# Function as a Service

*August 2018*

**Author**
Chirag Arora

**Supervisor**
Belmiro Daniel Rodrigues Moreira

**Group**
IT-CM-RPS

# Contents

# Preface

Writing this report wasn't a difficult task. Still, completion would not have been possible if I had not received the support of many individuals and an organisation. Therefore, I would like to extend my gratitude towards all of them.

First of all, I would like to thank my supervisor, Belmiro, for considering me as a deserving candidate for this position. I would also like to thank the IT-CM-RPS team who was always there to help me in desperate times.

To CERN, for introducing and organising this wonderful program that made this summer productive for procrastinating students like me. I should also not forget to thank the Openlab program coordinators who were always there to help us.

Finally, I would like to express my sincerest gratitude towards all the fellow Openlab students who played their part in a million little ways to make this summer as worthwhile as possible.

# Abstract

**Function as a service (FaaS)** is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage application functionalities without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app. Building an application following this model is one way of achieving a "server-less" architecture, and is typically used when building micro-services applications.

FaaS is an extremely recent development in cloud computing, first made available to the world by hook.io in October 2014, followed by AWS Lambda, Google Cloud Functions, Microsoft AzureFunctions, and Oracle Cloud Fn in 2017 which are available for public use. FaaS capabilities also exist in private platforms, as demonstrated by Uber's Schemaless triggers.

The project's goal was to understand how OpenStack Qinling can help the HEP community developing/deploying applications/workload along with how this project can be integrated with CERN cloud infrastructure.

# Introduction

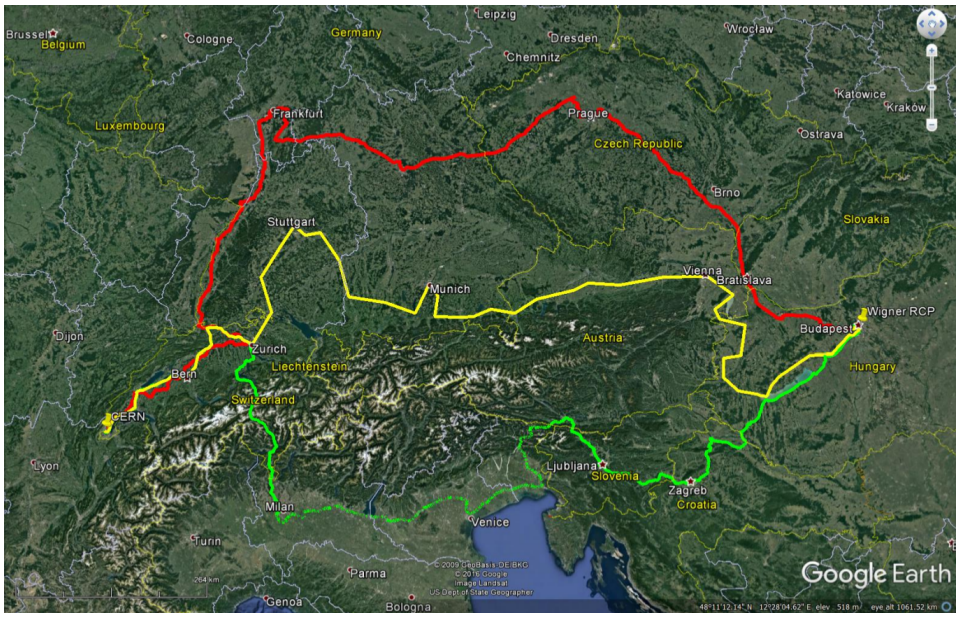
## OpenStack

OpenStack (O~S) is a free and open-source software platform for cloud computing, mostly deployed as infrastructure-as-a-service (IaaS), whereby virtual servers and other resources are made available to customers. The software platform consists of interrelated components that control diverse, multi-vendor hardware pools of processing, storage, and networking resources throughout a data centre. Users either manage it through a web-based dashboard, through command-line tools, or through RESTful web services.

OpenStack began in 2010 as a joint project of Rackspace Hosting and NASA. As of 2016, it is managed by the OpenStack Foundation, a non-profit corporate entity established in September 2012 to promote OpenStack software and its community. More than 500 companies have joined the project.

