Proceedings Fall2019 Engineering Cloud Computing

Gregor von Laszewski

Editor

laszewski@gmail.com

https://cloudmesh-community.github.io/book/vonlaszewski-proceedings-fa2018.epub

August 10, 2019 - 11:01 AM

Created by Cloudmesh & Cyberaide Bookmanager, https://github.com/cyberaide/bookmanager

Proceedings Fall2019 Engineering Cloud Computing

Gregor von Laszewski

Proceedings Fall2019 Engineering Cloud Computing

1 PROJECTS

1.1 MANAGE FILES ACROSS CLOUD PROVIDERS

Richa Rastogi
rirastog@iu.edu
Indiana University
hid: fa18-516-18
github:

Keywords: Multi-cloud data service, Cloud Computing, Python, Open API, Cloud Providers, MongoDB, Swagger

1.1.1 Abstract

The goal of this project is to manage files across different cloud providers. There are many cloud providers where we can store data in form of files like Amazon AWS, Microsoft Azure, Google cloud, etc. Here we are going to build an OpenAPI to manage these files, operations like copy, upload, download or delete from any provider. This system is self sufficient to work as a file manager.

1.1.2 Introduction

The objective of this project is to manage data across different cloud providers. We are going to build an RESTFUL OpenAPI for managing the data between all the cloud storages. We will analyse how these different clouds work and then build python methods to handle data across them. Final step will be to expose these functionalities as a RESTFUL API. This way we can also take advantage of cloud providers for cheaper solutions for storage by dividing the data across them. Since this project has its own MongoDB and User profiling so it can be used a file manager in itself.

1.1.3 Requirements

This project requires knowledge about Cloud Providers like AWS, Azure, Google Cloud, etc. * This project is using Amazon AWS and Google cloud as two cloud providers and their storage functionality. We can expand this * This also needs a database so we are using MongoDB through MongoEngine and store the files and User data in it. * Overall functionality can be accesses through console or RESTFUL OpenAPI which is built using Flask and Swagger.

1.1.4 Architecture

Please refer to below diagram #fig:fa18-516-18_Architecture.png for architecture of this project.

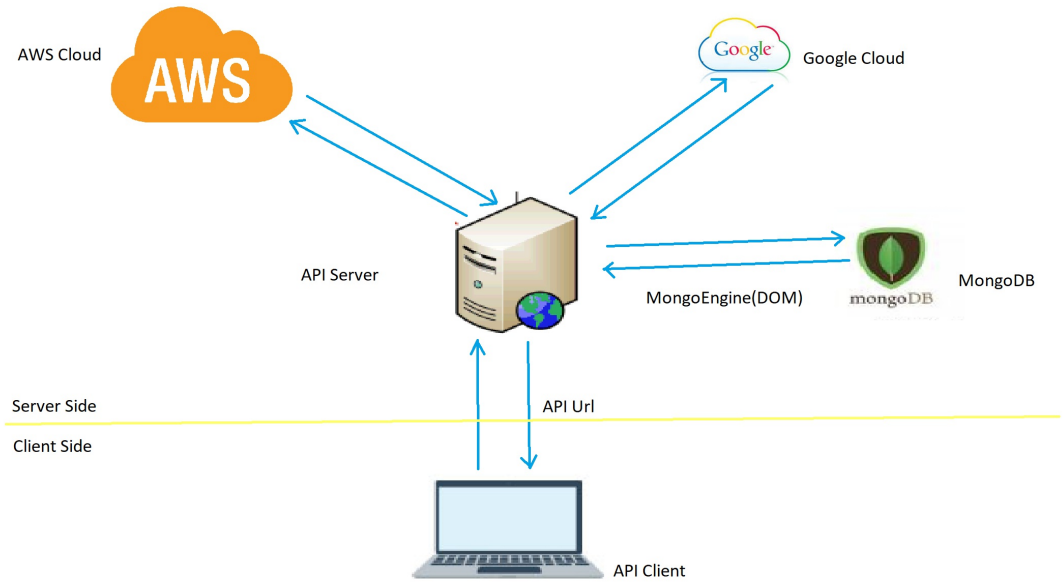

Figure 1: Architecture

1.1.5 Design

This project involves developing a RESTFUL API to manage files. We can perform following operations like upload, download, list, copy, rsync and delete. We can also this project to store files in MongoDB and assign specific User Role permissions to access the files.

- The very first thing required for that is to create accounts in AWS and Google(Since those are the two providers we are using).
- Then use their API Keys to connect to these providers through Python code.
- After we have a connection, we use their APIs to access the bucket by providing the name through Yaml file/API input.
- Now since we have the bucket, we can list, download or upload the files.
- There is Command console from where we can execute these functions directly using console or script file.
- On top of that there is an open API built to perform these functions using REST.
- We also have MongoDB storing the downloaded files.

1.1.6 Implementation

This project is using following technologies for implementation: * Python 3.7.0 for python code development * Swagger 2.0 for writing API specification. This specification describes REST endpoints for managing files across providers. * Python flask framework which consumes the OpenAPI specification and directs the endpoints to Python functions by building a RESTful app. * MongoEngine as a Document-Object Mapper for working with MongoDB from Python.

Enable a virtual environment so that all installations can be done specifically in that env.

```
python3 -m virtualenv /home/richa/venv/    //to install venv
source /home/richa/venv/bin/activate        //to activate venv

Now my console looks like:

(venv) (3.7.0) richa@richa-VirtualBox:~$
```

1.1.6.1 AWS access from Python:

- Install apache-libcloud by “pip install apache-libcloud”
- Follow instructions to create an AWS account - https://github.com/cloudmesh-community/book/blob/master/chapters/iaas/aws/aws.md
- Select S3 from Services and create a bucket
- To access this bucket, go to IAM and create a user and then create a new Access Key in “Security Credentials”
- While creating this key, system will promt to download pem file. Save that pem file onto your working machine.

