ChainFaaS: An Open Blockchain-based Serverless Platform

Release 1.0.0-alpha

Sara Ghaemi

Aug 14, 2020

CONTENTS:

The idea behind ChainFaaS is to use the untapped computational power of the current computers as a serverless platform. In ChainFaaS, regular computer users can rent out their unused computational power by connecting it to a large network of resources. On the other hand, those who need computing resources can rent from this vast pool of compute at scale. If enough personal computers were connected to ChainFaaS, the need for building new data centers would decline. However, the goal in ChainFaaS is not to replace the data centers and servers but to only offload some of their tasks by reusing the idle cycles of personal computers.

Another motivation for creating ChainFaaS is to improve developers' experience of internet-based computing services. Currently, cloud giants, such as Amazon, Google, and Microsoft, are the leading players in serverless computing. Serverless computing platforms offered by these companies are highly centralized, and such companies control every single detail of the platforms. There is no way for the developers to verify the reported billing information. On the other hand, ChainFaaS offers a low-price, transparent, reliable and easy-to-use serverless platform which is not managed by one entity. Anyone can join the network to participate in the management of the platform as well as to observe the transactions.

To sum up, ChainFaaS offers an open blockchain-based serverless platform with the following features:

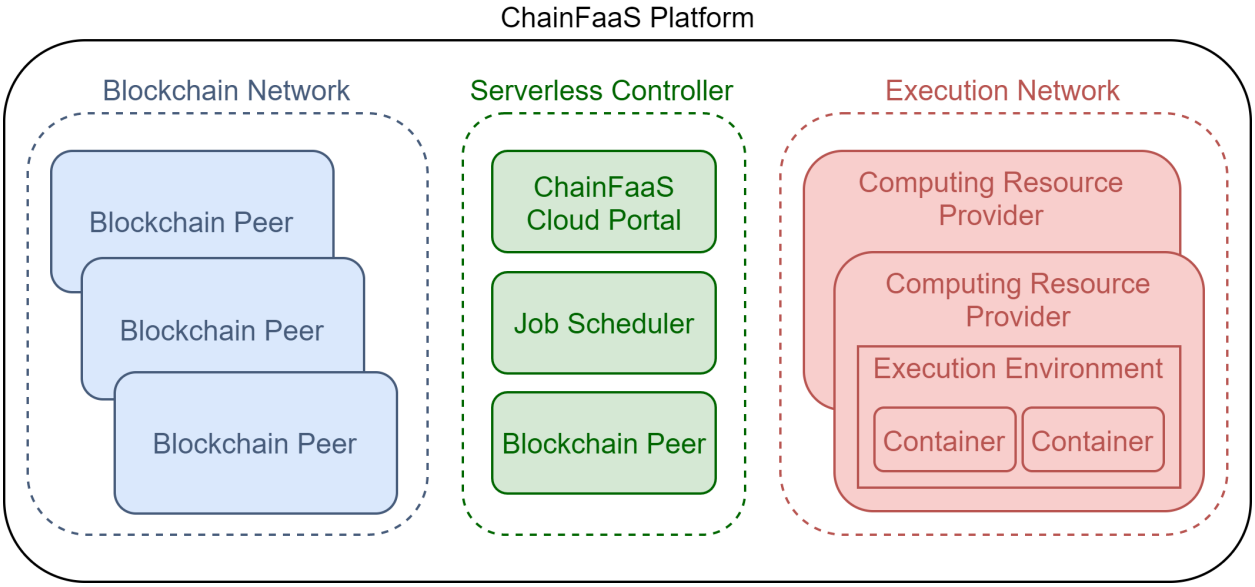
- It is public in the sense that anyone can be either a developer, provider or both.
- It is open and transparent to everyone.
- It is based on the excess computing power available on personal computers.
- It is affordable for developers, especially compared to similar centralized (in terms of management) cloud solutions.
- It is user-friendly and easy to use.

CHAPTER

ONE

INTRODUCTION

The following figure shows a high-level architecture of ChainFaaS. This platform has three main parts. The blockchain network consists of all the blockchain peers who are responsible for keeping track of the transactions on the platform. The serverless controller manages the cloud portal and the job scheduling task. The computing resource providers cooperate to shape the execution network.



1.1 Blockchain Network

The blockchain network has two main tasks: keeping records of all transactions and managing payments. From now on, we will refer to the record-keeping ledger and payment management ledger as *monitoring ledger* and *monetary ledger*, respectively. Every single transaction is stored and kept by all blockchain peers on the monitoring ledger. Any user can easily access the information of a job and see how it has evolved over time. Information such as the owner of the job, the computing resource provider who has been responsible for the job, the time it has taken to run the job, and its cost, are accessible through the monitoring ledger. The transparency feature of ChainFaaS is achieved through the monitoring ledger. The monetary ledger stores the financial account information and account balances of the users. Every time a job is successfully executed by a provider (i.e., a personal computer owner), this monetary ledger automatically transfers the cost from the developer's account to the provider's account.

The blockchain network comprises many peers that can be owned by different proprietors. Each peer keeps records of all the transactions occurring in the network. To add a new transaction to the ledger, the peers should reach a consensus on whether to accept the transaction or not. The peers receive a commission for each transaction because of

their contribution to the network. It is worth mentioning that a blockchain peer can also act as a computing resource provider in the execution network, which results in having two sources of income from ChainFaaS.

1.2 Serverless Controller

The serverless controller acts as the gateway and is responsible for providing the portal of ChainFaaS, which is publicly available. This portal is designed to help all users easily interact with the platform. Based on their enrolment in the system, users can submit new jobs, observe the status of their job, and monitor their contribution to the network. Moreover, the controller handles the job scheduling by finding a computing resource provider for each job. The job scheduling is based on a selection algorithm that may take into account many criteria to choose the provider, including the provider's availability and the computational power needed for the job. The serverless controller also acts as one of the blockchain peers and, just like any other blockchain peer, it receives a commission for the transaction.