## CERN Cloud

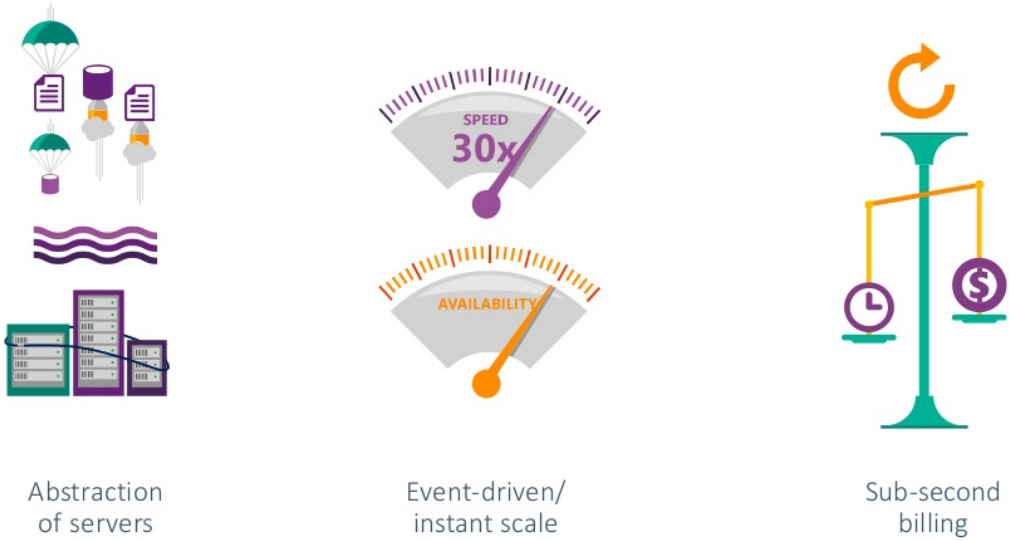
The CERN Private Cloud provides an Infrastructure-as-a-Service solution integrated with CERN's computing facilities. Using self service portals




CERN's data centres in Switzerland and Hungary are connected by three independent 100 Gb/s optical fibre cables.

or cloud interfaces, users can rapidly request virtual machines for production, test and development purposes. The machines can be of different capacities and run a variety of Windows or Linux operating systems. CERN has two data centres in Switzerland and Hungary. More than 300k cores available in these two data centres that run the analysis jobs of the LHC collision data, and plus all IT services.

# FaaS




## Serverless Computing

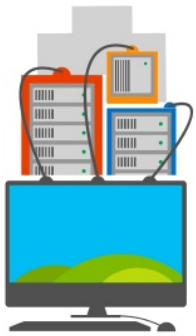
Serverless computing is a cloud computing model which aims to abstract server management and low-level infrastructure decisions away from developers. In this model, allocation of resources is managed by the cloud provider instead of the application architect, which can bring some serious benefits. In other words, serverless aims to do exactly what it sounds like — allow applications to be developed without concerns for implementing, tweaking, or scaling a server (at least, to the perspective of a user).

## Functions as a Service

FaaS is a relatively new concept that was first made available in 2014 by hook.io and is now implemented in services such as AWS Lambda, Google Cloud Functions, IBM OpenWhisk and Microsoft Azure Functions. It provides a means to achieve the serverless dream allowing developers to execute code in response to events without building out or maintaining a complex infrastructure. What this means is that you can simply upload modular chunks of functionality into the cloud that are executed independently. Instead of scaling a monolithic REST server to handle potential load, the server can now be split into a bunch of functions which can be scaled automatically and independently.

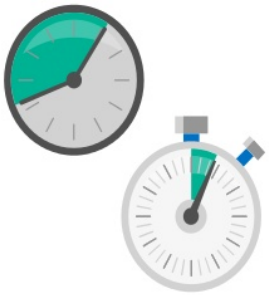
## FaaS Advantages

- Fewer developer logistics: server infrastructure management is handled by someone else.
- More time focused on writing code: higher developer velocity.



Reduced DevOps

Focus on Business Logic



Reduced Time To Market

- Inherently scalable. Rather than scaling the entire application, functions can be scaled automatically and independently with usage.
- Never pay for idle resources.
- Built in availability and fault tolerance.
- Business logic is necessarily modular and conform to minimal shippable unit sizes.

