**Technical Guide**

**For**

**<Enter SWIP Title>**

|  |  |  |  |
| --- | --- | --- | --- |
| **SWIP ID** | SWIP-EIDA-2019-48 | **Unit / Dept.** | Cybersecurity |

**Copyright Notice:**

The Institute for Infocomm Research makes no warranty of any kind with regard to this material and shall not be liable for errors contained herein or for incidental or consequential damages in connection with the use of this material.

Please contact I2R, if you wish to use or reproduce material from this document.



**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Editor(s) / Author(s)** | **Date** | **Change description** | **Revision** |
| Deng Gelei | 19/11/2019 | Initial Input |  |
|  |  |  |  |
|  |  |  |  |

[1. Introduction 4](#_Toc10325964)

[1.1 Acronyms 4](#_Toc10325965)

[1.2 References 4](#_Toc10325966)

[2. Release Note 5](#_Toc10325967)

[2.1 Software Release Information 5](#_Toc10325970)

[2.2 List of open source/ 3](#_Toc10325971)[rd](#_Toc10325971) [party libraries with version and copyright information 5](#_Toc10325971)

[2.3 List of features/functionalities supported by the software 5](#_Toc10325972)

[3. Getting Started 5](#_Toc10325973)

[3.1 System configurations and minimum requirements 5](#_Toc10325974)

[3.2 How to build, install and run the software 5](#_Toc10325975)

[3.2.1 Step by step to build, install and run 5](#_Toc10325976)

[3.2.2 Quick start 5](#_Toc10325977)

[4. User Guide 6](#_Toc10325978)

[4.1 Step by Step user guide 6](#_Toc10325979)

[5. Technical details 6](#_Toc10325980)

[5.1 Software Architecture and design 6](#_Toc10325981)

[6. Test results 6](#_Toc10325982)

[6.1 Performance results 6](#_Toc10325989)

[6.2 Benchmarking 6](#_Toc10325990)

[Appendix A: Supporting Diagrams 7](#_Toc10325991)

# Introduction

The software utilizes blockchain system (Ethereum) and Interplanetary File System (IPFS) to achieve a full authentication and storage process for traditional database. Users are authenticated through Ethereum blockchain, and user database queries are logged into the same blockchain system. The database output is stored in IPFS, which is a distributed file storage system for better data security and availability.

The system is built based on the requirements from project where multiple factories need to share massive amount of raw sensor data from a centralized database, and store the useful contents into a file storage system. The authentication server as well as the file storage system should be distributed as none of the endpoint companies should have full control over the system. Inspired by this requirements, the Ethereum/IPFS based solution is discussed and designed.

## Acronyms

## References

* SWIP submission form
* <Reference documents if any>

# Release Note



## Software Release Information

|  |  |
| --- | --- |
| **Version of the released software** | V1.0 |
| **List of known bugs and any workaround** |  |
| **Limitations** | Web UI hang when data output is extremely large; Need further improvements in display. |
|  |  |

## List of open source/ 3rd party libraries with version and copyright information

PyEthereum: v1.4.0, MIT

Py-ipfs-http-client: v0.4.10, MIT

Flask: v1.1.1, BSD 3-Clause

## List of features/functionalities supported by the software

<Describe the features / functionalities the Software IP provides>

|  |  |  |
| --- | --- | --- |
| **SN** | **Feature/Functionality Name** | **Description of the functionality** |
| 1 | cURL command execution through web UI | User is able to communicate with database that supports http protocol through cURL command |
| 2 | User authentication through Ethereum | User is authenticated through Ethereum blockchain by providing the correct user account |
| 3 | Database query logging | User’s input is logged through Ethereum blockchain |
| 4 | IPFS file upload/download | User can upload and download (with correct hash value) from IPFS file storage system. |

# Getting Started

## System configurations and minimum requirements

The software is only tested on Linux Ubuntu 18.04, with python version 3.7.4 or later, yet ideally it can be installed on any Linux machine with cURL command support and python 3 installed.

The machine that serves as Ethereum server should have at least 8GB of memory for the purpose of building Ethereum DAG. There are no specific requirements for IPFS machine, yet please consider that IPFS server serves as the data storage system, which means sufficient disk space should be provided.