1.3 Execution Network

The execution network consists of all computing resource providers. Anyone can easily connect their extra computational resources or their underutilized online computers to this network. This includes but is not limited to, personal computers, servers, and cloud resources. These providers get paid based on the time they spend on running the job to which they are assigned. Each job runs in an isolated execution environment on the provider's computer to ensure that it does not interfere with the provider's programs. This is also required to prevent different jobs from interfering with each other.

CHAPTER

TWO

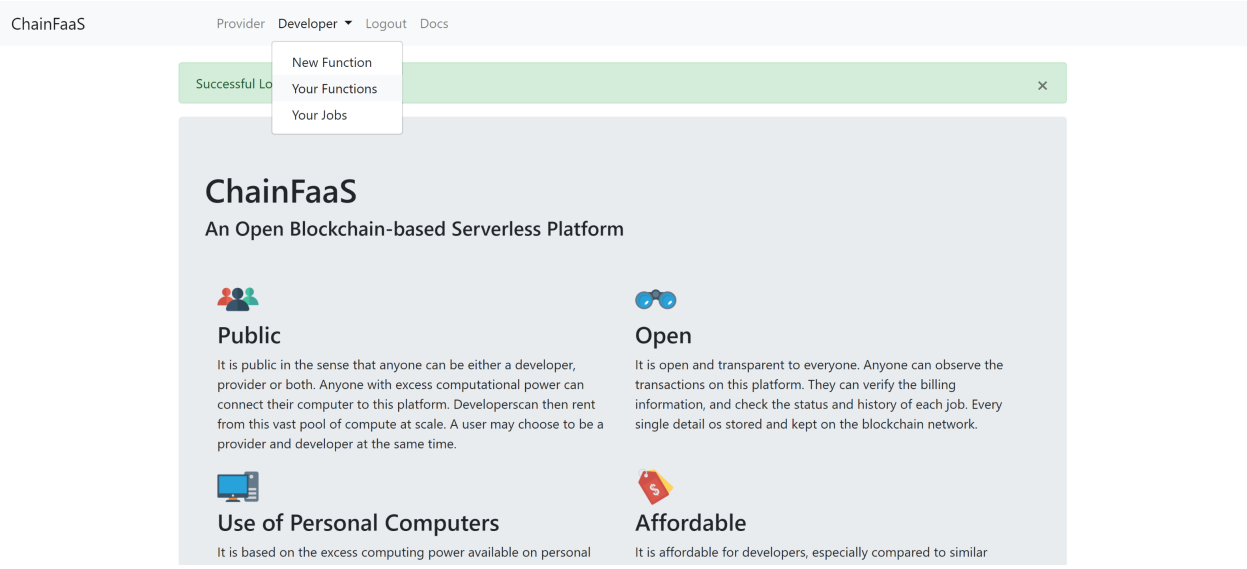
GETTING STARTED

In the current prototype of ChainFaaS, you can participate in the network as a provider or a developer. Since the platform is still running on a test network, no real currency is exchanged in the transactions. To use the test ChainFaaS network, you should first register in the chainfaas.com portal and login to your account. During the registration, you can choose to participate as a provider or a developer or both.

2.1 Developers

As a developer, you can run your functions on this serverless platform. The functions should be uploaded as Docker containers in the Docker Hub. You would need the link to your container to create a new function on ChainFaaS.

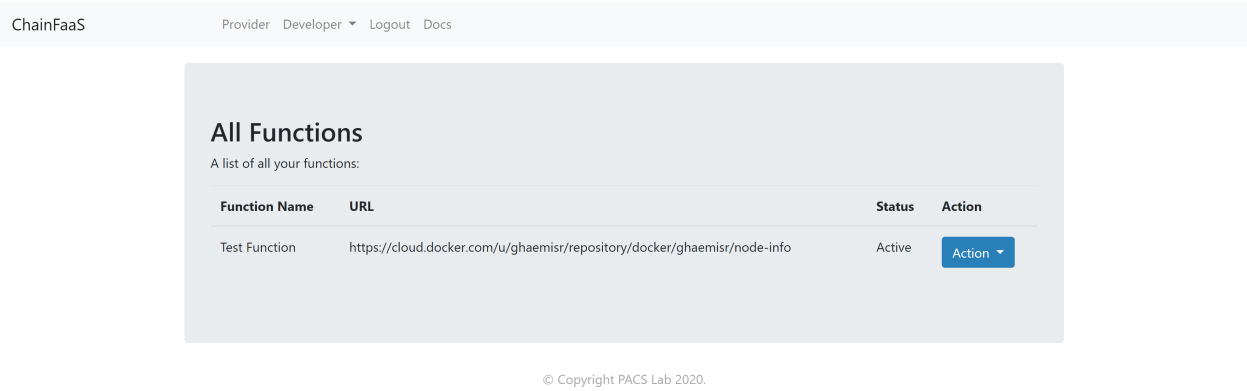
When you create a new account on ChainFaaS, an example function is automatically added to your account so that you can explore the functionalities of the platform. After a successful login, under the Developer tab, go to "Your Functions" to see the test function.



In the "Your Functions" page, you can see a list of all the functions you have added to ChainFaaS, as well as their status. For each function, you can activate the following actions:

- **Stop Function:** If the function is active, you can stop the function to disable it in the ChainFaaS network. When a function is disabled, no one can send requests to this function.
- **Send a Request:** Choosing this action will send a synchronous request to the function. The page will go to a loading mode till a response is received from ChainFaaS. This may take a few seconds to finish.

- **Send an Async Request:** Choosing this action will send an asynchronous request to the function. Instead of directly receiving the response, you can check the status of the job and the response in the Developer tab under "Your Jobs".
- **Delete Function:** This action will delete the function from your functions list.