1.1.6.2 Google Cloud Platform:

- Install “pip install google-cloud-storage”
- pip install google-auth google-auth-httplib2
- pip install –upgrade google-api-python-client
- Create an account on Google Cloud by going to https://cloud.google.com/
- Create a new Project from the top of the page.
- Create a new storage bucket in google cloud, select Storage -> Storage -> Browser
- To access this bucket now, follow https://cloud.google.com/storage/docs/reference/libraries

- This will download a JSON file in your working VM and use that file for authentication to access Google Cloud Storage.

All the dependencies can be installed easily by running requirements.txt inside project-code so no need to do any pip install.

```
pip install -r requirements.txt
```

AWS and Google Cloud specific functionality python files are under directory structure project-code/cloudmesh/data. cloudmesh-data.yaml is the yaml file holding all the information about these cloud setup. Aws_setup.py and google_cloud_setup.py uses this yaml file to authenticate the cloud providers and setup the connection to the cloud services.

It also has command.py under here to run the functionality from console passing in relevant input. To execute commands from console using cmdata commands, we need to setup cmdata by running in project-code dir:

```
pip install .
```

Now we can run all cmdata commands as given below. We can also test if cmdata is working by running a test command:

```
 cmdata test

Usage:
  cmdata data list [--format=FORMAT]
  cmdata set provider=PROVIDER
  cmdata set dir=BUCKET
  cmdata data add PROVIDER FILENAME
  cmdata data get PROVIDER FILENAME USER_UUID
  cmdata data ls PROVIDER
  cmdata data copy FILENAME PROVIDER DEST
  cmdata data rsync FILENAME SOURCE DEST
  cmdata data del PROVIDER FILENAME
  cmdata update user USER file FILENAME
  cmdata (-h | --help)
  cmdata --version

Options:
  -h --help     Show this screen.
  --version     Show version.
  --config      Location of a cmdata.yaml file

Description:

    cmdata data ls PROVIDER

        Description: CM command to list all the files in a Provider's bucket

    cmdata data add PROVIDER FILENAME

        Description: CM command to upload a file from local directory to the Provider's bucket

    cmdata data get PROVIDER FILENAME USER_UUID

        Description: CM command to download a file from the Provider's bucket to a local directory
```

```
        and then save that file to MongoDB with the username assigned

    cmdata data copy FILENAME PROVIDER DEST

        Description: CM command to copy a file from one Provider's bucket to another

    cmdata data del PROVIDER FILENAME

        Description: CM command to delete a file from a Provider's bucket

Example:
   cmdata data ls google_cloud
   cmdata data add google_cloud abc.txt
   cmdata data get google_cloud abc.txt richa
   cmdata data copy xyz.txt AWS GOOGLE
   cmdata data del google_cloud abc.txt
```

We also have MongoDB installed to save the downloaded files into the database. We are using MongoEngine as Document-Object Mapper to add records and save the file as a FileField into DB. File is stored into MongoDB using GridFS.

1.1.6.3 MongoEngine GridFS

GridFS is a specification for storing and retrieving files into MongoDB.

Instead of storing a file in a single document, GridFS divides the file into parts, or chunks, and stores each chunk as a separate document. By default, GridFS uses a default chunk size of 255 kB; that is, GridFS divides a file into chunks of 255 kB with the exception of the last chunk. The last chunk is only as large as necessary. Similarly, files that are no larger than the chunk size only have a final chunk, using only as much space as needed plus some additional metadata.

GridFS uses two collections to store files. One collection stores the file chunks, and the other stores file metadata. The section GridFS Collections describes each collection in detail.

When you query GridFS for a file, the driver will reassemble the chunks as needed. You can perform range queries on files stored through GridFS. You can also access information from arbitrary sections of files, such as to “skip” to the middle of a video or audio file.

GridFS is useful not only for storing files that exceed 16 MB but also for storing any files for which you want access without having to load the entire file into memory. See also When to Use GridFS.

This File database table structure is read from project-code/file.yml definitions and it has a primary key as the name of the file so that we can search based on this field. This also has user_uuid field to provide specific user access to the files.

```
class File(Document):
    name = StringField(primary_key=True)
    endpoint = StringField()
    checksum = StringField()
    size = StringField()
    timestamp = DateTimeField(default=datetime.datetime.now)
    last_modified = DateTimeField(default=datetime.datetime.now)
    user_uuid = StringField()
    file_content = FileField()
```

Similarly we have a User table to store Users with their roles and group.

This project also has RESTFUL APIs to perform all the above operations and their Swagger UI looks like below. For File APIs, please refer to screenshot below for Swagger UI for File APIs (refer to FileSwaggerAPI.png).