## How to build, install and run the software

### Step by step to build, install and run

**Installation manual for Ethereum system:**

Installing supporting libraries and Ethereum system

# /bin/sh

# Install Dependencies.

# sudo root

add-apt-repository -y ppa:ethereum/ethereum

apt-get update

apt-get install ethereum -y

apt-get install git

apt-get install python3-pip -y

pip3 install web3

# Start building up the blockchain

cd /home/blockchain/Ethereum

# 1. puppeth

# Please specify a network name to administer (no spaces, hyphens or capital letters please)

## > artc

# What would you like to do? (default = create)

## > 2

# Which consensus engine to use? (default = clique)

## > 1

# Which accounts should be pre-funded? (advisable at least one)

## > 0x

# Should the precompile-addresses (0x1 .. 0xff) be pre-funded with 1 wei? (advisable yes)

## > yes

# Specify your chain/network ID if you want an explicit one (default = random)

## >

# INFO [07-15|16:05:15.388] Configured new genesis block

#What would you like to do? (default = stats)

# 1. Show network stats

# 2. Manage existing genesis

# 3. Track new remote server

# 4. Deploy network components

## > Ctrl^C

# Depending on the settings, the genesis file will be most likely generated in ~/.puppeth/artc

# Export it (and copy to ~/Ethereum/ARTC), then go to ~/Ethereum/ARTC

geth --datadir . init artc.json

geth --datadir . account new

#Passphrase: artc

#Repeat the process and create three accounts.

#In artc.json, check chain ID (randomly generated).

#Based on ChainID, put the following data in startnode.sh

geth --networkid 21685 (chainID) --mine -minerthreads 2 --datadir "." --nodiscover --rpc --rpcport "8545" --port "30303" --rpccorsdomain "\*" --nat any --rpcapi eth,web3,personal,net --unlock 0 --password ./password.sec --ipcpath "~/.ethereum/geth.ipc" --allow-insecure-unlock

#Also create password.sec with password "artc" inside

#give access to startnode.sh

chmod +x startnode.sh

./startnode.sh

##########################################

# Installation of Smart Contract

# Install solc

sudo apt install curl

sudo apt-get install npm

npm install solc@0.4.25

pip install py-solc

python -m solc.install v0.4.25

(or alternatively: curl -o /usr/bin/solc -fL https://github.com/ethereum/solidity/releases/download/v0.4.25/solc-static-linux \ && sudo chmod u+x /usr/bin/solc)

##########################################

# Installation of flask and supporting libraries

pip install flask

pip install flask-restful

pip install flask\_sqlalchemy

**Installation manual for IPFS system:**

sudo apt-get update

sudo apt-get -y upgrade

sudo apt install git

sudo apt install make

sudo apt install gcc

pip3 install ipfscluster

Install Go

wget [https://dl.google.com/go/go1.11.4.linux-amd64.tar.gz](https://dl.google.com/go/go1.11.4.linux-amd64.tar.gz" \t "https://outlook.office.com/mail/search/id/_blank)

sudo tar -xvf go1.11.4.linux-amd64.tar.gz

sudo mv go /usr/local

mkdir $HOME/gopath

sudo nano $HOME/.bashrc

[Add to bottom of .bashrc file:]

export GOROOT=/usr/local/go

export GOPATH=$HOME/gopath

export PATH=$PATH:$GOROOT/bin:$GOPATH/bin

[Save and close file]

source ~/.bashrc

go version

Install IPFS

wget https://dist.ipfs.io/go-ipfs/v0.4.22/go-ipfs\_v0.4.22\_linux-amd64.tar.gz

tar xvfz go-ipfs\_v0.4.22\_linux-amd64.tar.gz

sudo mv go ipfs/ipfs /usr/local/bin/ipfs

ipfs init

ipfs version

[Generate the swarm key **once**and make sure the **same** file exists in all machine’s ~/.ipfs/ directory.]

go get -u github.com/Kubuxu/go-ipfs-swarm-key-gen/ipfs-swarm-key-gen

ipfs-swarm-key-gen > ~/.ipfs/swarm.key

export LIBP2P\_FORCE\_PNET=1

git clone https://github.com/ipfs/ipfs-cluster.git $GOPATH/src/github.com/ipfs/ipfs-cluster