2.2 Providers

To become a computing provider in ChainFaaS, you need to download and run the prvider's code on your computer.

The following are the prerequisites:

- Python 3.6 or greater
- Docker 18.01 or greater - prefered version: 19.03

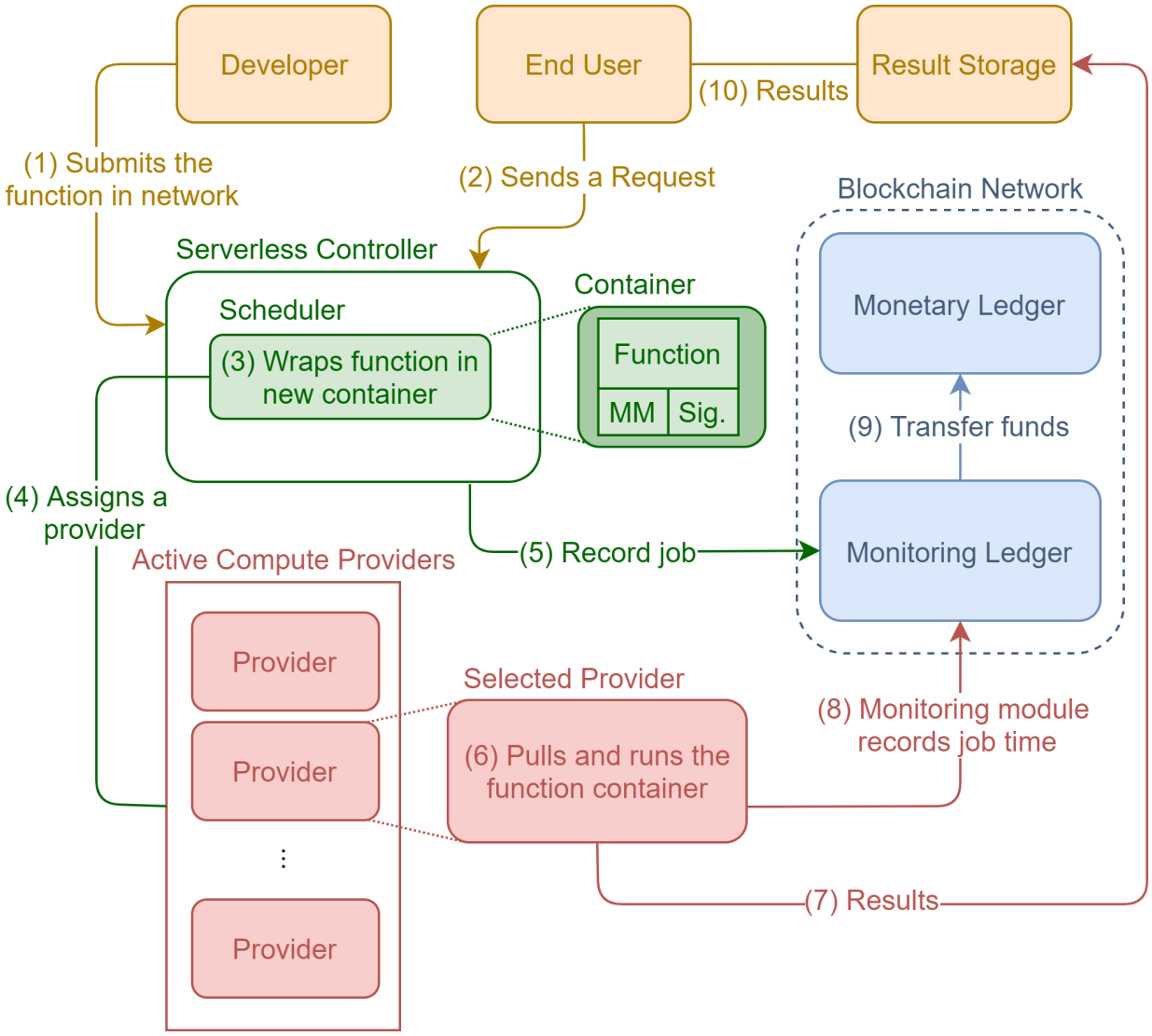
For installation instructions on Linux follow the steps provided on the ChainFaaS computer provider page on GitHub.

CHAPTER

THREE

SYSTEM DESIGN

A detailed architecture of ChainFaaS with a complete description of the process of serving a request is shown in the following figure. The developer is the owner of the function who can have clients sending requests to their function. These clients can be the developer themselves, a program owned by the developer, or anyone else. The developer can also choose to store the result of their function in a separate storage. This can be specified in the container they upload to the system. The result storage block in the following image represents this storage unit.



The scheduler in the serverless controller is responsible for scheduling the jobs, i.e., functions, and computing providers. It creates a new container that includes the function, a monitoring module (MM), and the controller's signature (Sig.). The module is designed to send back the run-time metrics of the job to the monitoring ledger. The signature is used to verify the container's creator to be the controller and not a malicious entity. When an appropriate provider is found for the job, the controller asks the monitoring ledger to keep a record of the job. The job ID stored there should be the same as the one the monitoring module later uses to record the job service time. The assigned provider is the only one who can have the job's run time recorded on the ledger. A detailed explanation of each step in the above image follows:

