Hyperledger Umbra: Simulating Hyperledger Blockchains using Mininet

Overview

This document contains the roadmap to accomplish the topics proposed by the 2019 internship project named Hyperledger Umbra.
It sets the project foundation elements, the planned architecture, and different views to be developed during the project execution.
Mostly important, this document defines the activities planned to realize the internship project.

Foundation

Mininet - http://mininet.org/
Maxinet - http://maxinet.github.io/
Containernet - https://containernet.github.io/

Mininet was developed for fast prototyping of emulated programmable networks in 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.

This project aims to elaborate its architecture on top of the enhancements proposed by containernet. As being evaluated, possible contributions to containernet will be performed in order to enhance it with the most recent features provided by Docker (i.e., current docker-py API) as well as mininet (i.e., currently in version 2.3).

Architecture

The prototype will be mainly written in Python 3, having its configuration files written in YAML.
The configuration file will be parsed as input to the main management module of the architecture to realize a test.
A test will deploy the topology and execute the events programmed in the configuration file.
A test scenario will be deployed using the containernet platform. The configuration file will define the structure (i.e., network topology), the set of hyperledger components and their roles running on Docker containers, and specified resource allocations for nodes and links.

During the tests, there will be modules performing the stimulus on the topology and its components, modules monitoring the test components and particular topology structures, and modules realizing the analysis of the collected data during the test and after it.
Stimulus refers to a set of actions performed by hyperledger nodes, and actions executed on the topology itself (e.g., node/link failures, adjustments in node/link resources).
Monitoring refers to the collection of node/link metrics (e.g., cpu percent, packets transmitted, etc) and particular hyperledger events (e.g., transaction confirmation time).
Analysis refers to the possible algorithms (e.g., statistics, machine learning) that might be applied over the collected data from monitoring during and after a test.

How it is going to work: the configuration file is parsed and converted into an internal data structure that is used to instantiate the specified topology. Executed the topology, the events specified in the configuration file take place. During the test, monitoring functions collect metrics to be shown in graphics online. After the test, the analysis of the whole monitored data is analyzed and a report is generated containing the test life cycle (i.e., status involving the phases of pre-deployment, execution, and post-mortem).

Use-Case View

The tests are meant to be fully automated. However Testers might interact with the running topology during the tests too.
A Tester defines the configuration file, performs the test in a target execution environment, and collects the experimental data (raw and analyzed) output of the monitoring functions.
A Viewer/User just visualizes and performs analysis on the test results. Visualization of test results can be triggered during the execution of the test and after it.

Logical View

Defines the project classes, their organization and their relationship in form of a use-case realization.
To be defined.

Process View (Test Execution Flow)

Shows how a project instance runs a test case, as stated in the following sequence diagram.