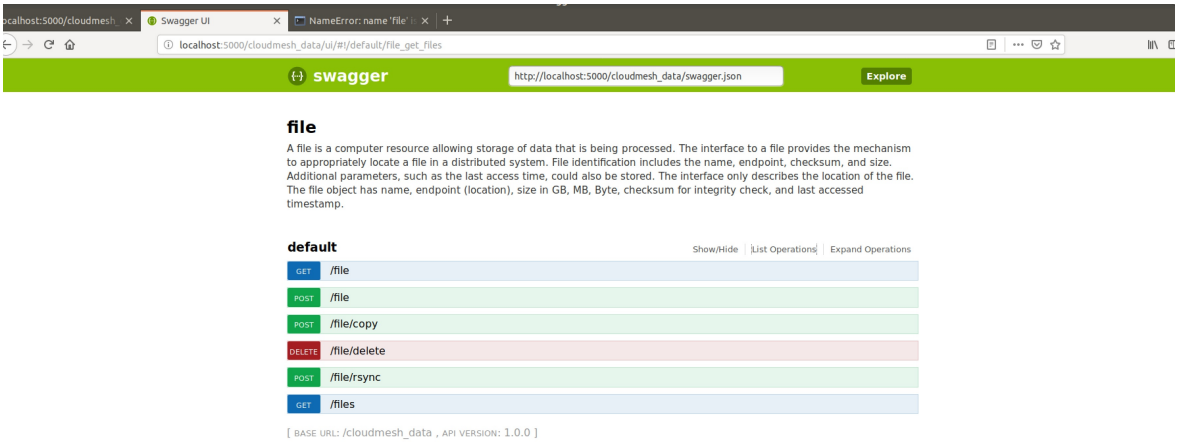

Figure 2: FileSwaggerAPI

API Path	Type	Description	Input Parameters
/cloudmesh/files?service={provider}	GET	Returns all files from a specific provider	Query Param: Provider name
/cloudmesh/file?service={provider}&filename={filename}&user_uuid={user}	GET	Returns a file for a specifc provider	Query Param: Provider Name, filename, user_uuid
/cloudmesh/file?service={provider}&filename={filename}	POST	Upload a file to a provider	Query Param: Provider name,filename
/cloudmesh/file/copy?service={provider}&filename={filename}&dest={destination}	POST	Copy a file to a provider	Query Param: Filename, Provider, destination
/cloudmesh/file/rsync?service={provider}&filename=	POST	Rsync a file to another	Query Param: Filename, Provider,

{filename}&dest={destination}		directory	destination
/cloudmesh/file/delete?service={provider}&filename={filename}	DELETE	Delete a file from a directory	Query Param: Filename, Provider

For User APIs, please refer to screenshot below for Swagger UI for User APIs (refer to UserSwaggerAPI.png).

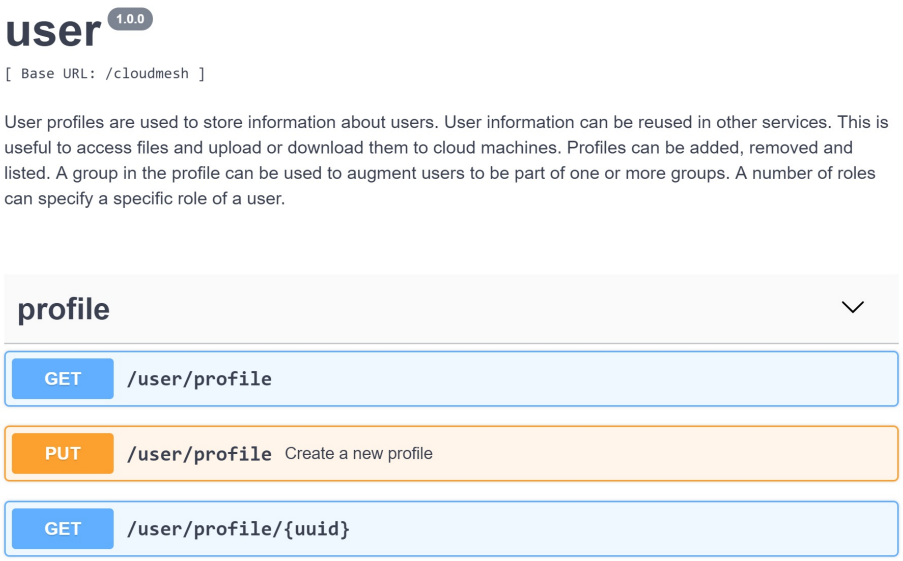

Figure 3: UserSwaggerAPI

API Path	Type	Description	Input Parameters
/cloudmesh/user/profile	GET	Returns all profiles	NONE
/cloudmesh/user/profile	PUT	Create a new profile	Body: Profile Object
/cloudmesh/user/profile/{uuid}	GET	Returns the profile of a user while looking it up with the UUID	Path Param: UUID

1.1.7 Dataset

REST API Output for getting files list (refer to FileListAPIOutput.PNG) API URL - http://localhost:5000/cloudmesh_data/files?service=google_cloud

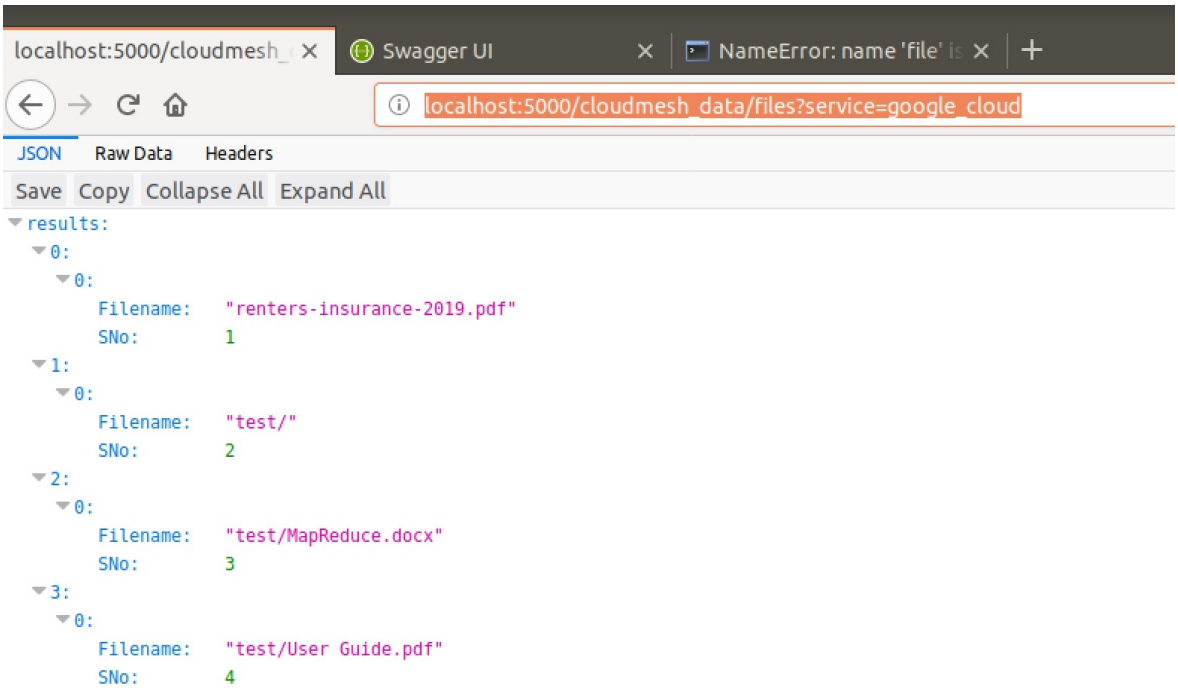

Figure 4: FileListAPIOutput

Database records for File table. This shows that file_content is stored in another table as per GridFS described above in fs.chunks and fs.files:


```
{'_id': 'MapReduce.docx', 'endpoint': 'AWS', 'checksum': '2c716d77f0916df41147f16c05c91e10', 'size': '149.5 KB', 'timesta
{'_id': 'aws_lambda.png', 'endpoint': 'AWS', 'checksum': 'a73e3d8449ab6e7366a7b1e7f24dab35', 'size': '99.6 KB', 'timestam
```