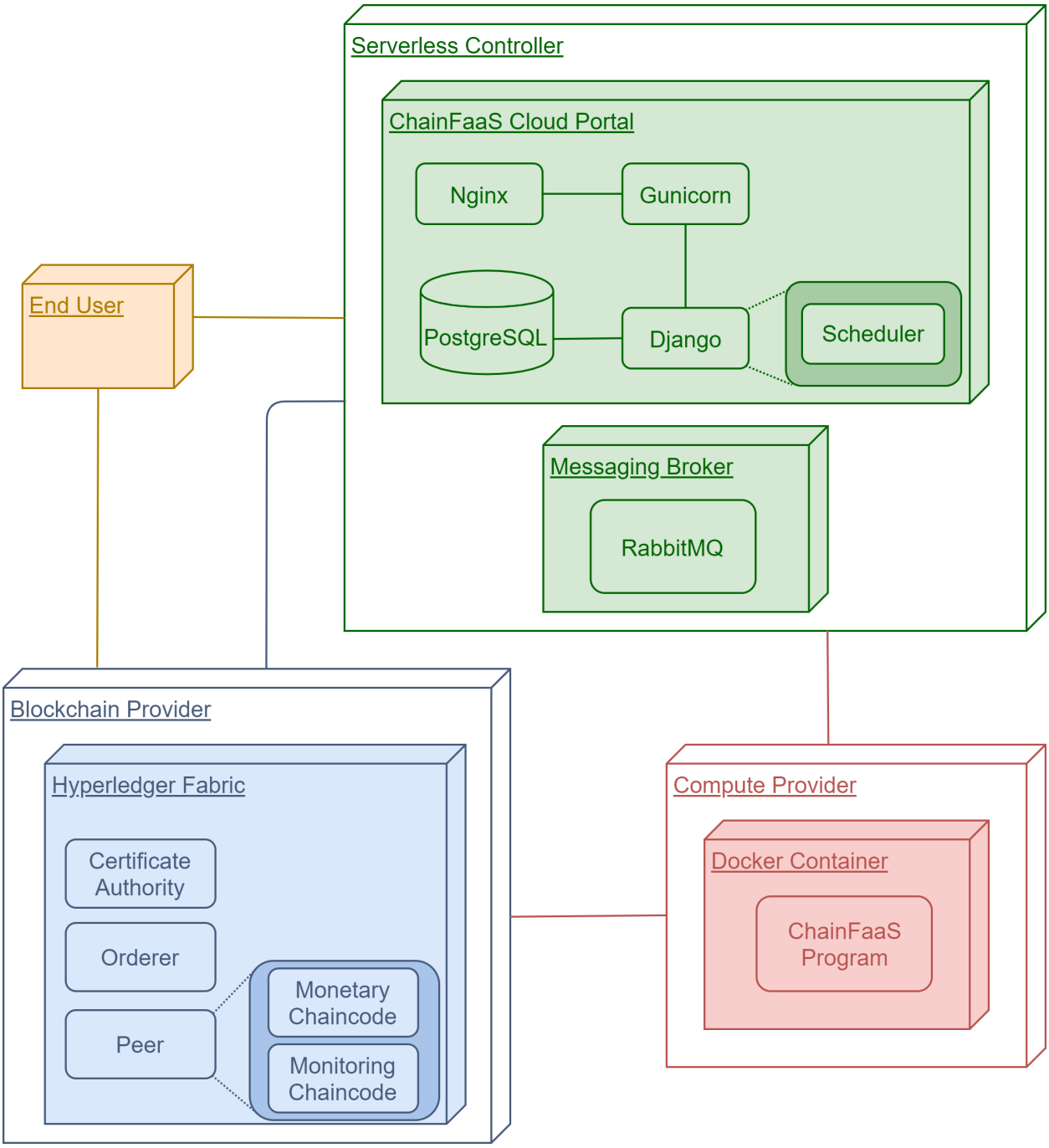
1. For a function to be available in the system, the developer first needs to submit it to ChainFaaS. To do so, the developer sets the access link and characteristics of the function in the ChainFaaS cloud portal. From the moment the function is submitted, others can send requests for that function to the serverless controller.
2. As soon as the controller receives a request for a function, a job is created, and the process starts.
3. The scheduler then adds the monitoring module (MM) to the function and wraps it in a new container with the controller's signature. The monitoring module is responsible for sending the job processing time to the monitoring chain in step 8. The signature is used in the provider to verify that the job has been sent from the controller.
4. The next step is for the scheduler to assign an appropriate provider to the job. This assignment is done based on a selection algorithm that takes into account the computational capability of the provider and its availability.
5. When a computing provider is found for the function, the controller sends a record-request to the monitoring ledger. In this request, the controller asks the blockchain network to add a new job to their records. The information added to the record includes the job ID, the developer of the job, and the provider assigned to the job.
6. The selected provider then pulls the container from the registry and runs it.
7. In the next step, the provider sends back the results to the target storage.
8. The monitoring module then uses the job ID received from the controller to ask the monitoring ledger to keep a record of the job's run time metrics.
9. In the next step, the monitoring chain charges the developer's account by transferring the amount of bill to the provider's account on the monetary ledger.
10. Finally, the result storage, which is managed and owned by the developer, sends back the results to the end-user. The developer can have the end-user manually get the result from the results storage or have the storage share the results with end-user whenever available.

CHAPTER

FOUR

IMPLEMENTATION AND DEPLOYMENT

To demonstrate the feasibility of our design, we have implemented a prototype of ChainFaaS as a proof-of-concept. In this section, we present the details of the implementation and deployment of this prototype. Details of ChainFaaS implementation, including the complete code base, can be found on GitHub. The following image shows the implementation and deployment view of ChainFaaS. The end-user represents the person who sends a request to a function. The end-users are FaaS users that can be the developers themselves, or their users who want to access the functions provided by the developer on ChainFaaS.


Serverless Controller
ChainFaaS Cloud Portal
Nginx
Gunicorn
PostgreSQL
Django
Scheduler
End User
Messaging Broker
RabbitMQ
Blockchain Provider
Hyperledger Fabric
Certificate Authority
Orderer
Peer
Monetary Chaincode
Monitoring Chaincode
Compute Provider
Docker Container
ChainFaaS Program

4.1 Serverless Controller

The controller has three main parts: cloud portal, job scheduler, and blockchain peer. The cloud portal is responsible for all interactions between the controller and the users. At the back-end of the web application, the job scheduler is also implemented. We have implemented a simple scheduler that randomly selects a provider from the available providers that can fit the request. The blockchain peer is just like other blockchain peers.

