

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**

-----oo-----



BÁO CÁO ĐÖ ÁN

ĐỀ TÀI:

**XÂY DỰNG VÀ TRIỂN KHAI MỘT PRIVATE BLOCKCHAIN DỰA
TRÊN HYPERLEDGER FABRIC VÀ VÀ TỰ XÂY DỰNG CƠ CHẾ
ĐỒNG THUẬN**

Giảng viên hướng dẫn: Trần Tuấn Dũng

Sinh viên thực hiện:

21522067 - Lê Huy Hiệp

21522756 - Nguyễn Thanh Tuấn

Lớp: Blockchain: Nền tảng, ứng dụng và bảo mật - NT547.P11.ANTT

TP. Hồ Chí Minh, ngày 21 tháng 12 năm 2024

LỜI CẢM ƠN

Lời đầu tiên, xin chân thành cảm ơn Ban Giám hiệu Trường Đại học Công nghệ thông tin,khoa Mạng máy tính và truyền thông và thầy Trần Tuấn Dũng đã tạo nền tảng, điều kiện giúp chúng tôi có hội tiếp cận và nghiên cứu chủ đề này.

Trong quá trình nghiên cứu về chủ đề này, có thể do hiểu biết còn nhiều hạn chế nên bài làm khó tránh khỏi những thiếu sót. Chúng tôi rất mong nhận được những lời góp ý chân thành của thầy để bài báo cáo ngày càng hoàn thiện hơn.

Trân trọng.

NHÂN XÉT CỦA GIẢNG VIÊN

Mục lục

LỜI CẢM ƠN.....	2
NHẬN XÉT CỦA GIẢNG VIÊN.....	2
Mục lục	3
I. Giới thiệu chung	4
1. Giới thiệu về đề tài	4
2. Phạm vi nghiên cứu.....	5
3. Nội dung nghiên cứu	5
4. Công cụ sử dụng.....	5
II. Cơ sở lý thuyết	5
1. Private Blockchain	5
2. Hyperledger Fabric	6
2.1. Giới thiệu	6
2.2. Các thành phần của mạng	6
3 Cơ chế đồng thuận RAFT	8
3.1. Giới thiệu	8
3.2. Nguyên tắc hoạt động của Raft.....	8
3.3. Các thành phần chính trong Raft	9

3.4. Quy trình hoạt động của RAFT	9
3.5. Ưu điểm của RAFT trong Hyperledger Fabric.....	11
3.6. So sánh RAFT và Kafka (cơ chế đồng thuận khác trong Fabric).....	11
III. Triển khai.....	12
1. Mô hình triển khai:.....	12
2. Các thành phần của mô hình	12
3. Quy trình giao dịch được tạo	14
4. Các bước để tạo 1 mạng private blockchain dựa trên Hyperledger Fabric	15
4.1 Giới thiệu các công cụ có sẵn của Hyperledger Fabriccc	15
1. configtxgen.....	15
2. configtxlator	15
3. cryptogen	15
4. discover	16
5. idemixgen	16
6. orderer	16
7. osnadmin	16
8. peer	16
9. fabric-ca-client.....	16
10. fabric-ca-server.....	17
4.2. Các bước xây dựng	17
4.3. Tương tác với chaincode	26
5. Chính sửa cơ chế đồng thuận	27
IV. Demo	28
1. Demo 1: Thực hiện transaction và kiểm tra trên ledger sau khi commit thành công	28
2. Demo 2: Thực hiện tắt 1 trong 3 orderer và xem kết quả thực thi đồng thời xem log của các node orderer	28
V. Tài liệu tham khảo	32

I. Giới thiệu chung

1. Giới thiệu về đề tài

Private Blockchain là một dạng công nghệ blockchain chuyên biệt, được sử dụng trong nội bộ một tổ chức hoặc đối tác cụ thể. Không giống như blockchain công khai, vốn mở cho mọi người và cung cấp tính minh bạch hoàn toàn, private blockchain hoạt động dựa trên cơ chế cấp phép, chỉ cho phép các thành viên được ủy quyền truy cập và xác thực giao dịch. Cấu trúc này mang lại tính bảo mật, quyền riêng tư và khả năng kiểm soát cao hơn, làm cho private blockchain trở nên đặc biệt hấp dẫn đối với các

doanh nghiệp muốn tối ưu hóa hoạt động và duy trì tính bảo mật. Hyperledger Fabric là một nền tảng mã nguồn mở dùng để xây dựng các giải pháp số cái phân tán, với kiến trúc mô-đun cung cấp mức độ cao về bảo mật, linh hoạt, khả năng chịu lỗi và khả năng mở rộng. Điều này cho phép các giải pháp phát triển trên Fabric có thể được tùy chỉnh để phù hợp với bất kỳ ngành công nghiệp nào. Trong đề tài này chúng tôi sẽ sử dụng Hyperledger Fabric để xây dựng mạng private blockchain và tự xây dựng cơ chế đồng thuận dựa trên Raft.

2. Phạm vi nghiên cứu

Trong đề tài này chúng tôi sẽ nghiên cứu việc xây dựng mạng private blockchain bằng Hyperledger Fabric dựa trên các công cụ nhị phân có sẵn.

