HYPERLEDGER

Preliminary Notes

We now introduce the Hyperledger Umbra (Umbra)[1] project, a project initiated as an Hyperledger Internship. This laboratory has been created with the cooperation of Raphael Rosa, creator, and maintainer of Umbra. Please, mind the official documentation page for eventual questions[2], and the official channels to communicate issues[3]. Beware with copying -paste the commands in this lab to the terminal: the buffer might contain invalid characters that corrupt the command. We recommend writing the command instead of copying it.

During this laboratory, start building an answer to the following question: **What are the advantages and disadvantages of Umbra, regarding the evolution of enterprise blockchain technologies?**.

1 Hyperledger Umbra Overview

The simulation research internship during the summer of 2018 led to the creation of the Hyperledger Umbra Lab (Umbra). Due to the overall difficulty of getting Hyperledger blockchain frameworks running under the Shadow simulation tool, work on the Umbra lab has slowed to a crawl. A different network emulation tool called Mininet was proposed as an alternative to using Shadow, and it has the potential to drastically reduce the startup cost of getting a network emulation tool running Hyperledger blockchains.

1.1 Scope

Umbra is a platform employing Mininet and plugins for Hyperledger distributed ledgers to run under emulation in a lab environment.

It is intended to be an ongoing project to provide a research tool for understanding and improving the Hyperledger blockchain platforms and conducting future research in consensus algorithms, scalability, security, etc.

1.2 How does it Work

Umbra works with the support of virtualization technologies, containers (Docker), and programmable switches (Open vSwitch). Using containernet, it deploys an underlying network that serves as the infrastructure for the blockchain platform of choice (e.g., Iroha, Fabric, Indy, etc.) to be executed as the overlay application network. Nodes and links can be configured with resource constraint rules (e.g., CPU, memory, bandwidth, latency, etc.). Besides, umbra allows events (e.g., transactions, chaincode invoke, etc.)

[1] https://github.com/hyperledger-labs/umbra

[2] https://umbra-labs.readthedocs.io/en/latest/intro.html

[3] https://github.com/hyperledger-labs/umbra/issues

to be scheduled targeting the blockchain platform using plugins. Umbra is supporting by Mininet.

Mininet was developed for fast prototyping of emulated programmable networks on a laptop. Later there were different extensions proposed on top of it, such as maxinet enabling experiments in distributed clusters of servers and containernet enabling the experimentation with Docker containers. Mininet was developed for high fidelity and later on extended to support the features proposed by Maxinet. Containernet was built on top of mininet version 2.2.0, therefore inheriting its most recent enhancements. Umbra elaborates its architecture on top of the upgrades proposed by containernet. As being evaluated, possible contributions to containernet will be performed to enhance it with the most recent features provided by Docker (i.e., current docker-py API) and mininet (i.e., currently in version 2.3).

2 Getting Started With Hyperledger Umbra

To get started with Umbra, please go to the official Umbra's repository[4], and clone it (tag umbra-course). Alternatively, you can fork it and then clone it.

Umbra is developed and tested in Ubuntu 20.04. The hardware requirements needed for umbra will depend on the scale of the experiments to be played with it (e.g., number of nodes in a blockchain topology, amount of events triggered into the topology, the topology resource settings, etc.). It is recommended for a simple setup to have available at least: 4 logical CPU cores, 8 GB of RAM, and 10GB of storage.

2.1 Installing Hyperledger Umbra

To install Hyperledger Umbra, first, we need to install prerequisites. Please run:

```
sudo apt install make
```

To obtain Umbra's code, run:

```
git clone https://github.com/raphaelvrosa/umbra
```

Go to the projects' folder:

```
cd umbra
```

Given the installation of the umbra requirements, the commands below install umbra.

```
sudo make install
```

ALTERNATIVELY, Umbra can also be installed using a Vagrant virtual machine, either using qemu-kvm/libvirt or virtualbox as providers, as stated below:

[4] https://github.com/raphaelvrosa/umbra

```
sudo make vagrant-run-libvirt # Installs umbra in a virtual machine using qemu-kvm/li
sudo make vagrant-run-virtualbox # Installs umbra in a virtual machine using virtualb
```

2.2 Generating the experiments

Having umbra installed, it is possible to experiment with it using the provided examples, inside examples/ folder. To do this, run:

```
cd examples
```

Now, let us examine the file "local-2orgs.py", under the *fabric* folder, which sets up a configuration file that defines a network with two organizations. At the core, this script builds a network configuration (fuction build), and logs the procedure, by importing a logger:

```
    setup_logging()
    builds()
```

