 3 and 4. From lines 9 to 16 we invoke our second action. We need to specify the path of the action we want to invoke within IBM Cloud Functions, meaning we need

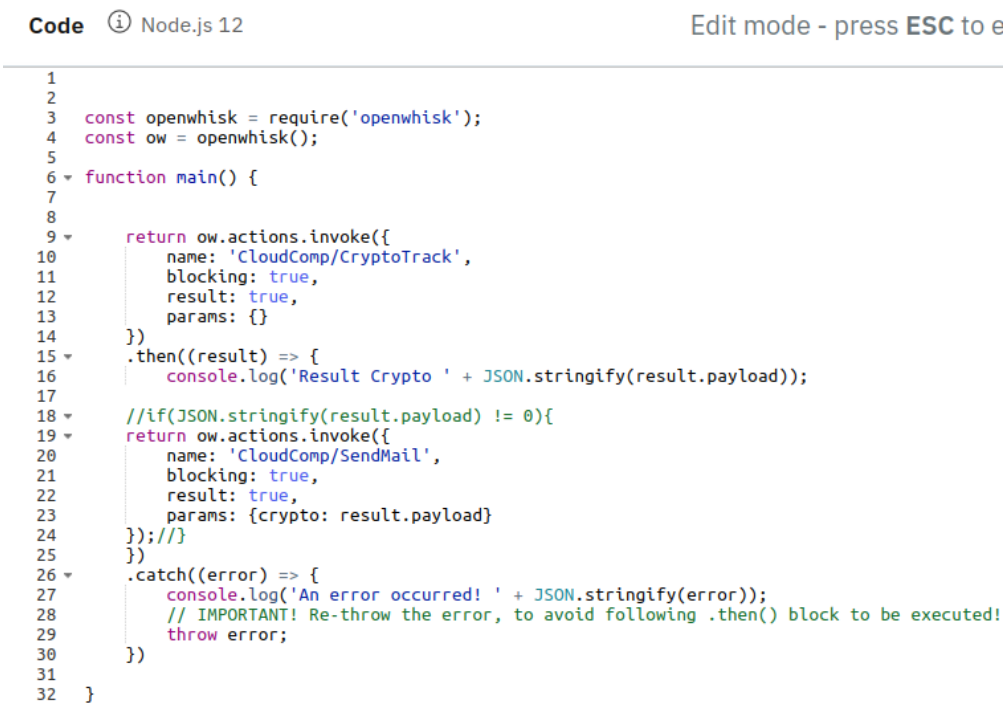


Figure 7: *Invoke_2*

to know the name of our action and in which enclosing package we created it. In our example the enclosing package's name is *CloudComp* and the action's name is *CryptoTrack*. Note that the *blocking* and *result* flag are set true to wait for the function to get a result. Since there is no way of debugging within IBM Cloud we displayed us the result from *CryptoTrack* in the console window as a string.

We invoke the third action *SendMail* the same way with one exception. Since we get an array containing five float numbers, we need to pass them into our *SendMail* action as a parameter. Therefor one needs to fill the *params* bracket with a name, here we called it *crypto*, and the payload we got back as a result from our second action.

6.2.3 2nd Action *CryptoTrack*

Our *CryptoTrack* action is depicted in figure 8.

In order to get the data of the exchange rates we use the API of *nomics.com* [15]. One needs to sign up for free in order to get the key and access the data of the platform. In our first version we only got the Bitcoin exchange rate and compared it to the threshold we defined (line 29). If it was surpassed we returned it within the payload, if not we just returned 0 in our payload. This way we could send or not send us an email simply by checking if the payload was different from 0.
The final version implements an array of urls containing five elements. The aim of the *get_jsonparsed_data()* function is to open the url and convert the data from a JSON file into a Python object since this function is implemented in Python. The price of each crypto currency is then available for us as a string, but still contains symbols we need to remove, namely backslashes. Also we typecast the

Figure 8: *CryptoTrack*

strings into floats. Finally we return our array containing the five float values.