cd $GOPATH/src/github.com/ipfs/ipfs-cluster

make install

ipfs-cluster-service --version

ipfs-cluster-ctl --version

source ~/.bashrc

### Quick start

Run Ethereum system with UI

# locate into the software root directory

python3 app.py

# Then navigate to http://127.0.0.1:5000/

Run IPFS file storage system with UI

# locate into software root directory

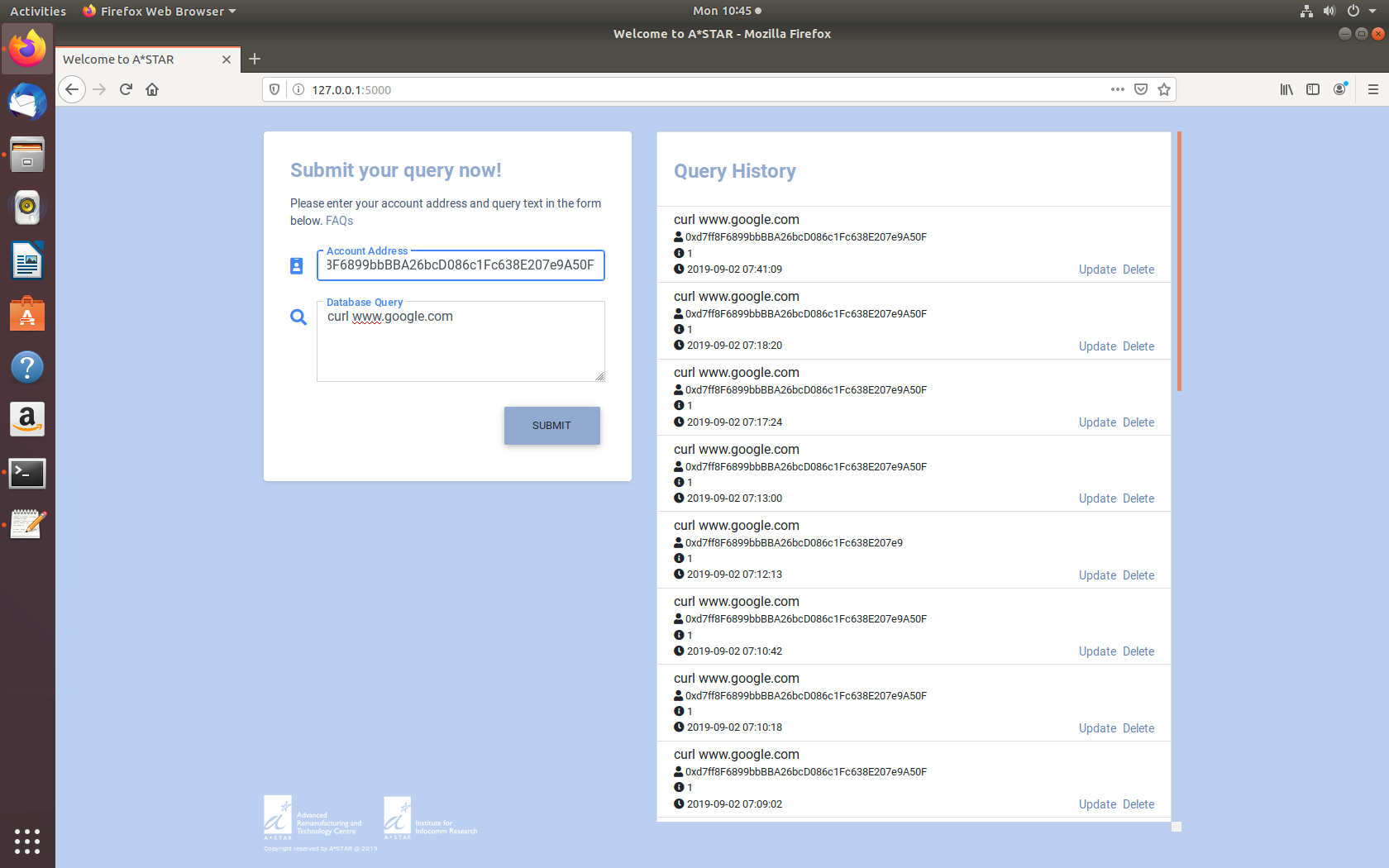
ipfs daemon

ipfs-cluster-service daemon

python3 http\_server.py

# User Guide

## Step by Step user guide

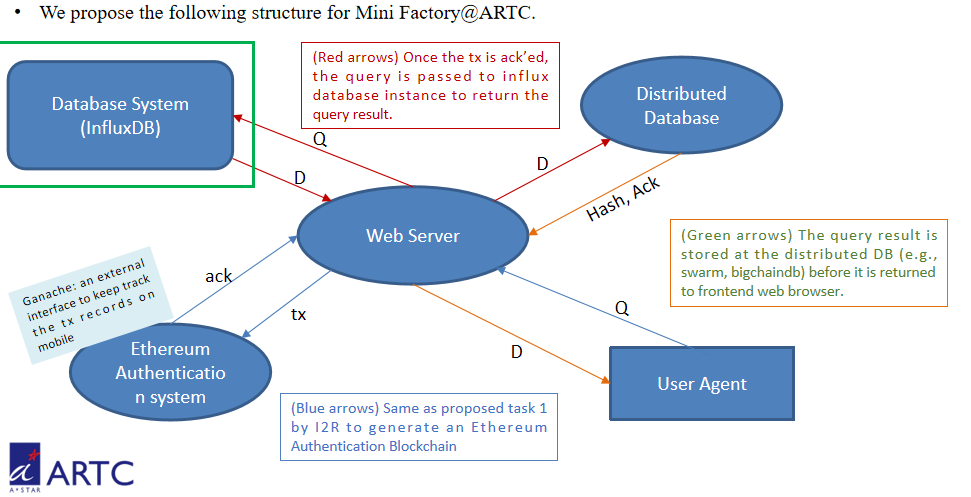
The following figure is the screenshot of the web user interface that users can normally access. 