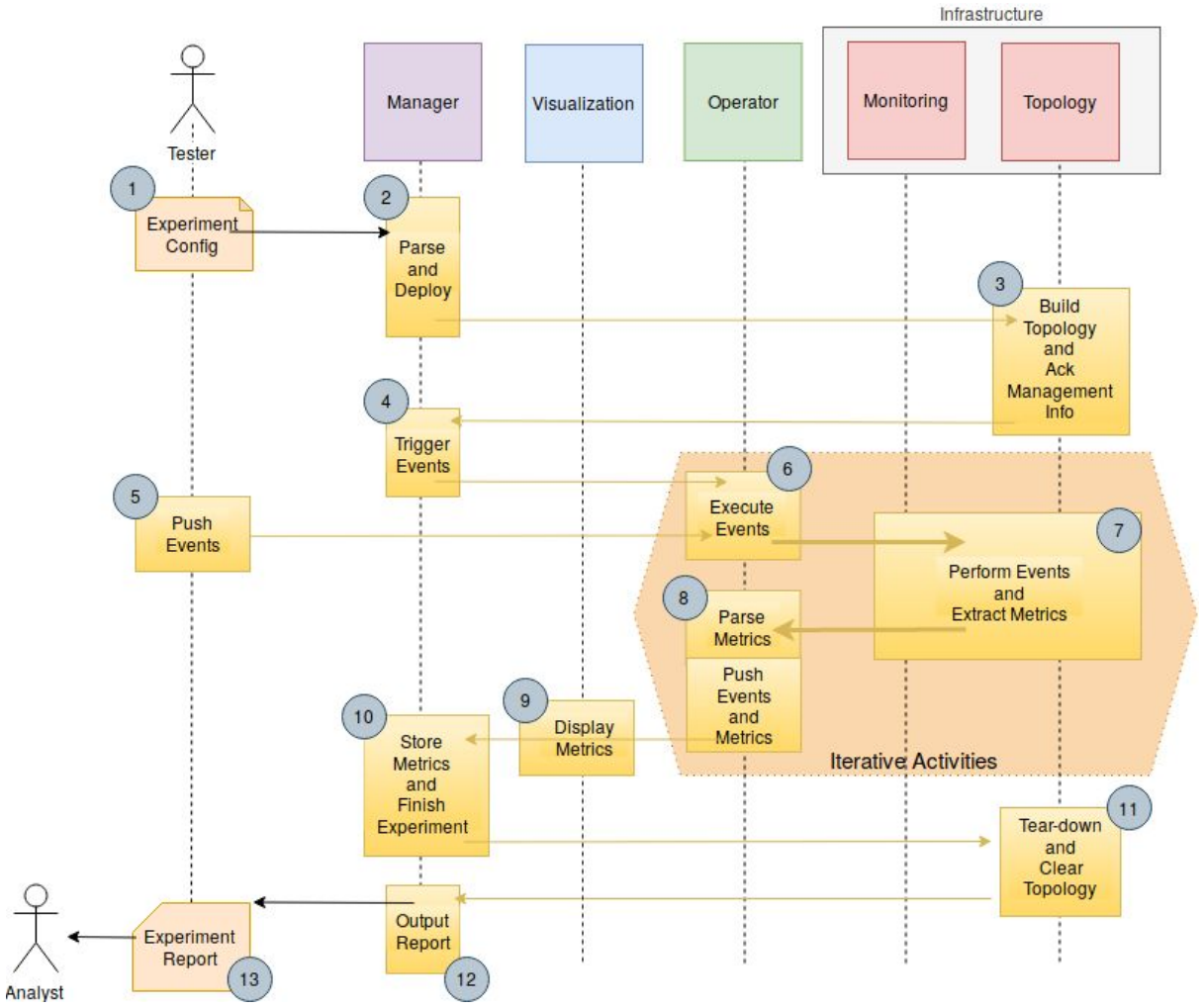

Figure 1. Sequence Diagram of an Experiment

To explain the process view shown above in Fig. 1, the items below make reference to each one of the numbered circles. It is considered the infrastructure is already set up with all the components installed and ready to execute the experiment.

1. The Tester defines the experiment configuration that will be executed. It contains the whole lifecycle of the structural (topology/infrastructure) and functional (events/visualization) definitions.
2. Given such configuration to the Manager component, it will execute the parsing of the configuration, set all the deployment configuration to be deployed.
3. A Topology component will deploy the given configuration file, creating the Containernet network. It will send back to Manager the management information of the deployed components. Such information contains among other settings, the IP address of the management interface of the containers deployed in the network.
4. Manager will start triggering the scheduled events (defined in the Experiment Configuration) to take place in the infrastructure and/or topology.
5. In addition, the Tester can utilize the Manager interface to trigger events during the execution of the experiment.

6. The Operator component then parses such event requests and schedule their occurrence in the topology and/or infrastructure.
7. Events take place, e.g.: containers running hyperledger project components start/stop mining, links/nodes fail/restore or have resources changed, metrics are monitored from infrastructure perspective, etc.
8. Metrics, output of events execution, are received in Operator. The set of events-metrics pairs are pushed to Manager, and possibly a Visualization component.
9. Metrics are displayed in different graphic formats according to the pushed instructions.
10. Given the schedule of the Experiment Configuration or a request from the Tester, Manager stores all events-metrics, and request the end of the topology/infrastructure components.
11. All the components are stopped and removed, leaving the infrastructure as it was before the experiment started.
12. Acknowledged the end of the infrastructure/topology execution, Manager outputs a Experiment report containing the comprehensive set of events and their collected metrics during the whole experiment.
13. Tester receives the output report and possibly send it to an experiment Analyst/Viewer.