## Existing FaaS providers

- Microsoft Azure Functions
- AWS Lambda
- Google Cloud Functions
- IBM Cloud Functions

# Qinling

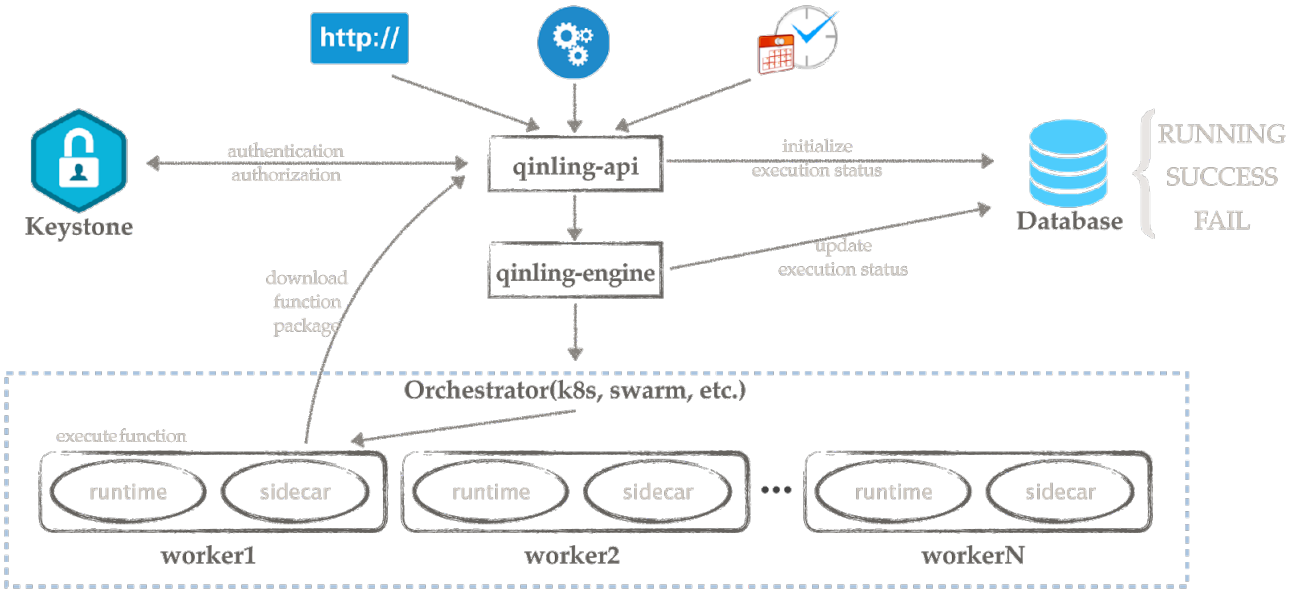
QINLING

*an OpenStack Community Project*

Qinling is an OpenStack project to provide “Function as a service”. This project aims to provide a platform to support serverless functions (like AWS Lambda). Qinling supports different container orchestration platforms (Kubernetes/Swarm, etc.) and different function package storage backends (local/Swift/S3) by nature using plugin mechanism.

With Qinling, you can run code without provisioning or managing servers. You pay only for the compute time you consume—there’s no charge when your code isn’t running. You can run code for virtually any type of application or backend service—all with zero administration. Just upload your code and Qinling takes care of everything required to run and scale your code with high availability. You can set up your code to automatically trigger from other OpenStack services or call it directly from any web or mobile app.

## Architecture



### qinling-api

A Web Server Gateway Interface that authenticates user and routes requests to qinling-engine after a preliminary handling for the request. Users can interact with qinling-api either by sending HTTP request or using openstack CLI provided by python-qinlingclient.

### qinling-engine

A standalone service whose purpose is to process operations such as runtime maintenance, function execution operations, function autoscaling, etc.

### kubernetes

Qinling uses kubernetes as the default backend orchestrator, in order to manage and maintain the underlying pods to run the functions.

### database

Qinling needs to interact with the database (MySQL) to store and retrieve resource information.

### etcd

etcd is a distributed key-value store that provides fast read/write operations for some specific internal resources in Qinling such as the mapping from functions to the function services, mapping from function to the workers, etc. In addition, etcd provides the locking mechanism in Qinling.

### rabbit messaging queue

Routes information between the qinling-engine and qinling-api.