Database records for User table.


```
{'_id': '11111', 'username': 'richa.rastogi', 'group': 'test', 'role': 'test', 'resource': 'test', 'context': 'test', 'de
```

1.1.8 Conclusion

The main objective of this project was to gain knowledge and understanding of different cloud providers and create RESTFUL APIs using OpenAPI architecture using either flask or Eve and then to use MongoDB database to manage this data coming in from cloud providers. There were several other technologies been used to bring this whole project together.

This project can be enhanced even further by including many other cloud providers like Openstack, Azure etc. and all these clouds can perform operations within themselves which can reduce costs for certain people since they will not be exceedingly dependent on only just one provider. Since this project has its own access and database system enabled so this can be used as a file manager in itself.

1.1.9 Acknowledgement

I am very thankful to Professor Gregor von Laszewski for helping me throughout this project development as I am new to all the technologies used in this project. I also took help from nist and cm projects to understand the OpenAPI development using flask.

1.1.10 References

- [1] https://github.com/cloudmesh-community/nist/tree/master/services
- [2] https://github.com/cloudmesh-community/cm/tree/master/cm4

1.2 Explore OpenFaaS Development and Deployment Aspects

Murali Cheruvu, Anand Sriramulu
mcheruvu@iu.edu, asriram@iu.edu
Indiana University
hid: fa18-516-11 fa18-516-23
github:
code:

Keywords: OpenFaaS, Serverless, Micro-Services, Function-as-a-service (FaaS)

1.2.1 Abstract

Explore creating micro-services using OpenFaaS that makes serverless functions simple for containers like Docker and Kubernetes. Integrate OpenFaaS serverless functions into public cloud offerings such as, AWS and Azure, to make them even better.

1.2.2 Introduction

Goals of this project is to learn how to create cloud-native micro-services using server-less concepts for better scalability and cheaper to maintain. Function-as-a-service (FaaS) methodology allows to decouple each functionality from the rest of the application, for better support, isolated deployment and scalability at each function level. We will use OpenFaaS, an open source alternative framework to develop and deploy micro-services as FaaS in cloud-agnostic way. OpenFaaS has an opiniated way of developing FaaS and deploying them to achieve the required scalability without much depending on the infrastructure of the public cloud offerings, using container concepts wrapped around our FaaS functions. In the context of developing loosely couple components, we can interchangeably use micro-service for function-as-a-service (FaaS).

Let us introduce some of the key concepts that are related to the function-as-a-service.

1.2.2.1 Micro-Services

Micro-Services is a new paradigm in the software architecture to break down complex monolithic applications into more manageable and decoupled components that can be created and supported in silos. Loosely coupled components offer scalability and also we can use programming-of-choice based on the nature and complexity of the component, anywhere from advanced Object-Oriented Programming (OOP) languages like Java and C#.net, to modern functional-programming (FP) languages like Scala or Python. Deploying micro-services can be as flexible as deploying each individual functionality as a

separate micro-service to grouping of related micro-services into one deployment package [3]. Micro-Services, targeting for web-based interfaces, can be implemented using simple REST-based API methodology.

1.2.2.2 API Gateways

Micro-Services offer flexibility and scalability but they bring complexity into the deployment and support. Too many micro-services can create confusion in discovering them and also, client applications may have to make multiple micro-service calls, hence create more network traffic even to populate a single webpage. As an example, Amazon uses lots of micro-services to display a typical product search result webpage. To reduce the network round-trips, it is advised, to create a gateway - API gateway, so that clients make unified calls to the gateway and all the related micro-services to fulfill a request will be made within the server and the results of these micro-services are bundled into one result, hence reduce the number of calls to make [4]. Cloud Offerings such as Microsoft Azure and AWS offer API Gateways with automatic API discovery and quota-based usage along with lots of DevOp tools for continuous monitoring the scalability, performance and health-check of the micro-services. OpenFaaS has built-in API Gateway and integrates well with continuous monitoring tools such as **Grafana** and **Prometheus** [5].

1.2.2.3 Serverless

Serverless is the new methodology that cloud providers such as AWS, Azure or Google Cloud brought to simplify the deployment, execution and support of micro-services. Cloud providers take care the responsibility of the deployment, execution of the code and supporting - monitoring, tracking and notifying errors, auto-scaling (up or down) and performance metrics. Application providers are responsible to develop in a cloud-native fashion and leave the rest of the responsibilities to the cloud providers. AWS Lambda, Azure Functions, Google Functions are serverless offerings that are available today with high scalability turned on and cheaper to host such serverless functions [6].

1.2.2.4 OpenFaaS

OpenFaaS (Functions as a Service) is a framework for creating micro-services that can be hosted in containers like Docker or Kubernetes and make these services ready to be served in a serverless fashion [7]. OpenFaaS can easily deployed into all the popular public cloud providers such as AWS, Azure and Google Cloud.

1.2.3 Requirements

This project has two goals - (1) How to deploy machine learning algorithm to production and make it to work as serverless function to get best scalability and (2) Explore OpenFaaS and related tools to develop, test and deploy onto public cloud providers such as Azure, AWS and Google Cloud.

High level requirements include: setting up OpenFaaS locally within the development environment in Windows and also create deployable aspects: containerized OpenFaaS to be able to deploy to public cloud offerings including AWS, Azure and Google Cloud. Create python based project exploring high level concepts of serverless/micro-services for web. Explore container features in the process.