In our implementation of ChainFaaS, the serverless controller is running on an instance with 4 VCPUs, 15GB RAM, and 83GB disk with Ubuntu 18.04. The cloud portal is written in Python using the Django web framework. The job scheduler is also implemented in the backend of the cloud portal. Gunicorn is used as the application server, and Nginx is used as a reverse proxy. Gunicorn is a Python Web Server Gateway Interface (WSGI) HTTP server that communicates with the Python application. Gunicorn optimally creates as many instances as needed from the web application, distributes the requests between them, and restarts them if necessary. Nginx is used as a reverse proxy to Gunicorn to handle all incoming requests.

New users can use the ChainFaaS cloud portal to register. Each user should select what role they want to play in the platform: developer, provider, or blockchain peer. After registration is complete, the user can start interacting with the platform. As a developer, the user can submit the link to their Docker container in the Docker registry and set its characteristics.

When the scheduler matches a provider with a job, there should be a way for the serverless controller to inform the provider of the link to the function and all its information. Since providers are personal computers, they are not directly accessible by the controller. As a result, in ChainFaaS, a messaging queue has been implemented to manage the interactions between the serverless controller and each provider. RabbitMQ has been chosen as the messaging broker for this system since it is a lightweight open-source broker that can be customized for different applications. During the registration step, the providers are also registered in the messaging broker to be able to access the appropriate queue. Each provider has its own queue that only the serverless controller and the provider can access. The serverless controller puts the jobs in the provider's queue, and the provider fetches it from the queue.

4.2 Compute Provider

On providers' computers, ChainFaaS runs a program that is written in the Python programming language. As long as the program is running on the provider's computer, it receives new jobs from the serverless controller, runs them, and waits for other jobs. Jobs run in isolated environments using Docker containers to keep them from interfering with the provider's computer and accessing their files. In ChainFaaS, having isolated environments for the jobs is particularly important since the provider may not necessarily trust the source of the job. Moreover, there may be more than one job running on the provider's computer at the same time. Using sandboxing solutions, we can ensure there is no conflict between the dependencies and resources of the jobs.

A popular solution for isolating the execution environment of software is using virtual machines. In this solution, a guest operating system runs on top of a host operating system and has virtual access to the system's underlying hardware. Another solution is using containers, which also provide an isolated environment for running a software service. Unlike virtual machines that virtualize the hardware stack, containers provide the developers with a logically isolated operating system by virtualizing the computer resources at the OS-level. As a result, compared to virtual machines, containers are far more lightweight, faster to start, and use far less memory. In the current implementation of ChainFaaS, the security and privacy concerns of providers are addressed using containers as the sandboxing solution. The reason for this choice is that the providers should be able to start using the platform with minimal effort and overhead. As discussed earlier, containers are fast, lightweight, and they provide the required isolated execution environment.

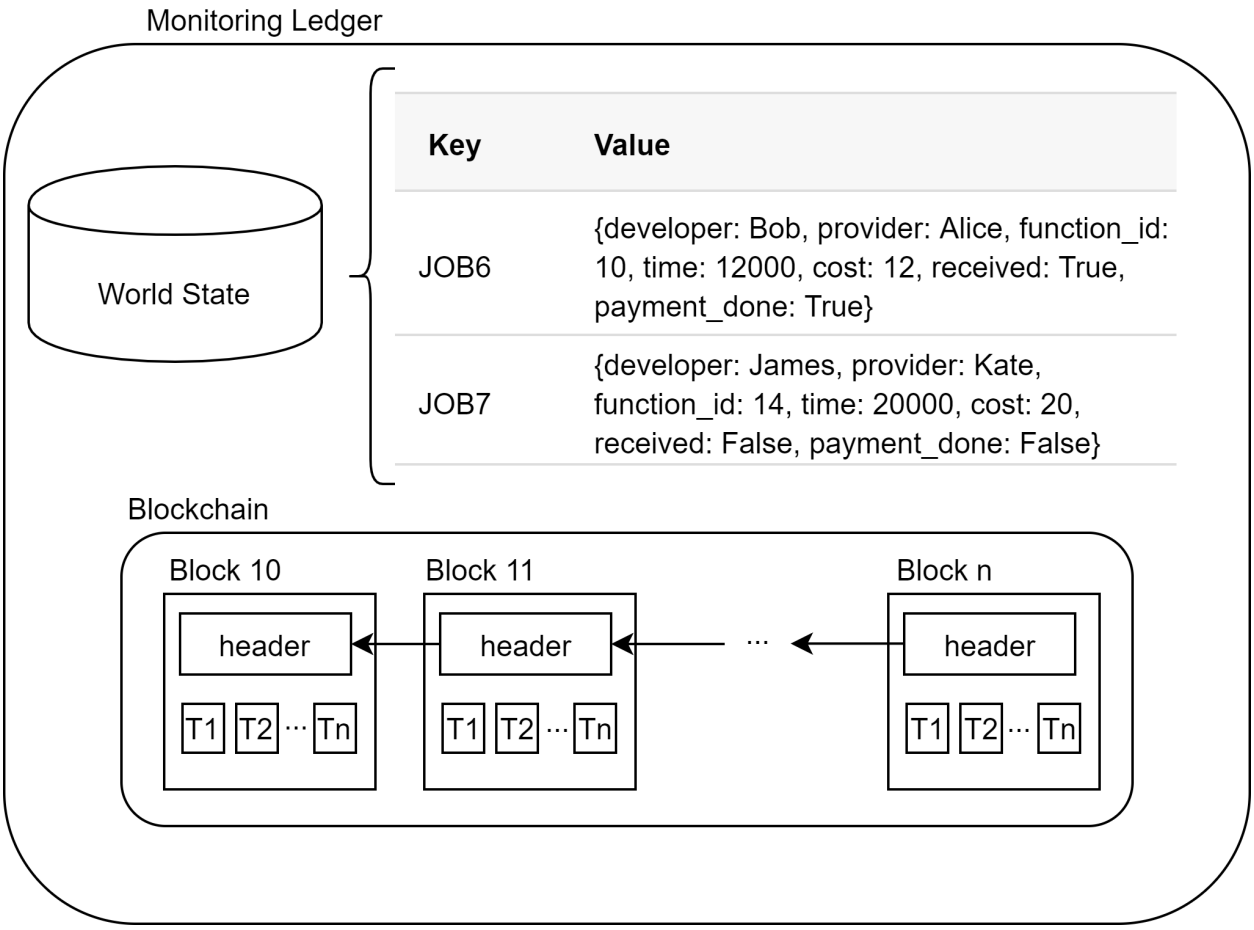
For more details on the security challenges of open platforms and a comparison on different solutions, refer to the peer-reviewd paper of ChainFaaS.