## Quick Examples

### Creating Runtimes

CLI command to create a Runtime:

```
$ openstack runtime create <docker image> --name <runtime name>
```

Example (Create Runtime):

```
$ openstack runtime create openstackqinling/python-runtime --name
python2.7 -f json
{
  "status": "creating",
  "created_at": "2018-07-20 09:47:07.226801",
  "description": null,
  "image": "openstackqinling/python-runtime",
  "updated_at": null,
  "project_id": "53bc238760d94399b376c6d499b9a384",
  "id": "20acaf45-6fbd-4f02-8a6f-6954b5c011ee",
  "name": "python2.7"
}
```

## Creating Functions

CLI command to create a function:

```
$ openstack function create --name <function name> \
    --runtime <runtime id> --entry <entry point> \
    --file <function source code>
```

Sample function that generates random names.

```
# function.py
import requests
import random

def main(*args, **kwargs):
    word_site = "http://svnweb.freebsd.org/csrg/share/dict/words?
view=co&content-type=text/plain"
    response = requests.get(word_site)
    words = response.content.splitlines()
    upper_words = [word for word in words if word[0].isupper()]
    name_words  = [word for word in upper_words if not word.isupper()]
    name = ' '.join([name_words[random.randint(0, len(name_words))]
for i in range(2)])
    return name

if __name__ == '__main__':
    main()
```

Example (Create Function):

```
$ runtime_id=20acaf45-6fbd-4f02-8a6f-6954b5c011ee
$ openstack function create --runtime $runtime_id --name function \
  --entry function.main --file function.py -f json
{
  "count": 0,
  "code": {
    "source": "package",
    "md5sum": "5b550c24641c4ae87f3dcdf6d9201bb8"
  },
  "description": null,
  "created_at": "2018-08-09 12:17:15.606715",
  "updated_at": null,
  "cpu": 100,
  "memory_size": 33554432,
  "runtime_id": “20acaf45-6fbd-4f02-8a6f-6954b5c011ee",
  "entry": "function.main",
  "project_id": "53bc238760d94399b376c6d499b9a384",
  "id": "2970cf72-502d-47a5-96fc-3cfa76d4eb1c",
  "name": "function"
}
```

## Executing Functions

CLI command to execute a function:

```
$ openstack function execution create <function id>
```

Example (Execute Function).

```
$ function_id=2970cf72-502d-47a5-96fc-3cfa76d4eb1c
$ openstack function execution create $function_id -f json
{
  "status": "success",
  "project_id": "53bc238760d94399b376c6d499b9a384",
  "description": null,
  "updated_at": "2018-08-09 12:23:11",
  "created_at": "2018-08-09 12:23:10",
  "sync": true,
  "function_version": 0,
  "result": "{\"duration\": 0.953, \"output\": \"Okinawa
Knoxville\"}",
  "input": null,
  "function_id": "2970cf72-502d-47a5-96fc-3cfa76d4eb1c",
  "id": "4b0ed0aa-1ca3-489e-b384-3ff60ec5be08"
}
```

## Webhooks

CLI command to create a webhook

```
$ openstack webhook create <function id>
```

Example (Creating Webhook)

```
$ function_id=2970cf72-502d-47a5-96fc-3cfa76d4eb1c
$ openstack webhook create $function_id -f json
{
  "function_id": "2970cf72-502d-47a5-96fc-3cfa76d4eb1c",
  "description": null,
  "created_at": "2018-08-09 12:33:31.037787",
  "updated_at": null,
  "function_version": 0,
  "webhook_url": “http://cci-qinling-001.cern.ch:7070/v1/webhooks/
92fbe186-14ec-4bc3-9d23-68e8e0d96e2b/invoke”,
  "project_id": "53bc238760d94399b376c6d499b9a384",
  "id": "92fbe186-14ec-4bc3-9d23-68e8e0d96e2b"
}
```

An instance of function execution can be made by sending an HTTP POST request to the url. The input can be specified in JSON format inside the request body.