- Development Environment: Windows 10 Enterprise
- Install Docker Community Edition and Docker Swarm with single-node cluster
- Setup developer account with Docker Hub for publishing Docker Images on the internet
- Install OpenFaaS CLI - command line interface for OpenFaaS
- Deploy OpenFaaS into the development environment to make it ready to use
- Install Python and related libraries to make it ready for the actual project for machine learning algorithms to run and also write OpenFaaS functions in python.
- Write Python based code for object (image) detection using Convoluted Deep Neural Network (CNN) algorithms
- Train the model using around 20,000 images of dogs and cats provided by one of Kaggle competition [9].
- create OpenFaaS function to predict the uploaded image - whether it is a

dog or cat
- create another OpenFaaS function to detect the uploaded image as what animal using pre-trained model by Keras library.

1.2.4 Design

Create image object detection and classification model using Convolutional Neural Network by exploring very small number of training examples, from the dataset of 20,000 images of dogs and cats, using Python libraries such as Keras and Tensorflow. We will train the small neural network as a baseline and apply fine-tuning of an existing pre-trained network provided by popular VGGNet image network [10]. We will provide some background on the concepts of neural network to understand the project domain.

1.2.4.1 Neural Network

Neural Network is modeled after the human brain, specifically the way it solves complex problems. Perceptron, the first generation neural network, created a simple mathematical model or a function, mimicking neuron - the basic unit of the brain, by taking several binary inputs and produced single binary output. Sigmoid Neuron improved learning by giving some weightage to the input based on importance of the corresponding input to the output so that tiny changes in the output due to the minor adjustments in the input weights (or biases) can be measured effectively. Neural Network is, a directed graph, organized by layers and layers are created by number of interconnected neurons (or nodes). Every neuron in a layer is connected with all the neurons from the previous layer; there will be no interaction of neurons within a layer. As shown in +Figure 5, a typical Neural Network contains three layers: input (left), hidden (middle) and output (right) [11]. The middle layer is called hidden only because the neurons of this layer are neither the input nor the output. However, the actual processing happens in the hidden layer as the data passes through layer by layer, each neuron acts as an activation function to process the input. The performance of a Neural Network is measured using cost or error function and the dependent input weight variables. Forward-propagation and backpropagation are two techniques, neural network uses repeatedly until all the input variables are adjusted or calibrated to predict accurate output. During, forward-propagation, information moves in forward direction and passes through all the layers by applying certain

weights to the input parameters. Back-propagation method minimizes the error in the weights by applying an algorithm called gradient descent at each iteration step.

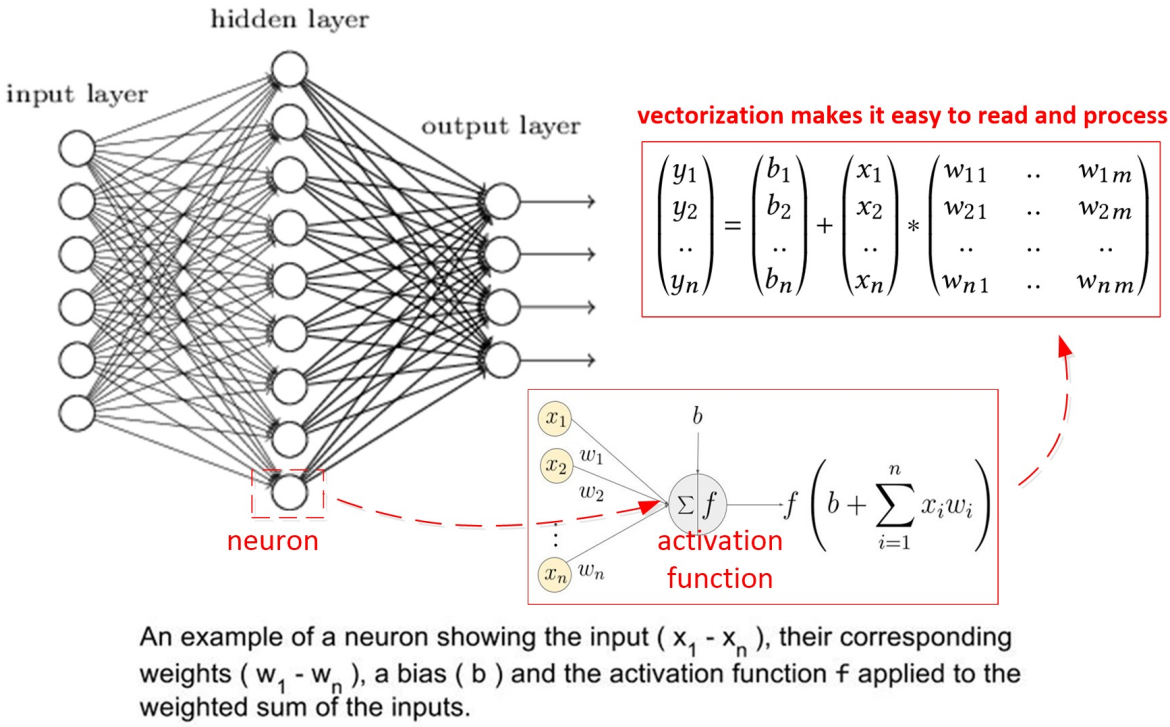

Figure 5: Neural Network [11]

1.2.4.2 Deep Neural Network

Deep Learning is an advanced neural network, with multiple hidden layers (thousands or even more deep), that can work well with supervised (labeled) and unsupervised (unlabeled) datasets. Applications, such as speech, image and behavior patterns, having complex relationships in large-set of attributes, are best suited for Deep Learning Neural Networks. Deep Learning vectorizes the input and converts it into output vector space by decomposing complex geometric and polynomial equations into a series of simple transformations. These transformations go through neuron activation functions at each layer parameterized by input weights. For it to be effective, the cost function of the neural network must guarantee two mathematical properties: continuity and differentiability.

