Operate and Extend Hyperledger Besu

Hyperledger Foundation workshop

March 21st, 2023

HYPERLEDGER
BLOCKCHAIN TECHNOLOGIES FOR BUSINESS

Welcome!

https://wiki.hyperledger.org/display/events/Operate+and+Extend+Hyperledger+Besu+Workshop

Operation session (8AM to 10AM Pacific)

- 30 minutes intro to Besu - what it does, how it works
- 30 minutes on Besu configuration - environment variables, toml file, cli flags, hidden flags. Docker vs binary installation.
- 60 minutes on running a Besu network
    - 15 minutes on Besu in dev mode, curl, easy first steps.
    - 15 minutes on Genesis block generation
    - 30 minutes on consortium creation, with monitoring and health report, with a Docker compose.

Developer session (10AM to Noon Pacific)

- 20 minutes on Besu and Github - source (3 repos), issues, PRs, code checkout, CI.
- 20 minutes on setting up with an IDE. Run and compile with Gradle. Checks and validations in place with spotless and errorprone.
- 20 minutes on the main repository content, describing types of tests in place, showing the tree of dependencies between Gradle modules.
- 30 minutes on adding an opcode to the EVM: show how to add a new opcode, add to the next hard fork revision, how to test it.
- 30 minutes on how to add a new JSON-RPC method, add tests and docs.

Prerequisites

https://wiki.hyperledger.org/display/events/Operate+and+extend+Hyperledger+Besu+Workshop+Prerequisites

Install:

- Java
- Git
- Docker
- Docker-Compose
- Your favorite IDE

Download:

- Besu sources
- Quorum Dev Quickstart

Hyperledger Besu development tasks

Spotless

```
./gradlew spotlessApply
```

Errorprone

```
./gradlew check
```

Hyperledger Besu layout

- A multi-module Gradle project
- The same project contains sources and the distribution logic
- Versions are centralized in one file

Hyperledger Besu - first run

In the terminal, type `./gradlew assemble`

Repository organization

The repository is made of modules.

Each module typically contains source code under src/main/java.

Additionally, unit tests are present under src/test/java.

Integration tests are more rare - present under src/integration-test/java

Two more set of tests specific to Besu:

- Acceptance tests, under the acceptance-test module.
- Reference tests, taken after the Ethereum tests.

Dive into the repository

Start with the besu and config modules as top-level modules.

See the ethereum module for most of the Ethereum-related logic

Note the other modules of interest:

- Enclave
- Crypto
- Datatypes
- Evm
- Nat, Metrics, etc.

See how consensus is defined separately as its own module.

Integration and dependency injection

- Common gripe with Besu - there is no dependency injection framework like Spring used.
- Lots of moving pieces as the software itself has complex requirements
- Using the builder pattern - helps with validation, but also creates indirection
- Scroll through BesuController and BesuControllerBuilder

Example of shared concerns with ForkIdManager

Used as part of the status message sent when connecting to other Ethereum clients

The fork ID is computed from the current known head block number

Client consensus set-up

Ethereum hard forks include changes to everything:

- EVM changes
- Transaction signing changes
- Reward calculations changes
- Block minting changes
- Transaction pool fee calculations
- More!

ProtocolShedule keeps track of all the configuration items as part of a specific range of block numbers for a chain.

EVM - Add a new opcode

See MainnetEVMs to understand how we register operations right now.

Open the hierarchy of the Operation interface.

Create our very own opcode with our implementation of a new operation.

The EVM!

A registry-based virtual machine

Uses a stack and a memory to manage the execution

Works in frames: a call can trigger a new execution of EVM code, passing its result back to the caller.

So how would we create our own opcode with these constructs?

JSON-RPC server

Take a tour of our JSON-RPC server setup: `JsonRpcMethodsFactory`

Exercise

We are going to create a new opcode that will ensure only clients configured with a shared secret can validate a block we care about.

The shared secret can be set via configuration or JSON-RPC.

Exercise - create our very own opcode

Create an opcode operation that checks the first word on the stack.

If the word equals a parameter passed in to the opcode operation, reply 1, else, reply 0.

Exercise - bind a new JSON-RPC method

Create a new JSON-RPC method that allows us to set a 32-byte word

Register the method under the eth namespace

Check it out with a curl command

Exercise - add a hidden flag to set the shared secret

Create a class to hold the value of our shared secret.

Inject the class everywhere!

Integrate our opcode as a new hard fork

Create our very own new hard fork with our opcode.

Add to ProtocolScheduleBuilder

Create a new hard fork - the workshop hard fork

Create a genesis file with the hard fork!

Deploy a contract calling out to the new opcode

See the contract code:

Deploy with curl:
https://github.com/atoulme/besu-workshop/blob/main/extend/README.md

Now call the contract!

Questions

Discussion