Deployment View

Shows how the project components can be deployed in an execution environment. In general, the project code can be installed in one or more servers interconnected by a common network, composing a cluster. In one server, specified as the jump server, the main modules of the project load the configuration file and execute the test.

Project Directory Structure

To be defined.

Activities

- ➔ Build reference architecture (Logical View and Process View)
  - ◆ Define modules and their components (initial directory structure)
  - ◆ Define basic structure of classes and their relationship
  - ◆ Define interactions among test components (events and interfaces)
    - e.g., Json messages via RPC interfaces?
  - ◆ Code basic structure of modules
  - ◆ Define installation requirements and scripts for the reference architecture
- ➔ Evaluate Fabric project with reference architecture
  - ◆ Define requirements to run Fabric
    - e.g., configuration files, roles of components, number of channels, number of organizations, chaincode model, etc
  - ◆ Build modular scenario with Fabric over reference architecture

        - Check how classes can be as modular as possible to be adapted for other Hyperledger projects

- ➔ Build Stimulus
  - ◆ Define interfaces and events needed to interact with Fabric components
    - e.g., REST APIs?
  - ◆ Code pluggable interface with programmable parameters
    - e.g., # transactions per second
  - ◆ Run tests with stimulus in different nodes
- ➔ Build Monitoring
  - ◆ Define monitoring interfaces for nodes and links
    - e.g., capture node cpu percent, link packet statistics
  - ◆ Set database interface to store monitored data (e.g., influxdb, elasticsearch)
  - ◆ Set visualization interface to show stored data (e.g., graphana)
- ➔ Build Dynamics
  - ◆ Define needed dynamicity in test execution
    - e.g., possible event types, schedule, duration (i.e., what, where, when)
  - ◆ Code basic set of events for topology events
    - e.g., node/link failures and resource modification
  - ◆ Code basic set of events for hyperledger components
    - e.g., transaction calls, parameter configuration
  - ◆ Test dynamics with Fabric scenario
- ➔ Build Analysis
  - ◆ Define possible analysis of results to generate test report
  - ◆ Define test report format
  - ◆ Code modules/classes to report test results
  - ◆ Test modules and output report
- ➔ Scale to cluster of servers
  - ◆ Define requirements to scale project
    - e.g., ssh keys, config files, docker images
  - ◆ Code needed modules/classes
  - ◆ Test all the architecture running with Fabric in scaled experimental scenarios
- ➔ Extend architecture to other Hyperledger project(s)
  - ◆ Define how architecture can be extended
    - e.g., needed components, interfaces, modules
  - ◆ Create the basic architecture to extend project
- ➔ Document the project
  - ◆ Iterate documentation of source code during project
  - ◆ Refine documentation by the end of project
  - ◆ Publish readthedocs documentation for the whole project
    - Containing multiple test cases exploring: basic architecture, monitoring, dynamics, analysis, extensibility

Activities Report 05/06 - Call Minutes

- Umbra motivation: run blockchain networks under tests, and observe behavior under certains test loads (e.g., transaction constant loads). Modify the network environment (e.g., delay, loss, bw). Capture packets and try to replay them, to test functionalities (eg., new consensus protocols). And have tests running for long time.
- MUST: Monitoring processes inside the container - check how docker create memory maps and if they can be monitored from the outside (host environment). Extract runtime metrics (cpu/memory/disk).
- MUST: Investigate by packet data (if available, not encrypted) the different phases of consensus algorithms.
- Mid-term prototype demo (end of July): pick one blockchain project and run in the containernet platform and have that fully reproducible! And later get real time statistics (metrics) and run some unit-tests on the system.
- Final-term prototype demo: average pdft (consensus) round works based on network statistics and analyzed (payload) traffic.
- Try Fabric running first! Iroha or Sawtooth as second option.
- Take over Umbra lab in github
- #umbra intro in the labs mailing list - labs@lists.hyperledger.org
- Write down cool ideas for issues/patches in the future

Activities Report 12/06

-