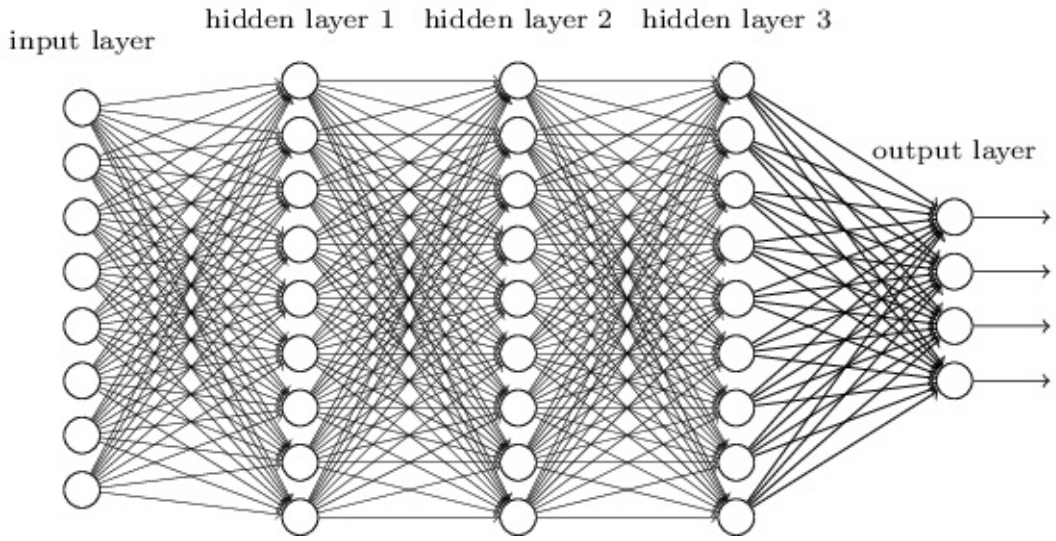

Figure 6: Deep Neural Network [11]

1.2.4.3 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN), also called multilayer perceptron (MLP), is a deep feedforward network, consists of (1) convolutional layers - to identify the features using weights and biases, followed by (2) fully connected layers - where each neuron is connected from all the neurons of previous layers - to provide nonlinearity, sub-sampling or max-pooling, performance and control data overfitting [12]. CNN is used in image and voice recognition applications by effectively using multiples copies of same neuron and reusing group of neurons in several places to make them modular. CNNs are constrained by fixed-size vectorized inputs and outputs.

Convolution Neural Network has two key components: (a) feature extraction - in this component, the network performs a series of convolutions (mathematical operation) and pooling operations to create the feature-maps, the list of features from the images. (b) classification - fully connected layers will serve as a classifier on top of these extracted features. They will assign probability for the object on the image being what the algorithm predicts it is.

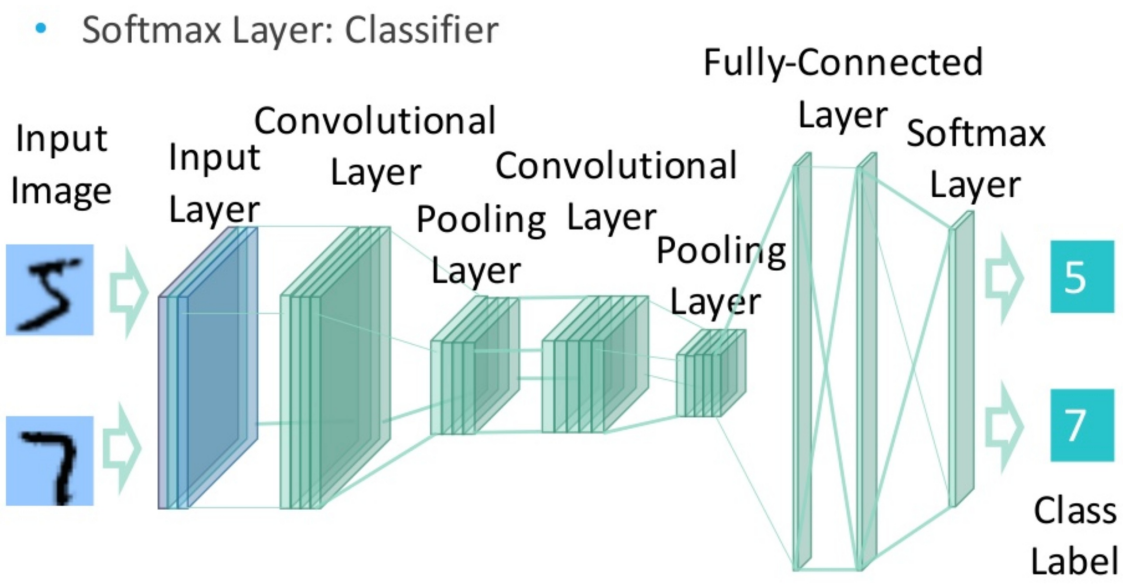

Figure 7: Convolutional Neural Network (CNN) [13]

1.2.5 Architecture

Computer vision is an interdisciplinary field that deals with how computers can be made for gaining high-level understanding from digital images or videos. Images are treated as a matrix of pixel values. By applying convolutional, mathematical operation, features such as edges, brightness or blur can be extracted as feature maps from the images. This process goes through series of convolutions followed by pooling to reduce the size the of the images for further processing. In the end, various features of the image are flattened into vector and create deep neural network to classify the image, in our case, whether it is a dog or cat. VGGNet is very popular image network that reduced the errors in the image classification and improved the processing performance from the predecessor image network models. We will create our based neural network model using a small sample dataset of 2000 images from the given 20,000 images and apply this on top of the pre-trained model that is optimized based on VGGNet algorithm. So that our effort is incremental and minimal.

1.2.6 Dataset

About 20,000 images of dogs and cats are provided part of the Kaggle competition. Dataset can be downloaded from HERE.