To start using the service, the user visit port 5000 on the Ethereum server machine, and key in the authentication Ethereum account in the Account Address section. Then, user should key in the database query as required by the database (normally curl command). By clicking on SUBMIT button, the user account address will be checked, and the database query will be executed as long as it passes the command check.

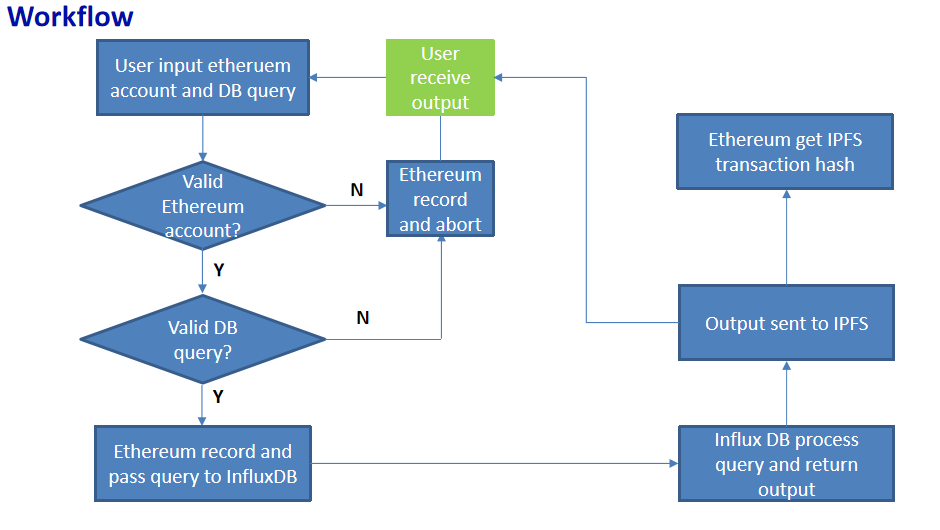
# Technical details

## Software Architecture and design

The detailed software architecture and design can be found in the attached power point slides, which is the summary slides for the whole project architecture. The key design architecture is attached below:



*Fig. ARTC MiniFactory Database architecture*



*Fig. Ethereum/IPFS system workflow*

# Test results



## Performance results

For testing purposes, the above mentioned Ethereum/IPFS systems are installed on two PCs in Cybersecurity lab. Furthermore, the IPFS system simply act as a data storage center, so the performance testing is focused on Ethereum PC. The PCs’ specifications are listed below:

**Operating System: Ubuntu 18.04**

**CPU: Intel CPU [G1840@2.8GHz,](mailto:G1840@2.8GHz,) 2 cores, 2 threads**

**RAM: 16GB**

The hardware performance of the system is monitored through the linux command line tool: htop. To be more specific, the system performance is checked at the beginning of the testing, 1 day after service starting, and 4 days after service starting. To save space, screenshots are attached in the same folder, where the system running time can be identified from the information provided by the screenshot.