6.2.4 3rd Action *SendMail*

Figure 9 shows our third and last action. Its purpose is to send a mail from one address to another. Besides *Google Mail* the service we use for this is *nodemailer* [16] so we need to include the module into our action.

Code ⓘ Node.js 12 Edit mode -

```
const nodemailer = require('nodemailer');


async function main(params){


let transport = nodemailer.createTransport({
    host: 'smtp.gmail.com',
    port: 465,
    secure: true,
    auth: {
        user: 'sahillioglu12@googlemail.com',
        pass: 'xxxx xxxx xxxx xxxx'
    }

});

var mailOptions = {
  from: 'sahillioglu12@googlemail.com',
  to: 'marcel.sahillioglu@stud.fra-uas.de',
  subject: 'Sending Email using Node.js',
  text: '1 Bitcoin = ' + JSON.stringify(params.crypto[0]) + ' USD.\n' +
        '1 Ethereum = ' + JSON.stringify(params.crypto[1]) + ' USD.\n' +
        '1 Thether = ' + JSON.stringify(params.crypto[2]) + ' USD.\n' +
        '1 Polkadot = ' + JSON.stringify(params.crypto[3]) + ' USD.\n' +
        '1 Ripple = ' + JSON.stringify(params.crypto[4]) + ' USD. '
};


let info = await transport.sendMail(mailOptions);

console.log("Message sent: %s", info.messageId);

}
```

Figure 9: *SendMail*

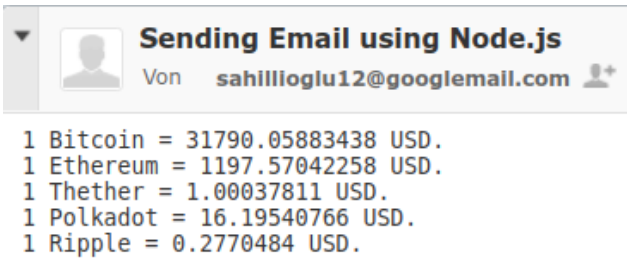


Figure 10: *Resulting Mail*

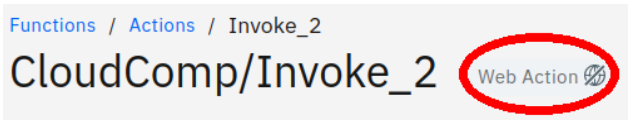


Figure 11: *Step 1*

We need to make *main* an *async* function like we will explain later in secion 6.4. Furthermore *main* gets parameters from the *Invoke_2* action. Using *nodemailer* we are able to create a Simple Mail Transfer Protocol (SMTP) transport instance containing the host, port number, a secure flag and the credentials of the sending mail address. Next, the mail options have to be specified meaning one needs to specify the sender mail address as well as the receiver address, subject of the mail and the text of course. The different values of the exchange rates are accessible via the float array which is passed as a parameter to the action like explained in section 6.2.2. Lastly the mail is send using the *sendMail* function of the transport instance with the specified mail options.
The final mail is depicted in figure 10.

6.3 Enabling Action as Web Action

With IBM Cloud Functions it is possible to enable any action as a web action. This makes it possible to invoke the action via an url. So invoking an action only requires a web browser and can be invoked without logging into the account or any further steps.

- Open the action you want to enable as web action and click on "Web Action" like displayed in figure 11
- Check the box "Enable as Web Action"
- As soon as this is done an url will be visible (see fig 12) which can be copied. Entering this link will invoke the action.