1.2.7 Implementation

1.2.7.1 Install Docker Swarm (Single-Node Cluster), Docker and OpenFaaS

- **Prerequisites:** Windows 10 Professional or Enterprise Edition, open the command prompt in Administrator moode
- **Step 1:** Install Docker Community Edition
- **Step 2:** Install Git Bash for pulling the latest OpenFaaS artifacts and all the other software from GitHub
- **Step 3:** Run **docker swarm init** to set up the single-node docker swarm cluster
- **Step 4:** Create an account with Docker Hub, if the created docker images to be shared with others through internet
- **Step 5:** Run **docker login** to make sure docker is linked to your account
- **Step 6:** Download latest **faas-cli.exe** from HERE
- **Step 7:** Copy the **faas-cli.exe** to *C:folder to make it available for the command prompt. Or you will need to add the path of the faas-cli.exe into the system environment variables
- **Step 8:** Test the faas-cli using the command - **faas-cli version**
- **Step 9:** Clone the OpenFaaS artifacts from GitHub using : git clone https://github.com/openfaas/faas
- **Step 10:** Go into the **faas** folder that to checkout the git master repository - cd faas and git checkout master
- **Step 11:** Run **deploy_stack.sh –no-auth** to deploy the latest OpenFaaS into our environment
- **Step 12:** Run **docker service ls** to verify whether openfaas has been deployed to our environment

1.2.7.2 Trouble Shooting

If there are any issues with the docker and/or OpenFaaS functions, we can reset the environment using the following commands

- **restart docker** - to restart the docker
- **docker stack rm func** - to remove all the functions
- **docker swarm leave –force** - to shutdown the docker cluster
- **docker swarm init** - to initialize docker swarm cluster

- **{open_faas_github_folder}/deploy_stack.sh** - to pull the latest code from openfaas GitHub

1.2.7.3 Build and deploy a serverless OpenFaaS function

1.2.7.3.1 Get FaaS-CLI

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

1.2.7.3.2 Build, deploy and push to Docker Hub

```
$ cd fa18-516-11/project-code
$ docker build -t faas-ressnet .
$ faas-cli deploy --image faas-ressnet --name faas-ressnet
$ docker tag faas-ressnet $anandid/faas-ressnet
$ docker push $anandid/faas-ressnet
```

1.2.7.3.3 Testing OpenFaaS function

1.2.7.3.3.1 Test Request : 1

faas - OpenFaas - tiger

```
Input:
curl -X POST -H  \
  --data-binary @data/tiger.png \
  "http://127.0.0.1:8080/function/faas-resnet"

Output:

Predicted: [('n02129604', 'tiger', 0.92411584), ('n02123159', 'tiger_cat', 0.04635064), ('n02391049', 'zebra', 0.01765487
```

1.2.7.3.3.2 Test Request : 2

faas - OpenFaas - cow

```
Input:
curl -X POST -H  \
  --data-binary @data/cow.jpg \
  "http://127.0.0.1:8080/function/faas-resnet"

Output:

Predicted: [('n02403003', 'ox', 0.55445725), ('n03868242', 'oxcart', 0.36393312), ('n02109047', 'Great_Dane', 0.035532992
```

1.2.7.4 Deploying to AWS

1.2.7.4.1 Setup AWS Instance

1. Purchased Spot Instance for Ubuntu 16.04, and with instance type we'll use m4.xlarge
2. Enabled Security group allowing ports 22, 31112, and 6443 for ingress
3. Created a key-pair file, so that we can SSH in to the instance
4. Test the Instance

```
ssh -i "faas.pem" ubuntu@ec2-18-191-176-209.us-east-2.compute.amazonaws.com
```

1.2.7.4.2 Setting up Kubernetes on AWS

1. Prep the machine by installing some necessary components. Run the following commands to enter superuser mode, install some necessary components from this gist, then exit back into the ubuntu user.

```
$ sudo su
$ curl -sSL https://gist.githubusercontent.com/ericstoekl/1d4372e9398d9cec7ec028629b2c36e2/raw/6f03cf3481c10e3bcf01a495a2
exit
```

2. Deploy Kubernetes

```
$ sudo kubeadm init --kubernetes-version stable-1.8
```

3. Networking layer for the cluster, to allow inter-pod communication

```
$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"
```

4. To Allow container placement on the master node and confirm the cluster is running

```
$ kubectl taint nodes --all node-role.kubernetes.io/master-

$ kubectl get all -n kube-system
```

1.2.7.4.3 Deploying OpenFaas on Kuberetes using faas-netes

1. Clone the node, Deploy the Whole Stack and deploy OpenFaas

```
$ git clone https://github.com/openfaas/faas-netes
$ kubectl apply -f https://raw.githubusercontent.com/openfaas/faas-netes/master/namespaces.yml
$ cd faas-netes && \
kubectl apply -f ./yaml
```

2. Install the CLI, deploy samples

```
$ curl -sL https://cli.openfaas.com | sudo sh
$ git clone https://github.com/openfaas/faas-cli
```

3. Pull docker image (OpenFaas functions)

```
docker pull anandid:faas-resnet
```

4. Deploy OpenFaas Functions

```
faas-cli deploy --image anandid/faas-resnet --name faas-resnet --gateway http://18.191.176.209:31112
```

5. Test OpenFaas function

```
curl http://18.191.176.209:31112/function/faas-resnet --data-binary @data/tiger.jpg
```

1.2.7.5 Deploying to Raspberry PI Clusters

PI Cluster Case [14]

1.2.7.5.1 Burn 3 Raspbery PI clusters thru cm-burn

```
$ git clone https://github.com/cloudmesh-community/cm-burn
$ cd cm-burn
$ python setup.py install
$ cmburn create --group g1 --names red[001-003] --key c:/users/anand/.ssh/id_rsa.pub --image 2018-06-27-raspbian-stretch.
```

1.2.7.5.2 Steps to setup OpenFass in Rasberry PI

1. Install Docker using the following utility script

```
$ curl -sSL https://get.docker.com | sh
```

Note: the above step can take between 2 to 5 minutes

2. Run the following command to use Docker as non-root user

```
$ sudo usermod pi -aG docker
```

3. Change default Password

```
$ sudo passwd pi
```

4. Logout and log back to take this effect for the above 2 steps

5. Setup docker swarm cluster

```
$ docker swarm init
```

Copy the output from the above command as like following, and need to be used in other PI nodes to join the cluster docker swarm join –token SWMTKN-1-25qnthaepgkcxx9qhfouh7yx0ht23od2shf44bw8tfphibsod8-b1c8o5ljjetm0vjkmamo3kk9k 192.168.1.111:2377

6. Setup OpenFaas

```
$ git clone https://github.com/alexellis/faas/
$ cd faas
```

7. Deploy sample functions

```
$ ./deploy_stack.armhf.sh
```

Other RPis will now be instructed by Docker Swarm to start pulling the Docker images from the internet and extracting them to the SD card. The work will be spread across all the RPis so that none of them are overworked.