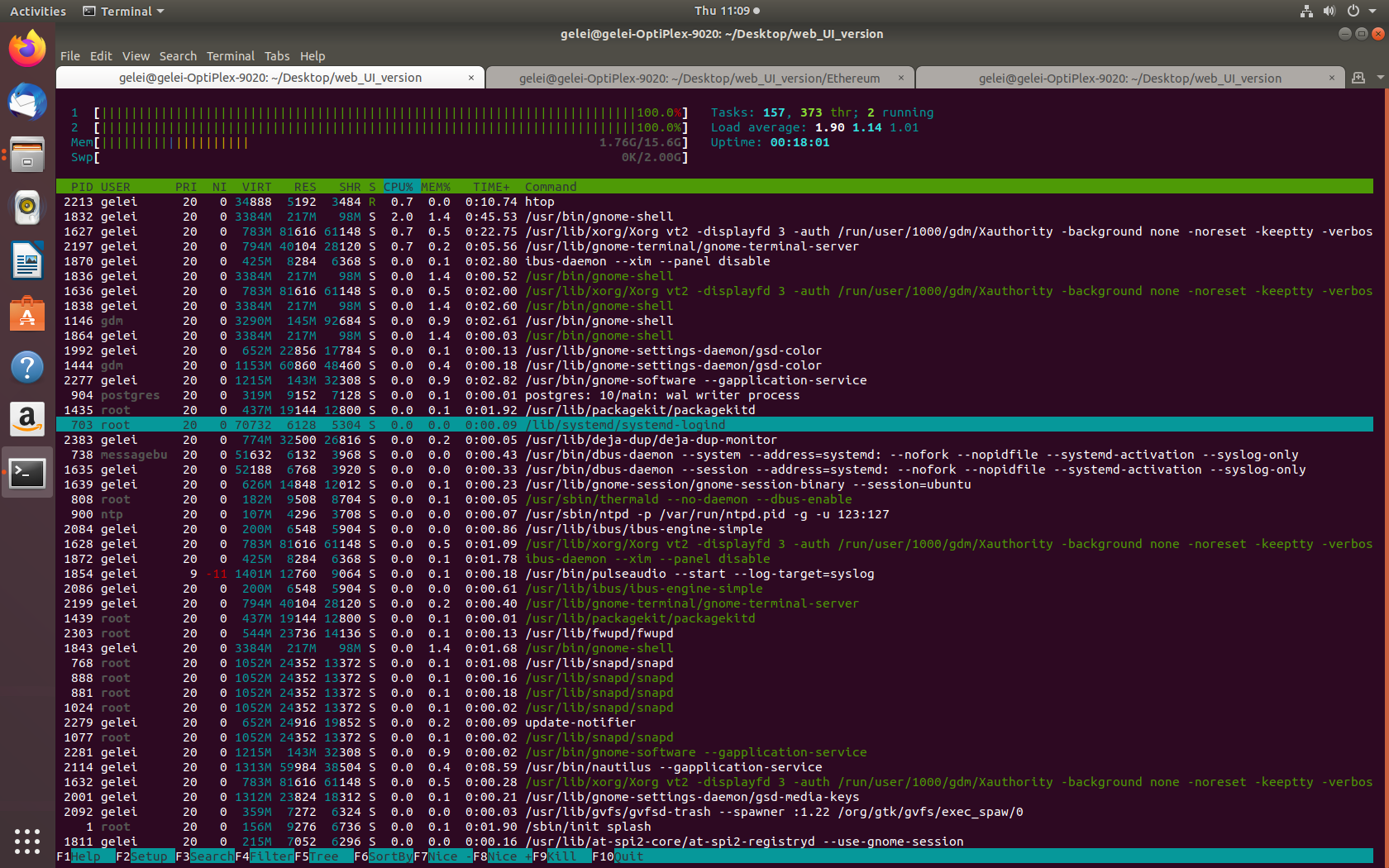
As a conclusion, the RAM usage does not increase significantly while program running. The services maintains low latency time and high response rate after continuous operation. Yet, a in-built cache cleaning algorithm would still be preferred in the future development work, if applicable.