# Contributions to Upstream

## Bugs Discovered

### Runtime incompatibility with systemd cgroupdriver

| | | |
|---|---|---|
| heychirag | Hi, my name is Chirag and I am testing the Qinling project for CERN so that it can integrated into their cloud. | [08:44] |
| heychirag | So I am trying to connect Qinling to an external Kubernetes cluster. | [08:47] |
| heychirag | But my function execution on python runtime is returning a HTTP 500 on line 86-102: cglimit.py | [08:49] |
| heychirag | I tried skipping cglimit.py and the function execution worked flawlessly. | [08:50] |
| heychirag | Can someone explain why cglimit.py is necessary and what would happen if I don't use it? | [08:50] |
| huntxu | heychirag: cglimit is used to limit the resource used when running a function(i.e. during an execution) | [08:53] |
| huntxu | heychirag: it prevents *bad* functions exhausting the resources of your nodes | [08:53] |
| huntxu | heychirag: if you are not acting as a service provider, which means that all your functions are from trusted sources, just pay attention to your execution stastics if you don't use the limiting | [09:00] |
| huntxu | *statistics | [09:01] |
| huntxu | heychirag: back to your issue, could you please check whether dockerd and kubelet are using the 'cgroupfs' cgroupdriver instead of 'systemd', unfortunately, the current cglimit implementation requires the former being used | [09:06] |
| heychirag | huntxu: Thanks! the cgroupdriver is systemd. This resolves my query. | [09:20] |

I discovered that the provided runtime was incompatible with a Magnum provisioned Kubernetes cluster on CERN's cloud. This was because the runtime tried to limit the resources without checking for the cgroupdriver that was being used. As you can decipher from the above chat, the runtime is compatible with cgroupfs, whereas we are using systemd as the cgroupdriver at CERN. Instead of failing the execution, it makes much more sense to just print a warning and continue with the execution without limiting the resources.

## Patches Submitted

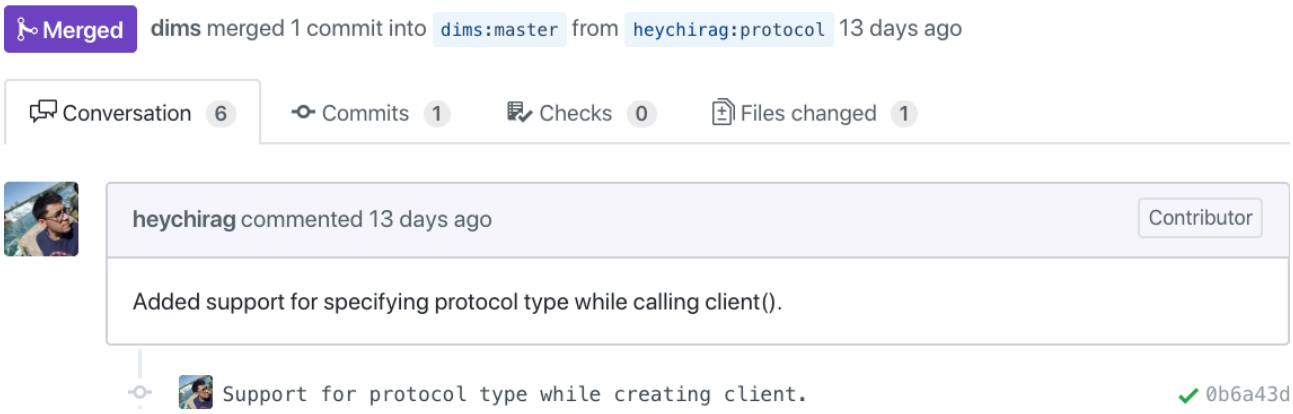
openstack. All Projects Documentation
Open Merged Abandoned
owner:heychirag AND (is:open is:mergeable OR Changes Search Sign In

Search for owner:heychirag AND (is:open is:mergeable OR is:merged)

| Subject | Status | Owner | Project | Branch | Updated | Size | CR | V | W |
|---|---|---|---|---|---|---|---|---|---|
| Added support for secure connection to etcd | | Chirag Arora | openstack/qinling | master (etcd) | 12:53 PM | | | +1 | |
| Timeout support for api and engine | | Chirag Arora | openstack/qinling | master (qintime) | Aug 10 | | | +1 | |
| Function timeout support for python2 runtime | | Chirag Arora | openstack/qinling | master (timeout) | Aug 10 | | | +1 | |
| python3 runtime | Merged | Chirag Arora | openstack/qinling | master (python3) | Aug 10 | | ✓ | ✓ | ✓ |
| Client support for timeout in function execution create | | Chirag Arora | openstack/python-qinlingclient | master (timeout) | Aug 8 | | | +1 | |
| Client support for timeout in function create | | Chirag Arora | openstack/python-qinlingclient | master (time) | Aug 8 | | -1 | +1 | |
| Custom initial number of replicas | | Chirag Arora | openstack/qinling | master (replicas) | Aug 7 | | -1 | -1 | |
| Client support for custom initial number of replicas | | Chirag Arora | openstack/python-qinlingclient | master (replicas) | Jul 26 | | | +1 | |