4.3 Blockchain Peer

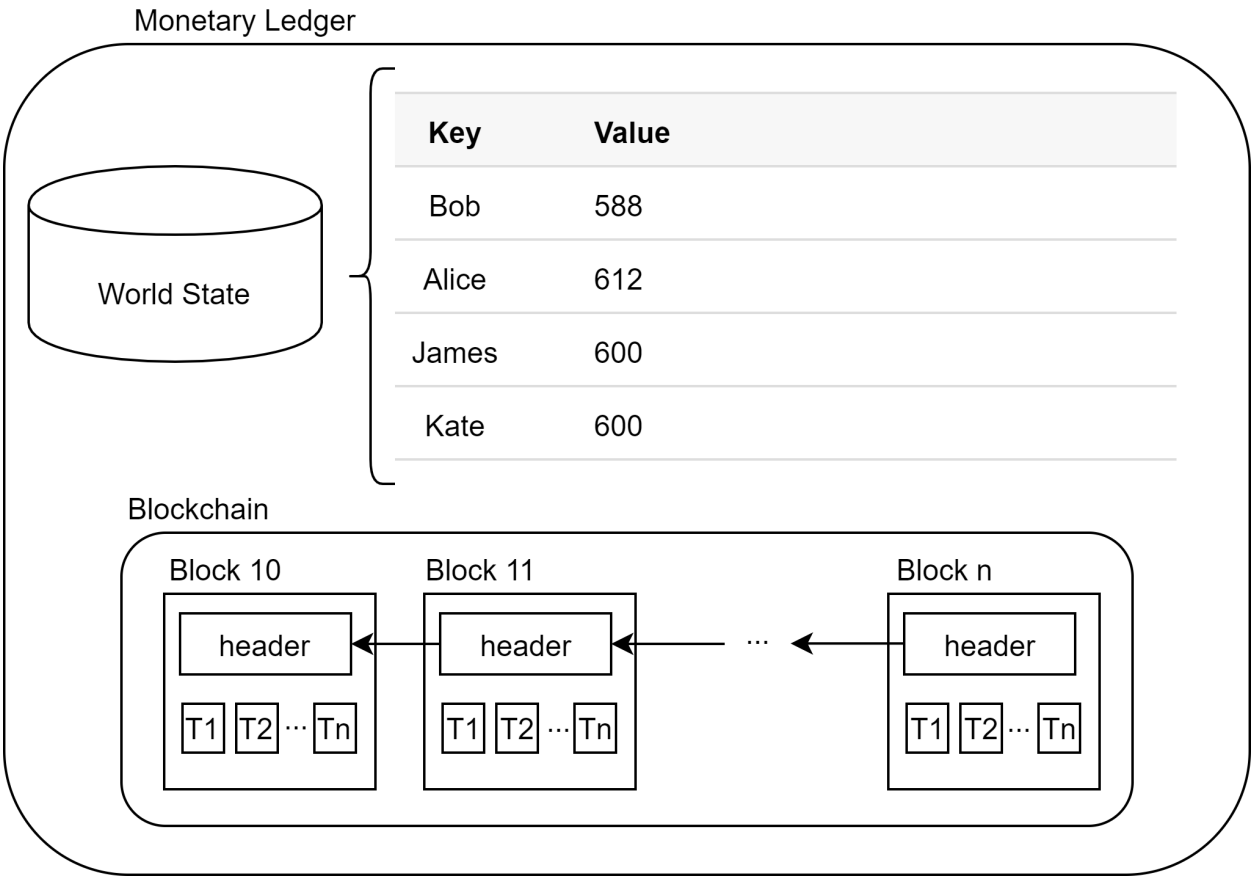
The blockchain used in ChainFaaS is implemented using Hyperledger Fabric V1.4. When choosing the blockchain solution, one of the most important criteria was its energy consumption. One of the initial motivations for developing this platform was to decrease the carbon emission of the ICT ecosystem by increasing the usage of personal computers. However, blockchains that use proof of work as their consensus mechanism, such as Bitcoin, require computationally powerful computers to solve meaningless puzzles to get rewards. On the other hand, in Hyperledger Fabric, there is no need for solving such problems since the consensus algorithm is not based on proof of work. Moreover, Hyperledger Fabric has been designed explicitly for enterprises and takes into account their needs. In Hyperledger Fabric, everything is highly configurable and can be customized for different use cases. These features make it an excellent choice for ChainFaaS. The reason for selecting V1.4 is its stability and long term support. It is the first version with long term support, while V2.0 is still under development.