## Benchmarking

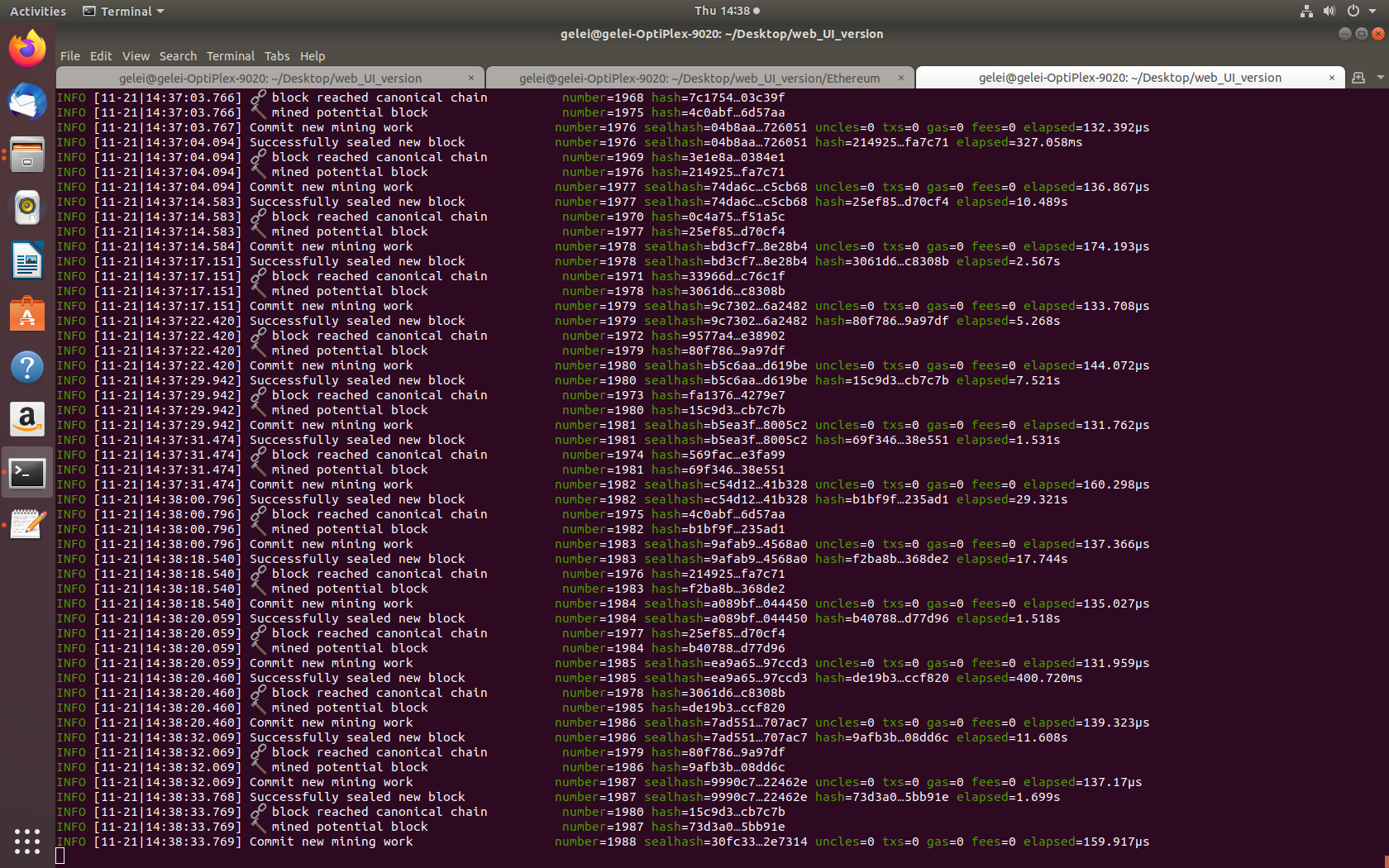
To the best of our knowledge, there are no existing solutions that provide the same functions as our software solution. The main reason is probably because that the proposed solution is customized for MiniFactory project @ ARTC.