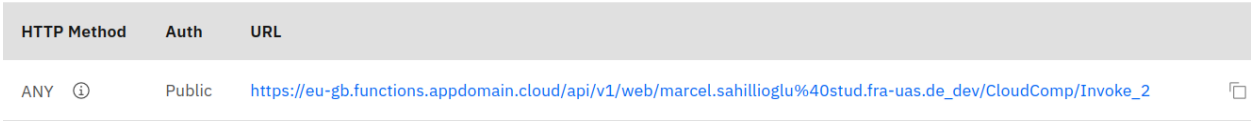
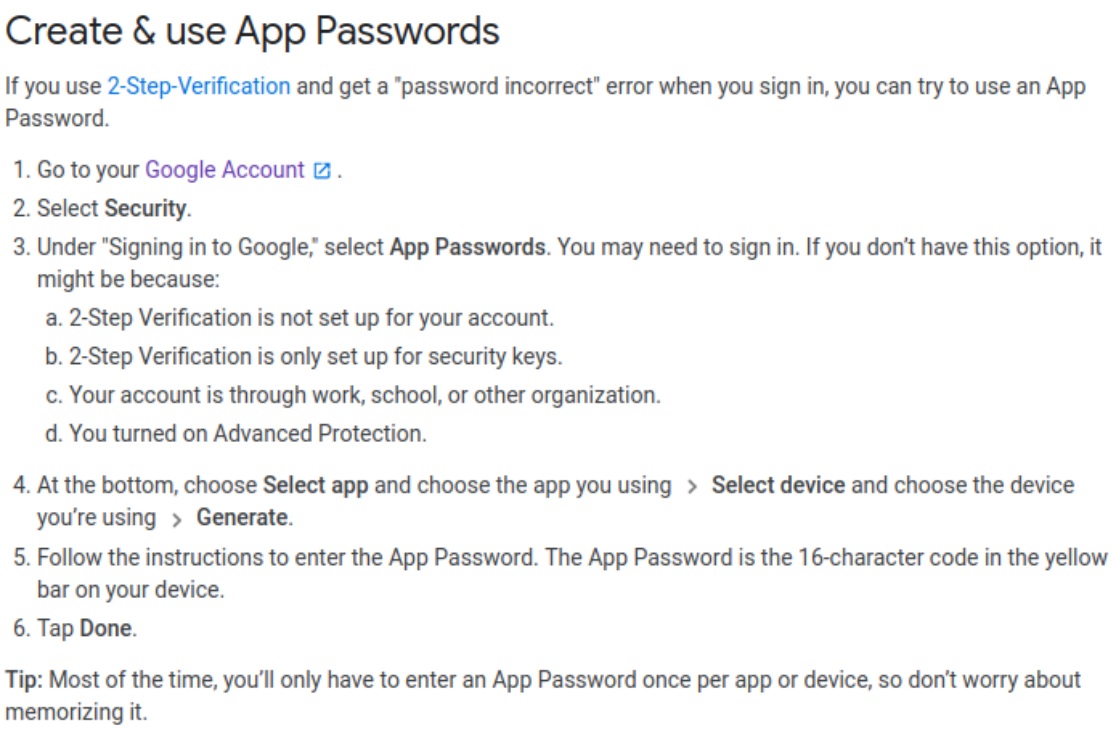


Figure 12: *Step 3*



Create & use App Passwords

If you use 2-Step-Verification and get a "password incorrect" error when you sign in, you can try to use an App Password.

1. Go to your Google Account .
2. Select **Security**.
3. Under "Signing in to Google," select **App Passwords**. You may need to sign in. If you don't have this option, it might be because:
   a. 2-Step Verification is not set up for your account.
   b. 2-Step Verification is only set up for security keys.
   c. Your account is through work, school, or other organization.
   d. You turned on Advanced Protection.
4. At the bottom, choose **Select app** and choose the app you using > **Select device** and choose the device you're using > **Generate**.
5. Follow the instructions to enter the App Password. The App Password is the 16-character code in the yellow bar on your device.
6. Tap **Done**.

Tip: Most of the time, you'll only have to enter an App Password once per app or device, so don't worry about memorizing it.

Figure 13: *App Password*

6.4 Encountered Problems

Of course we encountered some problems implementing our actions. One of them was the challenge to invoke one action via another. Even though there is some documentation to be found in the internet not everything worked the way it was described. We had to experience a bit aroud until it was finally working the way we wanted it to since a lot of the documentation is refering to IBM Bluemix, which is basically the older version of IBM Cloud. Passing the parameter to the action we want to invoke was also not that simple. Like depicted in figure 7 line 23 it is not possible to just pass *result.payload*. We need to give the parameter a name in order to pass it to another action and work with it within that action like in figure 9 lines 22 to 26.