In the current implementation of ChainFaaS, all components of the Hyperledger Fabric network have been deployed on an instance with 8 VCPUs, 16GB RAM, and 160GB disk with Ubuntu 18.04. There is one Fabric Certificate Authority (CA), one Solo orderer, and two organizations, each with two peers. In the future version of ChainFaaS, there can be included more of these components, each running on different computers. Anyone with a public IP can run the blockchain peers. Each peer stores all the latest information about the ledgers and verifies the new transactions.

We are using chaincodes, which are Hyperledger Fabric's smart contracts, to implement the functionalities needed for ChainFaaS in the blockchain. There are two main chaincodes: monitoring and monetary. The monitoring chaincode is responsible for keeping track of every job that has been served in ChainFaaS. The following image shows the monitoring ledger with an example. Anything that is stored on Hyperledger Fabric blockchain is shown by a key-value pair. In the case of monitoring ledger, the key is the job ID, and the value is its developer, provider, function ID, time, cost, received, and payment-is-done information. The time shows the time the provider spends on running the function, received shows whether the end-user has received the result, and payment-done shows whether the payment has successfully taken place. Monitoring ledger keeps track of all changes that happen to each job. Consider JOB7 as an example. Right now, the job has finished, but the end-user has not yet received the result. As soon as the end-user receives it, a new transaction is created that consists of the change of JOB7's received value from False to True. The world state database is part of the Hyperledger Fabric structure and stores the latest version of every job. In other words, if anyone follows all the changes in the blocks from genesis to the last block, they will reach the value inside the world state. This implementation makes it easy to query the latest values very fast.



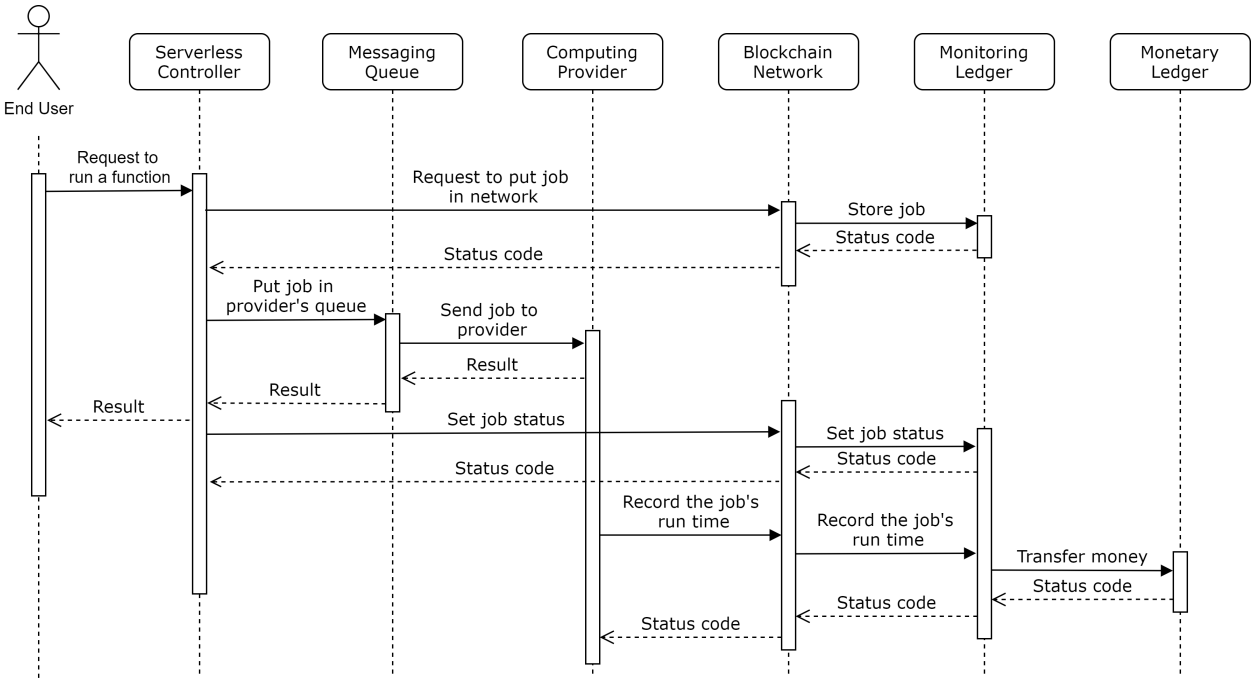
The monetary chaincode is responsible for keeping track of user account balances. The key in this ledger is the username of the user, and the value is how much they own in ChainFaaS. The following image shows the details of the monetary ledger. Similar to the monitoring ledger, the blockchain keeps track of changes that happen in accounts, and the world state stores the latest version of account balances. To better understand the functionality of the two ledgers, consider the examples shown in two images. Imagine that before JOB6 and JOB7, Bob, Alice, James, and Kate all had 600 units of money in their accounts. As soon as JOB6 is finished and received by the end-user, the cost (12 units) is deducted from Bob's account and added to Alice's account. Since the end-user has not yet received JOB7, the account balance of James and Kate has not changed.



4.4 The complete process

The sequence diagram shown in the following image describes the process in which the prototype of ChainFaaS serves a request from an end-user. In the current implementation, the result storage is the serverless controller.

When the serverless controller receives a request to run a function, the scheduler first finds an active provider capable of serving the request. When a provider is found, the controller asks the blockchain network to store the job on the monitoring ledger. In this step, the job is created in the ledger and information such as the developer, the provider who is supposed to execute the job, and the function ID is set. The controller then puts the job in the provider's messaging queue. The provider fetches the job from the queue, runs the Docker container, and sends back the results to the controller to be accessed by the end-user. As mentioned earlier, the serverless controller is set as the result storage for the prototype implementation. In the future versions of ChainFaaS, the result storage will be chosen by the developer. After receiving the result, the controller confirms that the result has been received by setting the status of the job in the blockchain network. In the meantime, the provider also sets the time it took to run the function. The provider is only able to set the run time if the job has already been created by the controller in the blockchain network. Moreover, only the provider that has been assigned to the job can set the run time metrics. This ensures that others cannot tamper with the job's information in the blockchain network. When the monitoring chaincode receives both the confirmation from the controller and the run time from the provider, the service fee will be transferred to the provider's account from the developer's wallet.

