Report on Deploying a Private Ethereum Network

Ghislain CHENG

1. Context

The challenge involves setting up a **private Ethereum network** with at **least three nodes**, utilizing **Docker containers**. You must ensure that all nodes are properly communicating, syncing, and secured with appropriate network configurations.

In addition to setting up the network, you will need to automate its deployment using **Infrastructure as Code (IaC) tools** like **Terraform**, Ansible or GitOps tools like **ArgoCD** or Flux. This setup should be **modular and scalable**, allowing for easy modifications and expansions. **Monitoring and logging** are also crucial components of this exercise. You'll be required to implement monitoring for the Ethereum nodes using tools such as Prometheus and Grafana, and set up **alerts for key metrics**, ensuring that the network's health and **performance are closely tracked**.

2. Defines goals and non-goals of the project

Before choosing the most suitable consensus algorithm for the private Ethereum network, it's essential to define **goals** (what the consensus mechanism must achieve) and **non-goals** (what is less essential or a lower priority). This approach ensures that the selection aligns with the specific needs of a private, permissioned blockchain setup.

Goals

- **Select a Relevant Consensus Mechanism**: Choose Proof of Authority (PoA) since we aim to deploy a private network. Proof of Stake (PoS) is becoming less favored by the Ethereum community, and Proof of Work (PoW) is largely being abandoned.
- **Define a Clear Architecture and Configuration**: Establish a well-defined architecture and configuration strategy for the private network.
- **Deploy in Kubernetes**: Utilize Kubernetes for deploying the private Ethereum network, as container orchestration simplifies management.
- **Implement Monitoring and Logging**: Set up a monitoring and logging stack to ensure visibility and enable alerts for incidents.
- **Ensure Modular and Scalable Source Code**: The source code must be designed to be modular and scalable to accommodate future modifications and expansions.

Non-Goals

- **Decentralized Validator System**: Since the network is private and permissioned, there's no need for a fully decentralized and open validator system as required in public blockchains. Validators will be pre-approved.
- **Complexity of PoS**: While PoS offers strong security in public chains, the complexities of slashing and staking rules are less relevant in a controlled, private network.

3. Architecture Overview

a. Choose the right Proof of Authority consensus algorithm

Generally, for private networks, we aim to build on top of Practical Byzantine Fault Tolerance (PBFT). This algorithm addresses the Byzantine Generals Problem, which describes the challenge decentralized parties face in reaching consensus without relying on a trusted central authority.

Several consensus protocols are based on this algorithm, with the most recent being QBFT, an improvement over IBFT. Both protocols use the gossip protocol to propagate consensus proposals more quickly. With QBFT, we can validate transactions through RPC execution, requiring the approval of a majority of validators. Also QBFT provides a better efficiency during the proposal block.

Search client/consensus client that supports multiple POA algorithms. Besu seems a great fit because it supports private/public networks. It seems more modular and scalable. With POA only Besu is required to deploy a private ethereum network.

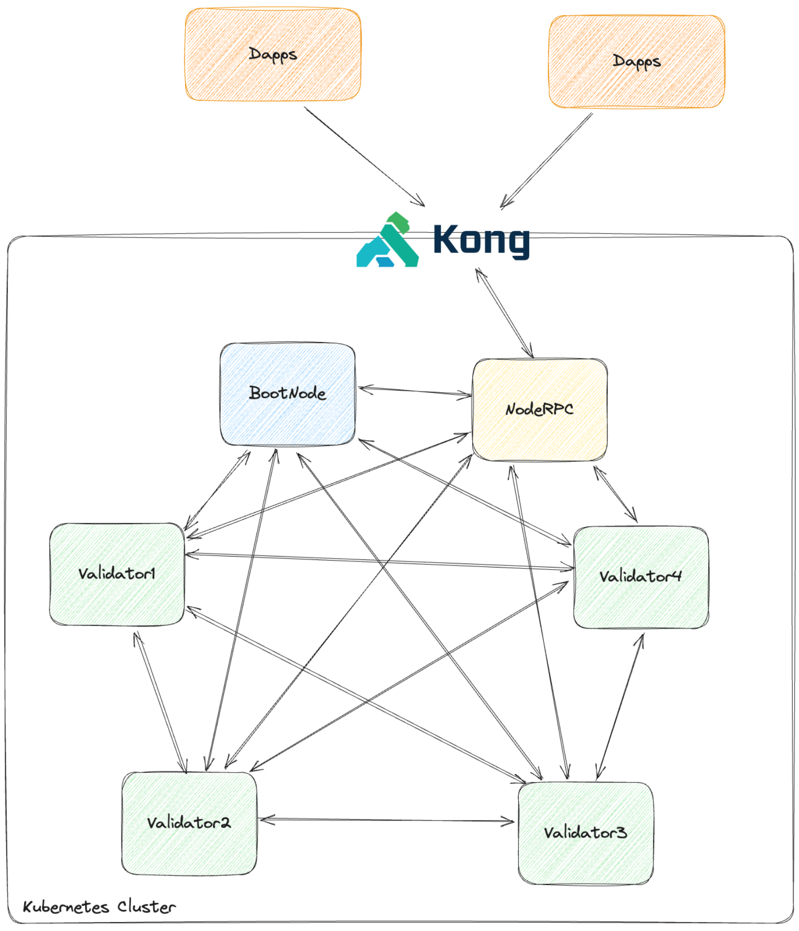
b. Architecture of the private ethereum network with Besu and QBFT algorithm

To deploy the private Ethereum network using the QBFT consensus algorithm, we must adhere to the rules of PBFT (Practical Byzantine Fault Tolerance). PBFT ensures both safety and liveness, allowing the system to reach consensus on the correct block ordering and to make progress even in the presence of Byzantine faults. By tolerating up to F faulty nodes in a network of N=3F+1 validators, PBFT provides resilience against malicious or faulty behavior.