Next, focus on the first lines of the same file:

```
# Then, import the configtx definitions, which are going
# to be used by each one of the orgs policies, and also
# for the whole construction of the configtx.yml file.
# Each experiment/topology can have its own configtx custom definitions.

from base_configtx.configtx_2orgs import (
    org1_policy,
    org2_policy,
    orderer_policy,
    configtx,
)
```

This code sample imports pre-defined policies to generate a two-org Fabric network. The next steps happen throughout the file:

```

    fab_topo = FabricTopology("local-2orgs", chaincode_dir=chaincode_dir)

    experiment = Experiment("local-2orgs")
    experiment.set_topology(fab_topo)

```

```
7       fab_topo.add_network("s1", envid="umbra-default")
8
9       fab_topo.add_org("org1", domain, policies=org1_policy)
10      fab_topo.add_peer(
11          "peer0", "org1", anchor=True, profile="nodes", image_tag=image_tag
12      )
13
14      [...]
15
16      fab_topo.configtx(configtx)
17      p1 =
        ↪  "TwoOrgsOrdererGenesis.Consortiums.SampleConsortium.Organizations"
18      p2 = "TwoOrgsOrdererGenesis.Orderer.Organizations"
19      p3 = "TwoOrgsChannel.Application.Organizations"
20      fab_topo.set_configtx_profile(p1, ["org1", "org2"])
21      fab_topo.set_configtx_profile(p2, ["orderer"])
22      fab_topo.set_configtx_profile(p3, ["org1", "org2"])
23
24      # The interconnection of umbra orgs/orderer to the network must be
        ↪  defined.
25      # When an org is connected to a network, all its peers/CAs are
        ↪  connected to the network too.
26      fab_topo.add_org_network_link("org1", "s1", "links")
27      fab_topo.add_org_network_link("org2", "s1", "links")
28      fab_topo.add_org_network_link("orderer", "s1", "links")
29
30      node_resources = fab_topo.create_node_profile(cpus=1, memory=1024,
        ↪  disk=None)
31      link_resources = fab_topo.create_link_profile(bw=1, delay="2ms",
        ↪  loss=None)
32
33      experiment.save()
34
35
```

Line 1 defines a Fabric topology by setting its name (local-2orgs) and chaincode directory. The topology is the definition of a Fabric network comprising the orgs, peers, CAs, and orderers. This is the building base of an Experiment (lines 2 and 3), along with Events.

Environments in umbra are the places (i.e., baremetal servers and/or virtual machines) where the components of umbra are executed, and consequently, the topology itself. An environment can be remote or local (remote parameter set to true or false). All the proper settings regarding the reachability of the nodes, network, environment are

handled by umbra. The network must be associated with the environment where it is going to be placed/executed. All the nodes connected to the network will be deployed in the environment where the network is placed.

Eventually, an org is added (line 8), as well as its peers (line 9), orderer, and CA (see example). For each organization, two peer nodes and a CA is added to the topology. An orderer is added as well, with a predefined policy, *orderer_policy*.

The script loads the *configtx* file, a specification that contains the definition of the profiles to be used by the network to generate the fabric topology artifacts (lines 15-21). An interesting aspect about Umbra is that we can define the links between an org and the network, so later we can alter them. We can define resources allocated for each node (line 29) and network parameters (line 30). In future experiments, this allows us to understand what happens if a node experiences a degradation in terms of performance or network resources (e.g., bandwidth). Finally, the experiment is persisted in the form of a configuration file that Umbra can parse (line 32).

To generate this configuration, run:

```
python3 examples/fabric/local-2orgs.py
```

According to the official documentation, "The command will compile the experiment's definition (and its artifacts) and save it in /tmp/umbra/. The configuration is saved under a folder of its referenced blockchain project, with the name used to define the experiment, and in a JSON format. For instance, the command above configuration will produce a folder /local-2orgs inside the folder /tmp/umbra/fabric/. In the folder /tmp/umbra/fabric/local-2orgs/ a file named local-2orgs.json is the one that defines all the configuration of the experiment compiles. This file references all the generated topology (nodes and links) and their respective resource profiles and artifacts (e.g., certificates, crypto keys, genesis block)."

This configuration can be loaded by Umbra, as we will see in the next section.

2.3 Running Umbra

Running the configurations Start the umbra-cli component:

```
umbra-cli --uuid umbra-cli --address 127.0.0.1:9988
```

You should see the following output on your terminal:

```
1
2 rafael@x:~/Projects/umbra$     umbra-cli --uuid umbra-cli --address
  ↪ 127.0.0.1:9988
3
4                 <<< Welcome to Umbra >>>
```

```
:umbra>
```

Executing the command above a command-line interface (CLI) prompt will start. To exit it, just type CTRL+D. Using the umbra-cli all the interactions with an umbra experiment and its environments are possible. The uuid and address fields are needed because umbra-cli uses those parameters to receive status logs from umbra-broker.