The challenges we encoutered within the *CryptoTrack* action were mainly converting between data types since this was our only action written in Python. Conversions between JSON files and Python objects had to be made. Also the data of one crypto currency was very huge. One is able to get a lot of data out of one crypto currency besides the price we focused on.

Most of our problems were caused by the *SendMail* action. The action did not work as long as no app password was specified. It differs from your ordinary *Google* account password. Following the six steps provided by *Google* and displayed in figure 13 the action will be able to work. The password needs to be put in like shown in figure 9 line 13. It is possible that access for less secure apps also has to be enabled. One can find this option within the security settings of the *Google* account.

It was necessary to use the *await* keyword in line 30 since we needed to wait for the connection to be established and for the mail to be sent otherwise the action will not wait for this to happen causing it to not deliver a mail by invoking it once. One needed to invoke it multiple times in a short time frame to finally send the mail and the mail was then sent multiple times which was unwanted behavior.
Lastly one challenge was to access the data inside the parameter we pass to the action (lines 22 to 26). One needs to refer to it by the name it received within the *Invoke_2* action (line 16).

7 Conclusion

In this report, we explained what the Apache OpenWhisk platform is and how to set up and run it locally. After that, we learned about IBM Cloud Functions, a commercial product of IBM based on OpenWhisk technology. Using IBM Cloud Functions, we develop and deploy a microservices that can help us track the prices of crypto currencies through Email. Even though there are things that can be improved like using a different Email API to avoid messing around with Goggle Email, we can say that our final product satisfies all the use cases initially planned. Through this project, our team members learned a lot about different cloud technologies and many cloud deployment platforms.

A Source code

Crypto Currencies Prices Tracking Microservices Repository: https://github.com/NaginataAI/CloudWS2020
Last accessed date: 27-01-2021

References

[1] "Apache openwhisk homepage." `https://openwhisk.apache.org`. Accessed: 25-1-2021.

[2] S. Quevedo, F. Merchán, R. Rivadeneira, and F. X. Dominguez, "Evaluating apache openwhisk-faas," in *2019 IEEE Fourth Ecuador Technical Chapters Meeting (ETCM)*, pp. 1–5, IEEE, 2019.

[3] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 405–410, IEEE, 2017.

[4] "Openwhisk documentation." `https://openwhisk.apache.org/documentation.html`. Accessed: 25-1-2021.

[5] "Install window subsystem for linux." `https://docs.microsoft.com/en-us/windows/wsl/install-win10`. Accessed: 27-1-2021.

[6] "Github - apache/openwhisk." `https://github.com/apache/openwhisk`. Accessed: 25-1-2021.

[7] "Cloud functions overview | ibm." `https://www.ibm.com/cloud/functions`. Accessed: 27-1-2021.

[8] "Install docker on ubuntu." https://docs.docker.com/engine/install/ubuntu. Accessed: 27-1-2021.

[9] "Open whisk cli." https://s.apache.org/openwhisk-cli-download. Accessed: 27-1-2021.

[10] "Open whisk cli configuration." https://github.com/apache/openwhisk/blob/master/docs/cli.md. Accessed: 27-1-2021.

[11] "Ibm cloud." https://cloud.ibm.com/login. Accessed: 25-1-2021.

[12] "Sign up for ibm cloud." https://cloud.ibm.com/registration. Accessed: 25-1-2021.

[13] "Getting started with the ibm cloud cli." https://cloud.ibm.com/docs/cli/reference/bluemix_cli/download_cli.html. Accessed: 27-1-2021.

[14] "Cloud functions cli." https://cloud.ibm.com/docs/cloud-functions-cli-plugin?topic=cloud-functions-cli-plugin-functions-cli. Accessed: 27-1-2021.

[15] "Crypto market caps- prices, all-time highs, charts." https://nomics.com. Accessed: 25-1-2021.

[16] "Nodemailer." https://nodemailer.com/about. Accessed: 25-1-2021.