We have decided to accommodate one faulty node, which requires deploying at least four validators to ensure availability in the worst-case scenario. To facilitate peer-to-peer communications, we will also use a bootnode, which helps in discovering other nodes.

Additionally, to enable communication between external DApps and this private Ethereum network, it is advisable to deploy one or more RPC nodes. Node RPC allows us to perform public transactions and execute read-heavy operations, such as those required for a blockchain indexer or explorer. However, with RPC nodes, we generally avoid exposing administrative methods, such as voting, admin, and debug Ethereum methods, to enhance security.

Furthermore, the Kong API Gateway is an excellent choice for implementing ACL (Access Control List) rules for DApps. For instance, we can create consumers within Kong and apply different rate limiting rules tailored to each DApp.



4. Deployment and automation Strategy

To facilitate deployment through Docker containers, we utilize an open-source container orchestration tool called Kubernetes. In our setup, we have chosen to use a **KinD** (Kubernetes in Docker) cluster to simulate Kubernetes locally. The controller manager and worker instances will be represented as Docker containers. From the perspective of a Site Reliability Engineer, we aim to avoid using Minikube or Kubernetes with Docker Desktop, as we want to simulate a setup that closely resembles real-world environments with other Kubernetes providers.

To automate the Kubernetes deployment, we will employ GitOps, where the Git repository serves as the source of truth. This approach provides significant flexibility for upgrading the infrastructure and allows for easy rollbacks in case of failures. In our case, **ArgoCD** is the best fit, as it offers visualization capabilities that can assist less technical team members.

5. Monitoring and logging strategy

In a private blockchain that functions similarly to a database, it is crucial to retrieve and monitor basic metrics such as CPU usage, memory usage, and disk usage. Disk usage is particularly important as it will continuously increase with the growth of the blockchain. It may be beneficial to implement a custom operator to extend the volume when it reaches a certain threshold; however, it's important to note that extending a Persistent Volume Claim (PVC) requires restarting the pods. Therefore, controlling which pods can be restarted is essential to maintain compliance with the PBFT requirements.

Additionally, the blockchain metrics to monitor will depend on your specific use case, but generally, key metrics include current block height, synchronization status, and block times. These metrics are vital for ensuring the proper functioning of your blockchain nodes.

In our setup, each Kubernetes component has its own exporter, which allows us to use Prometheus to pull metrics directly. To automate monitoring and establish rules, it is recommended to utilize the Prometheus Operator. For blockchain use cases, it is advisable to pull blockchain node metrics at intervals shorter than the network's block time to ensure that we capture all metrics for each block.

6. Challenges and Solutions

One of the primary challenges was ensuring that all nodes could effectively communicate and remain synchronized. Achieving proper synchronization required careful configuration of network parameters, such as node discovery protocols and peer-to-peer (P2P) connections. To address this issue, we deployed a bootnode to facilitate P2P discovery using dynamic discovery, while also utilizing a static-nodes.json file, which is continuously updated whenever a new node is deployed. This approach was implemented within a helm chart deployment, without the need to develop a custom Kubernetes operator.

To satisfy this approach, we have to handle the concurrent access to configmap in order to update safely.

The most challenging aspect of this project is handling the monitoring component. Some key metrics are difficult to set up and track. For example, obtaining an accurate number of transactions per second can be challenging if you aim for precise values. In fact, this value is not available directly. We have to make some approximations using besu_blockchain_chain_head_transaction_count (which returns the number of transactions in the current block). However, this value doesn't appear to be cumulative, making it difficult to calculate the rate or increase of these values. To obtain the TPS, we can perform some mathematical operations:

- **avg_over_time(besu_blockchain_chain_head_transaction_count[1m])** : gives the average number of transactions per block
- **rate(ethereum_block_height[1m])** represents the number of blocks produced per second.

By multiplying these two values, we can estimate the number of transactions per second.

7. Improvements

To improve node operations, we can introduce a cron snapshot to establish a disaster recovery plan in case any nodes fail.

We can also build our own operator to automate all node operations, which will help avoid using bash scripts inside containers, as those can be difficult to maintain in the future.

To enhance security, it's possible to use TLS for peer-to-peer communications between the nodes.

Additionally, if we deploy multiple Kubernetes clusters, we can rewrite or forward logs and metrics to a centralized monitoring platform and adapt the retention of the logs/metrics.

Finally, to improve alerting, instead of using Slack integration with Alertmanager, we can use OpsGenie for production. This enterprise tool allows us to set up a complex alerting workflow, which can be highly beneficial for SRE teams, especially if there are on-call rotations to manage.

8. Conclusion

We demonstrate that our approach of defining goals and non-goals allows us to eliminate certain technologies and focus on the Besu architecture. We successfully deployed our private Ethereum network with monitoring and logging management. In this project, we also demonstrate how to correctly configure alerting. At the end of this report, we suggest several improvements for our private Ethereum network.