In umbra-cli, load your configuration:

```
umbra-cli> load /tmp/umbra/fabric/local-2orgs/local-2orgs.json
```

The expected response is:

```
:umbra> load /tmp/umbra/fabric/local-2orgs/local-2orgs.json

-> task: Loading configuration file at
↪  /tmp/umbra/fabric/local-2orgs/local-2orgs.json
-> result: Configuration loaded

:umbra>
```

Loading the configuration means umbra-cli is ready to work with it, meaning the installation of umbra in the defined configuration environments, the start of the umbra components needed in each environment, and the topology's instantiation its events as programmed by the experiment. Likewise, the tear-down of the topology, stop of the components, and uninstall of umbra in the environments can be executed with umbra-cli.

In umbra-cli, install the environments of your configuration:

```
umbra-cli> install
```

The expected response is:

```
:umbra> install

: Installing :

-> task: Installing Umbra at environment umbra-default
[sudo] password for rafael:
-> result: Install Umbra in environment umbra-default Ok

:umbra>
```

The install command means umbra is going to reach all the environments defined by the configuration, included umbra-default, and install umbra and the dependencies needed to run the experiment in the configuration's assigned environments. For instance, if an experiment requires fabric dependencies, all the fabric container images will be downloaded in the environment(s) needed to run the configuration.

In umbra-cli, start the components of your configuration:

```
umbra-cli> start
```

The expected response is:

```
:umbra> start

: Starting :

-> task: Starting component 'umbra-broker' in environment umbra-default
-> result: Started component 'umbra-broker' in environment umbra-default
 ↪  Ok


-> task: Starting component 'umbra-monitor' in environment umbra-default
-> result: Started component 'umbra-monitor' in environment umbra-default
 ↪  Ok


-> task: Starting component 'umbra-scenario' in environment umbra-default
-> result: Started component 'umbra-scenario' in environment umbra-default
 ↪  Ok

:umbra>


```

When the start is called, all the components in all the environments are initiated. Each of those components is a process (e.g., umbra-monitor, umbra-scenario, umbra-broker) with an UUID and address.

In umbra-cli, begin the experiment:

```
umbra-cli> begin
```

The expected response is:

```
1
2  :umbra> begin
3
4  : Beginning :
5
6  -> task: Experiment Begin
7  -> result: Umbra Experiment Ok
8
9  :umbra>
10
```

The beginning of an experiment means the instantiation of the topology, and the triggering of its events went fine. Here, the experiment indicates that the topology was created.

Congratulations! You have an entire Fabric network running over Umbra.

2.4 Shutting Down Umbra

In umbra-cli, after performing the topology and the events' instantiation and the events, it is possible to end the experiment. The command below indicates umbra-broker to tear down the instantiated topology and stop monitoring its components (environments and containers).

```
umbra-cli> end
```

In umbra-cli, stop the components of your configuration:

```
umbra-cli> stop
```

Stop means all the components will be finished in their execution environment.
Type ctrl+d to exit umbra-cli:

```
umbra-cli>  <ctrl+d>
```

If all went fine, this should be what you saw:

```
1  :umbra> end
2
3  : Ending :
4
5  -> task: Experiment End
```

```
-> result: Ended Umbra Experiment

:umbra> stop

: Stopping :

-> task: Stopping component 'umbra-broker' in environment umbra-default
[sudo] password for rafael:
-> result: Stopped component 'umbra-broker' in environment umbra-default
 ↪  Ok


-> task: Stopping component 'umbra-monitor' in environment umbra-default
-> result: Stopped component 'umbra-monitor' in environment umbra-default
 ↪  Ok


-> task: Stopping component 'umbra-scenario' in environment umbra-default
-> error: Stopped component 'umbra-scenario' in environment umbra-default
 ↪  Error

:umbra> <CTRL+D>

        <<< See you soon! Cheers, Umbra >>>
```

2.5 Exercises

What are the advantages and disadvantages of Umbra, regarding the evolution of enterprise blockchain technologies?

Creating a topology with Umbra

Setup an Umbra topology with 3 orgs, 2 peers each, one orderer, and one CA per org. The endorsement policy should be at least two orgs.

Hint: Create a Python file called *local-3orgs.py*, and start from the studied file. Duplicate the file configtx_2orgs.py under fabric/base_configtx, and rename it to configtx_3orgs.py. Adapt it to your needs.