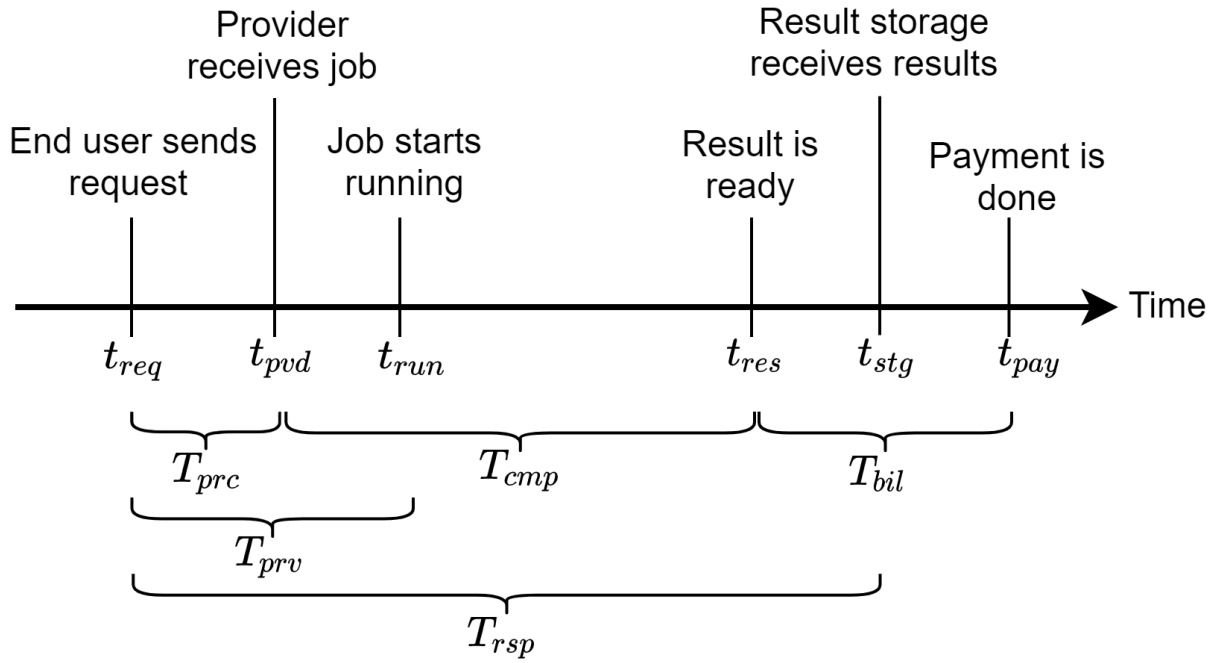

End User
Serverless Controller
Messaging Queue
Computing Provider
Blockchain Network
Monitoring Ledger
Monetary Ledger
Request to run a function
Request to put job in network
Store job
Status code
Status code
Put job in provider's queue
Send job to provider
Result
Result
Result
Set job status
Set job status
Status code
Status code
Record the job's run time
Record the job's run time
Transfer money
Status code
Status code
Status code

CHAPTER

FIVE

PERFORMANCE METRICS

A new job is created in the system as soon as a request for a function is received by the controller, as shown by `t_req` in the following image. The time it takes for the result storage to receive the result of the request is called response time `T_rsp = t_stg - t_req`. This time depends on both the request processing time of ChainFaaS and the completion time. The request processing time, T_prc, is defined as the time it takes for the provider to receive the job after the controller receives a request for the function. It corresponds to the time the serverless controller requires to schedule the job. The time it takes for the provider to pull and run the function is called completion time, `T_cmp`. This time depends heavily on the size of the developer's container, the network delay, and the job itself. Since response time depends on the function, it can only be measured for a specific workload and not in general.



Another important performance metric in this system is the provisioning time, `T_prv`. It is the time it takes for the job to start when a request is received. The provisioning time consists of the request processing time and the container pull time. Finally, from the provider's point of view, the time it takes for them to receive the payment after the job is finished, `T_bil`, is of great importance.

The most important performance metrics of ChainFaaS are summarized below:

- *Request processing time* (`T_prc`) is the time it takes for the provider to receive the job after the serverless controller receives a request.
- *Provisioning time* (`T_prv`) is the time it takes for the job to start running after the serverless controller receives a request.

- *Completion time* (`T_cmp`) is the time it takes for the provider to pull and run the job's container.
- *End-to-end response time* (`T_rsp`) is the time it takes for the result to be available in the result storage after the serverless controller receives a request.
- *Billing time* (`T_bil`) is the time it takes for the provider to receive payments after they are finished running the job.

CHAPTER

SIX

PUBLICATIONS AND CITATION

S. Ghaemi, H. Khazaei and P. Musilek, "ChainFaaS: An Open Blockchain-Based Serverless Platform," in IEEE Access, vol. 8, pp. 131760-131778, 2020, doi: 10.1109/ACCESS.2020.3010119.

You can use the following bibtex entry for citing our work:

```
@article{ghaemi2020@chainfaas,
author={S. {Ghaemi} and H. {Khazaei} and P. {Musilek}},
journal={IEEE Access},
title={ChainFaaS: An Open Blockchain-based Serverless Platform},
year={2020},
volume={8},
number={},
pages={131760-131778},
}
```