8. After few minutes, the following command will provide the status of the functions

```
$ watch 'docker service ls'
```

9. Testing a function to see the scheduled RPI for this function

```
$ docker service ps func_markdown
```

10. The Openfaas functions can be access via

```
http://192.168.1.111:8080
```

11. Pull docker image (OpenFaas functions)

```
docker pull anandid:faas-resnet
```

12. Deploy the image to the OpenFaas

```
$ faas-cli deploy --image anandid/faas-resnet --name faasresnet --gateway http://192.168.1.111:8080
```

13. Test OpenFaas function

```
curl http://192.168.1.111:8080/function/faasresnet --data-binary @data/tiger.jpg
```

1.2.7.5.3 Install Python Libraries

Following are the steps used to install Python libraries:

- Install latest Anaconda for Windows 64-bit for Python 3
- Once Anaconda installed, use the Anaconda command prompt, to install Tensorflow and Keras
- We can add Python exe folder into windows system environment path variable, so that we can use python from the regular command prompt
- We can install Jupyter Notebook as well, given we are using Jupyter notebook to write the CNN algorithm for the image classification
- Detailed steps are provided in the installation/deployment instructions

1.2.7.6 Project Files

File	Description
readme.md	Instructions on how to deploy and use OpenFaaS Serverless Functions.
Dockerfile	Docker Image file with our OpenFaaS function and all the python dependencies that can be deployed onto typical public cloud providers: Azure, AWS, Google Cloud.
resnet_pretrained_classify	OpenFaaS serverless function folder with related files to classify the uploaded animal image using ResNet image network pre-trained model using Keras and Tensorflow libraries. The classification happens very quick, hence qualifies to be a serverless function.
index.py	Entry python file to call in the docker image.

image_classifier_dogsandcats.ipynb	Jupypter notebook with detailed analysis and exploration of Convolutional Neural Network to train the model using about 20,000 images of dogs and cats. The saved model will be used in the OpenFaaS function to test the classification of the uploaded image. Python uses Keras and Tensorflow Neural Network to perform the modeling.
classify_pre_trained_model.ipynb	Jupypter notebook to classify image of an animal using ResNet50 pre-trained model through Keras and Tensorflow.

1.2.8 Conclusion

OpenFaaS facilitates clean design, development, deployment and support of the function-as-a-service (micro-services) implementations. OpenFaaS creates co-opetitive (co-operation and competition) environment with public cloud providers. With all the needed built-in methodologies - API Gateway, FaaS, etc. and tools - security, logging, integrations with DevOp tools, etc., OpenFaaS is already a very good open source alternative for building micro-services and maturity of this framework is drastically increasing with growing usage community and adoption. OpenFaaS can be helpful to host variety of FaaS functions all the way from http-based functions to complex functions such as machine learning based predictions and classifications.

1.2.9 Team Members and Work Breakdown

- Murali Cheruvu worked on the CNN, OpenFaaS function, Docker Image and Deploying Azure (Single-Node Cluster)
- Anand Sriramulu worked on OpenFaaS function, Docker Image and Deploying to Raspberry Pi (Multi-Node Cluster)

1.2.10 Acknowledgement

The author would like to thank Dr. Gregor von Laszewski and the Teaching Assistants for their support and valuable suggestions. Author would also like to thank authors listed in the bibliography along with OpenFaas and the community of OpenFaaS for great collaboration and providing invaluable documentation and sample projects.

[1] "Cloudmesh nist services." [Online]. Available: https://github.com/cloudmesh-community/nist/tree/master/services

[2] "Cloudmesh cm project." [Online]. Available: https://github.com/cloudmesh-community/cm/tree/master/cm4

[3] M. Fowler, "Micro Services." Mar-2014 [Online]. Available: https://martinfowler.com/articles/microservices.html

[4] SnartBear, "Using an API Gateway in Your Microservices Architecture." 2018 [Online]. Available: https://smartbear.com/learn/api-design/api-gateways-in-microservices/

[5] SnartBear, "Monitoring with Prometheus and Grafana." Jan-2016 [Online]. Available: https://github.com/hashicorp/faas-nomad/wiki/Monitoring-with-Prometheus-and-Grafana

[6] A. Innovations, "What is Serverless?" [Online]. Available: https://serverless-stack.com/chapters/what-is-serverless.html

[7] OpenFaaS, "OpenFaaS: Introduction." [Online]. Available: https://docs.openfaas.com/

[8] OpenFaaS, "Become your own Functions as a Service provider using OpenFaaS.".

[9] Kaggle, "Dogs vs. Cats." [Online]. Available: https://www.kaggle.com/c/dogs-vs-cats

[10] F. Chollet, "Building powerful image classification models using very little data.".

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press,

2016 [Online]. Available: http://www.deeplearningbook.org

[12] Christopher Olah, “Conv Nets: A Modular Perspective.” Jul-2014 [Online]. Available: http://colah.github.io/posts/2014-07-Conv-Nets-Modular/

[13] M. Chang, “Applied Deep Learning 11/03 Convolutional Neural Networks.” Oct-2016 [Online]. Available: https://www.slideshare.net/ckmarkohchang/applied-deep-learning-1103-convolutional-neural-networks

[14] G. von Laszewski, “Raspbery PI 5 Node Cluster Case.” [Online]. Available: https://github.com/cloudmesh-community/case