DU PARC LOCMARIA Armand
KOVACEVIC Veljko

BDE Project Fall 2021: Kubernetes + Geth + Miningcore + Ethminer

0. Motivation and Challenges

Aside from the speculative opportunity, cryptocurrencies, being decentralized and censorship resistant, seem to be a promising tool to fight dictatorship. Being both interested in cryptocurrencies we choose to work on deploying a private Ethereum blockchain and a mining pool.

The main challenge with this project was the lack of documentation. It is uncommon to deploy a private blockchain to kubernetes and it is also rare to deploy a mining pool. Therefore it was difficult to find documentation and tutorials, let alone good ones.

1. Deploying a Private Ethereum Blockchain on a Kubernetes Cluster

The Ethereum standard is implemented by different softwares. We chose to go with Geth as it seemed to be popular and well maintained. We then went on and followed their README to deploy a blockchain on one computer before moving on to deploying it on Kubernetes. To do so we followed this tutorial, the only decent one we could find. Even though it is well written there were still a few debug challenges. Some of the configuration files are badly formatted and therefore not working. Some of the syntax of bash scripts presented in the tutorial is wrong. We also had to bump up Geth's version from 1.8 to 1.10 and update the CLI flags accordingly. We also had to find how to interact with the blockchain to make sure it was working.

Here is how it works. We have three types of files, config files, deployment files and service files. Deployment files describe pods, service files describe the network configuration (how different pods communicate) and config files contain configuration information (such as the blockchain initial state).
We then deploy three types of nodes:
Bootstrap Nodes: these nodes are responsible for discovery, they help the other nodes find each other.
Miner Nodes: these nodes mine to validate transactions
Transaction Nodes: these are the endpoints we use to send transactions to the network.

There is also the **registrar** which keeps track of the bootnodes and their addresses. Other nodes can then query it to retrieve this list and connect to the rest of the network. This is based on this open source project and we're not sure this would be part of a public blockchain.

One tricky bit is that each Geth node needs to be initialized with the chain id and the genesis block (the first block of the blockchain). To carry out the initialization we use two init containers. One to fetch the bootnode list from the registrar and one to initialize Geth with the genesis block. To pass this genesis block to those containers we use a configmap which lets us store multiple config files in a kubernetes config file and mount those as ‘real’ files on the containers’ filesystems. The init containers are only run when creating the container.

Once everything is set up, the private blockchain is ready to use.

After successfully initializing the blockchain, we wanted to send some transactions. To send a transaction, we had to run `geth attach` in a transaction pod to access the “Geth Javascript console” which seems to be the only way to connect wallets and send transactions from a Geth node.

Before running the blockchain, we created an account and filled it with coins by setting its balance in the genesis block (genesis.json). Once the blockchain was running, we connected to the bash of the “member” pod, imported the secret key in “/root/.ethereum/keystore/” and used the “Geth Javascript console”.

But at this step we encountered a problem. When we wanted to unlock the account with the function called `unlockAccount()`, there was an error that seemed to indicate the function was not accessible and pointed to a Github repository containing the function.

2. Deploying a Mining Pool

On an Ethereum blockchain, nodes validate transactions by ‘mining blocks’ to participate in a consensus mechanism. To incentivize mining, financial rewards are given to miners. Unfortunately if you’re mining alone (on limited compute) it may take a long time (weeks or months) before receiving your first reward. But running hardware costs money and it may not be feasible to run hardware for long stretches of time with only the hope of a future reward. Mining Pools solve this problem. They appear as a single node on the Ethereum blockchain but aggregate the computing power of many miners behind the scenes. When the pool receives a reward it splits it up and sends it to the miners, in proportion to their mining effort.

There doesn’t seem to exist a lot of well maintained and well documented mining pool software. We think this is because there is no real use for it. A lot of mining pools develop their own software in house based on their objectives. For example, some pools run modified nodes to front-run transactions and capture even more benefits.
We came across this blog post which suggested that Miningcore is a good choice. And indeed out of the 3 other mentioned options this is the only one that is still maintained.

Unfortunately, we didn’t manage to get it to work but we did learn a lot by trying. The way it is supposed to work is the following. There is one geth node which is connected to the rest of the blockchain. Behind that node sits Miningcore. It fetches ‘jobs’ from the blockchain and then distributes work to the miners. The miners use software such as Ethminer to do computations and connect to Miningcore through the stratum protocol.

To try deploying Miningcore we first cloned our *miner.yaml* config to *pool.yml*. This gave us a pod with a working Geth node. We then modified the Geth config so that it would be in mining mode but not use any resources from the computer. We added a second container to the pod to run Miningcore and that is where we got stuck. There is no recent image of Miningcore on Dockerhub, the one we tried crashed or weirdly received SIGTERM signals.

We also tried building our own image but didn’t succeed. The official Dockerfile didn’t work, nor did unofficial ones.