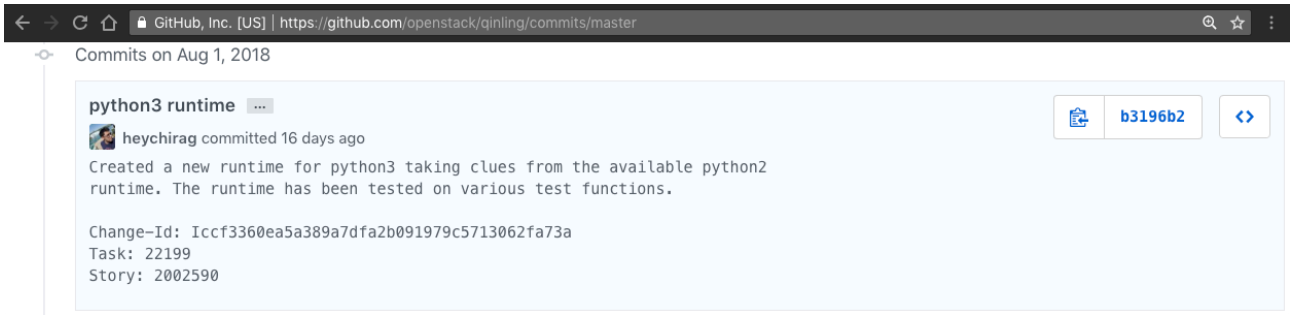
## Support for protocol type in etcd3gw <#26> (Merged)

The etcd3gw package didn't support the passing of protocol type while creating a Client instance, but to support secure connection to the etcd server, it is necessary to be able to specify the protocol type ('http'/'https'). Therefore I submitted a patch and got it merged on Github.




## python3 runtime support <2002590> (Merged)

There was a story on the storyboard to add python3 runtime support. I worked on this feature and submitted a patch which now has been accepted and merged to the qinling repository.




## Support secure connection to etcd <2003284> (Under Review)

Qinling uses etcd key-value store that provides fast read/write operations for some specific internal resources. So it is important that we provide a way to connect securely to the etcd server if the server demands so. I submitted a patch that allows the functionality to be enabled.

## Customise timeout for function <2002174> (Under Review)

Providing execution timeout is a failsafe feature that prevents over utilisation of resources and also saves from scenarios such as an infinite loop.

## Custom initial number of replica sets <2003095> (Abandoned)

It can also be sometimes useful to specify the initial number of replica sets during creation of a runtime. We later came to a conclusion that it would be better to provide runtime scaleup/scaledown feature just like the function scaleup/scaledown.

# Next Steps

If I had more time in my hands, I would have worked on the following:

- Testing Qinling scalability: One of the main things that should be tested is perhaps the breaking point of the service. We need to check what would happen if the workload is increased dramatically. This can be achieved by creating and executing thousands of function concurrently.
- Integration with other OpenStack services: Since Qinling is an OpenStack project, it would be amazing to have it integrated with other OpenStack services such as aodh and glance. This is similar to how AWS Lambda can easily be integrated with other AWS services.
- Integration with CERN Infrastructure: Since I am working on this project as part of CERN, it would have been even exciting to find interesting uses cases over here. One of them could be integrate Qinling function with monitoring services such as Graphana. Graphana can trigger the function webhook which in turn can run your custom code and perform analysis on the input.

# Conclusion

FaaS has helped in bringing an era of serverless computing. Focusing less on the infrastructure, people can be more productive with their work while also being efficient in terms of using resources in a quantised and on-demand manner.

With closed-source cloud providers excelling in the world of FaaS, there was a need of a open-source alternative. The constant effort of the upstream community has finally lead to maturity of the Qinling project. The first preview of the Qinling dashboard was released last week. And this means the project is becoming user friendly day by day. It would be even more amazing to see runtimes that can support other programming languages. It will also be helpful if seamless installation on any environment through distributable packages can be made available anytime soon.

All in all this internship was a fantastic experience for me. There are currently very few people involved in the Qinling project, but I can proudly say that this internship has helped in adding one more name to the list of contributors.