# 

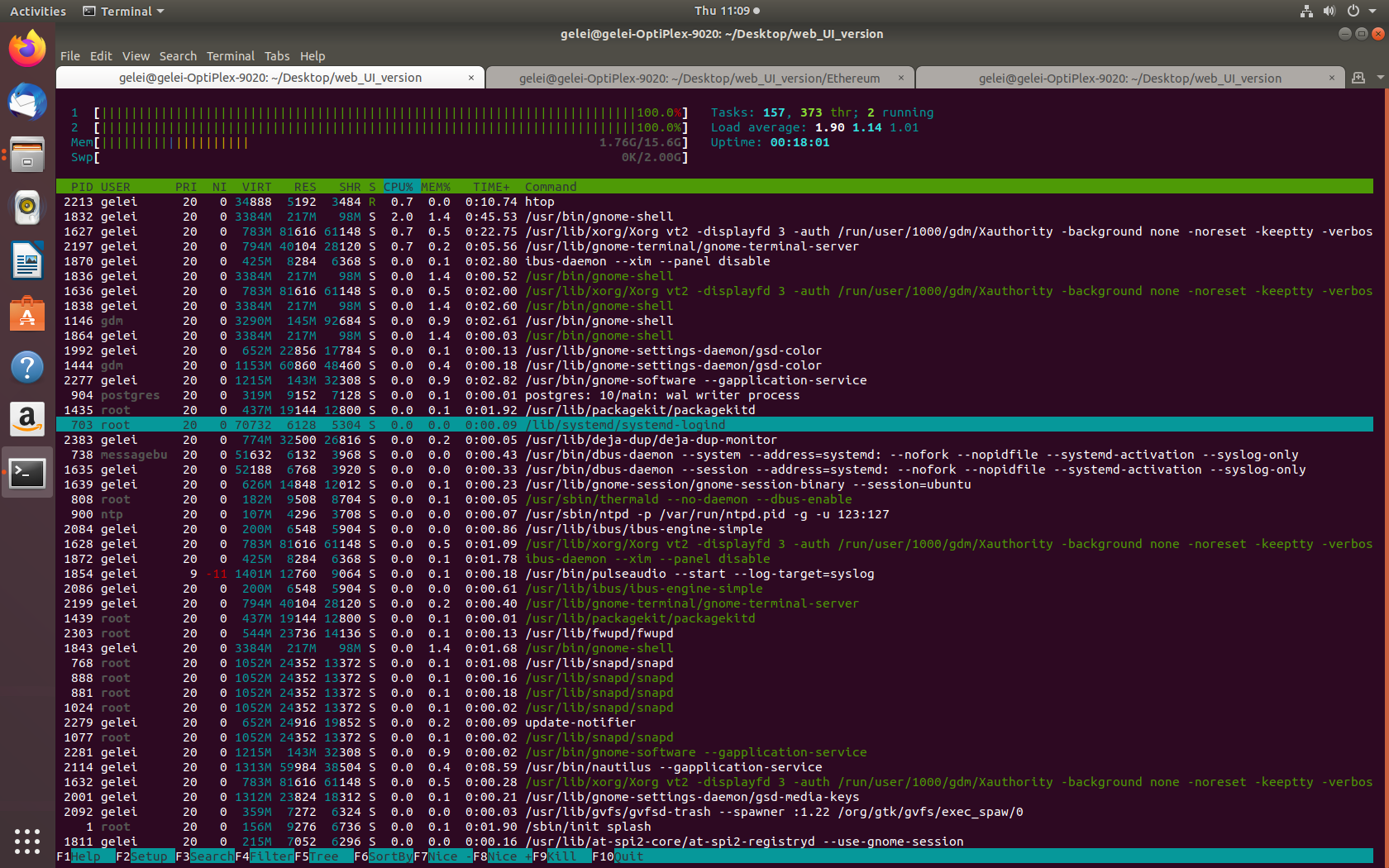
# Appendix A: Supporting Diagrams



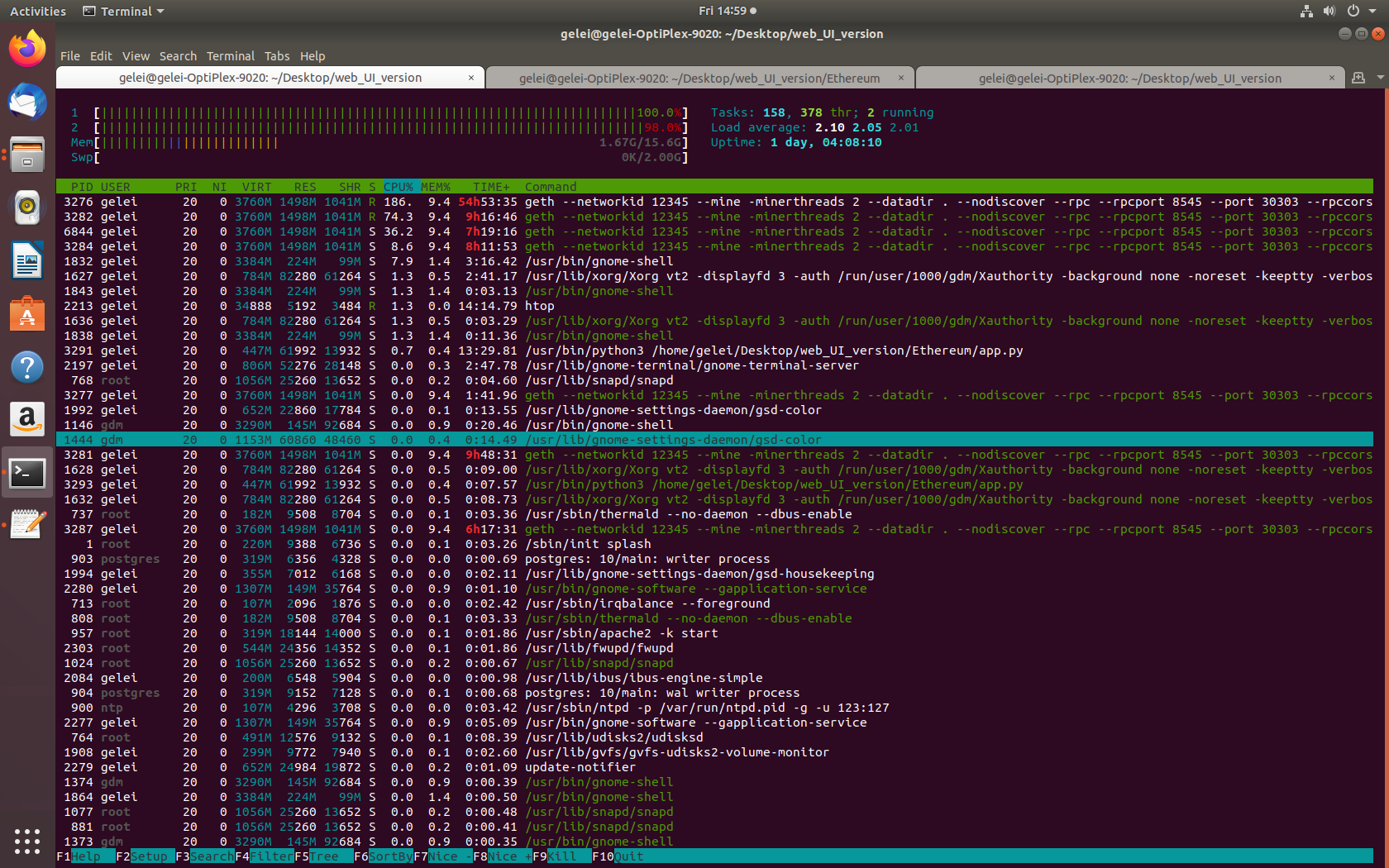
Before starting Ethereum service



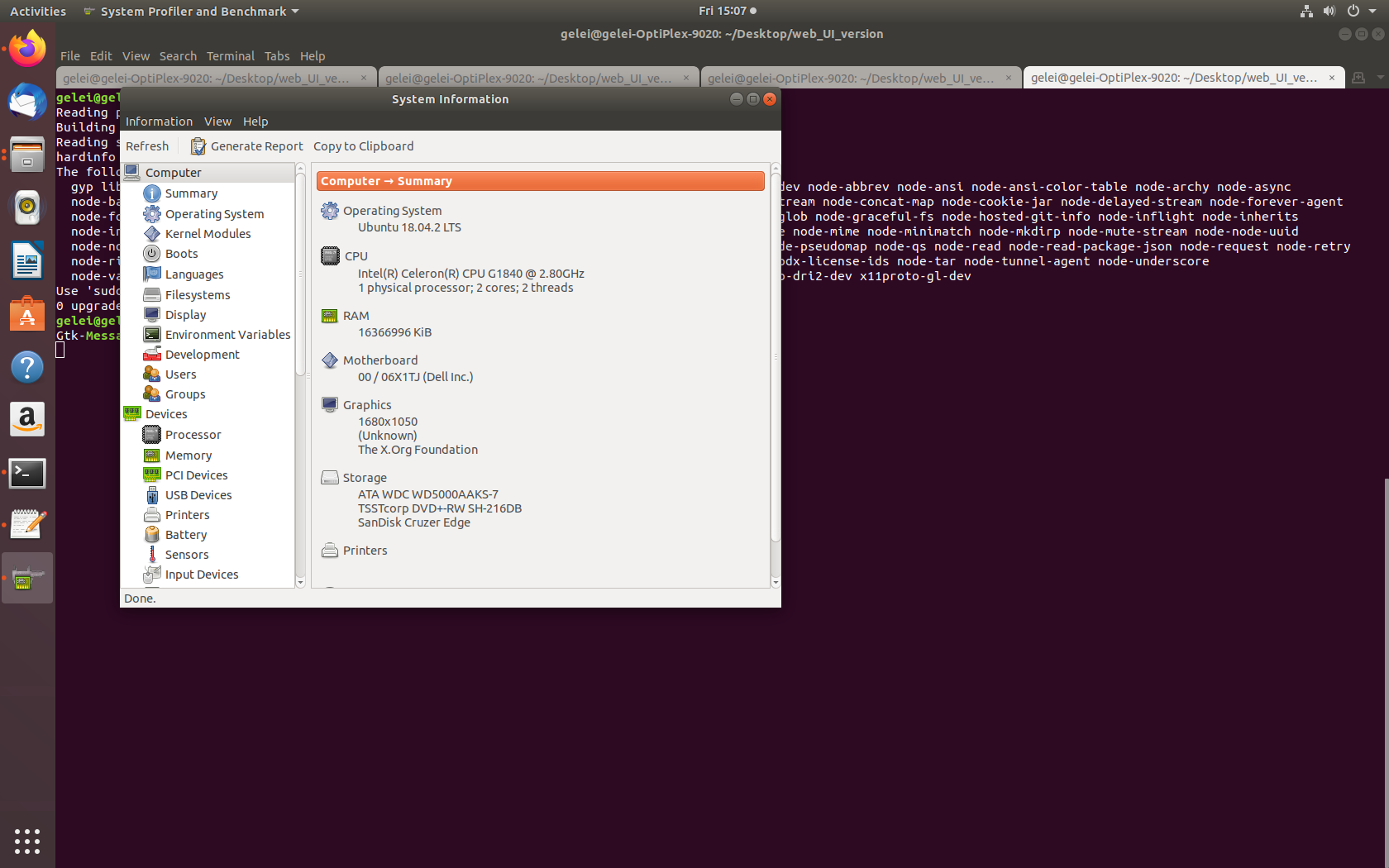
After starting Ethereum service, PC start to mine



After starting Ethereum service



1 day after starting Ethereum service



System hardware