3. Nội dung nghiên cứu

Cơ sở lý thuyết: private blockchain, hyperledger fabric, cơ chế đồng thuận Raft; Từng bước cách xây dựng một mạng private blockchain dựa trên công cụ nhị phân; chuyển giao tài sản; tự xây dựng cơ chế đồng thuận bằng cách điều chỉnh cơ chế đồng thuận Raft

4. Công cụ sử dụng

- Hệ điều hành: ubuntu-24.04.1-desktop-amd64
- git
- docker, docker-compose
- golang
- Visual studio code
- Python
- Nodejs, npm

II. Cơ sở lý thuyết

1. Private Blockchain

Private Blockchain (blockchain riêng tư) là một dạng blockchain mà quyền truy cập bị hạn chế và chỉ dành cho các thành viên được ủy quyền trong một tổ chức hoặc một mạng cụ thể. Không giống như blockchain công khai (public blockchain) như Bitcoin hoặc Ethereum, nơi bất kỳ ai cũng có thể tham gia, private blockchain tập trung vào việc kiểm soát quyền truy cập và bảo mật dữ liệu.

Đặc điểm chính của Private Blockchain:

1. **Quyền truy cập hạn chế:** Chỉ các thành viên được chỉ định mới có quyền tham gia và tương tác với mạng.
2. **Tính phi tập trung hạn chế:** Mặc dù vẫn giữ một số tính chất phi tập trung, quyền kiểm soát chủ yếu nằm trong tay một tổ chức hoặc một nhóm quản trị.
3. **Tốc độ và hiệu suất cao:** Vì số lượng nút và yêu cầu đồng thuận được kiểm soát, private blockchain thường hoạt động nhanh hơn so với public blockchain.
4. **Bảo mật cao:** Dữ liệu trong private blockchain được bảo vệ và không thể truy cập bởi người ngoài.

5. **Đồng thuận tùy chỉnh:** Các cơ chế đồng thuận (consensus mechanism) có thể được điều chỉnh theo nhu cầu cụ thể, ví dụ: PBFT (Practical Byzantine Fault Tolerance), Raft,...

Ứng dụng của Private Blockchain:

- **Doanh nghiệp:** Theo dõi chuỗi cung ứng, quản lý tài sản và kiểm toán nội bộ.
- **Tài chính:** Tạo các hệ thống thanh toán nội bộ giữa các ngân hàng.
- **Y tế:** Lưu trữ và chia sẻ hồ sơ sức khỏe bệnh nhân an toàn.

2. Hyperledger Fabric

2.1. Giới thiệu

Hyperledger Fabric là một nền tảng công nghệ sổ cái phân tán (**DLT**) cấp doanh nghiệp, mã nguồn mở, được thiết kế để sử dụng trong các bối cảnh doanh nghiệp, phát triển bởi Linux Foundation. Nền tảng này cung cấp một số khả năng nổi bật khác biệt so với các nền tảng blockchain hoặc sổ cái phân tán phổ biến khác. Hyperledger Fabric là một trong số các dự án thuộc hệ sinh thái Hyperledger, được thiết kế đặc biệt để hỗ trợ private blockchain và các ứng dụng mạng blockchain permissioned (yêu cầu cấp quyền). Hyperledger Fabric có tính mô đun khá cao nên nó cho phép các Doanh nghiệp dễ dàng plug and play để xây dựng một ứng dụng Private Blockchain phù hợp các yêu cầu nghiệp vụ của mình.

Điểm nổi bật của Hyperledger Fabric:

- **Permissioned Blockchain:** Chỉ các thành viên được xác thực mới có thể tham gia mạng.
- **Modular Architecture:** Kiến trúc mô-đun cho phép tùy chỉnh từng thành phần, chẳng hạn như cơ chế đồng thuận, quản lý thành viên, và dịch vụ hợp đồng thông minh (smart contract).
- **Pluggable Consensus:** Hỗ trợ nhiều cơ chế đồng thuận khác nhau, ví dụ: Raft, Kafka.
- **Hợp đồng thông minh (Chaincode):** Được triển khai bằng nhiều ngôn ngữ lập trình, phổ biến nhất là Go và JavaScript.
- **Bảo mật dữ liệu:** Hỗ trợ mã hóa và phân quyền chi tiết, đảm bảo rằng chỉ các bên liên quan có thể truy cập dữ liệu cụ thể.

2.2. Các thành phần của mạng

Hyperledger Fabric được tổ chức thành nhiều module hoạt động tương tác với nhau, tạo thành một hệ thống hoàn chỉnh và mạnh mẽ. Dưới đây là các module quan trọng:

Fabric CA (Fabric Certificate Authority):

Fabric CA là một thành phần quan trọng trong hệ sinh thái Hyperledger Fabric, chịu trách nhiệm **quản lý và cấp phát** chứng chỉ số (digital certificates) cho các thành viên trong mạng blockchain. Chứng chỉ này giúp xác thực danh tính của các thực thể tham gia mạng, bao gồm người dùng, ứng dụng, và các tổ chức. Fabric CA giúp tạo ra một

môi trường tin cậy cho việc xác thực và bảo mật trong quá trình giao dịch trên block-chain.

Membership Service Provider (MSP):

Membership Service Provider (MSP) của một tổ chức đóng vai trò quan trọng trong việc **xác định** cơ quan cấp chứng chỉ (Certificate Authority - CA) nào được ủy quyền phát hành danh tính hợp lệ cho các thành viên của tổ chức đó.

MSP không chỉ đơn thuần quản lý danh sách các thành viên tham gia vào mạng lưới hoặc channel. Thay vào đó, MSP còn có khả năng định nghĩa các vai trò cụ thể mà một cá nhân hoặc tác nhân có thể đảm nhiệm trong tổ chức mà MSP đại diện, chẳng hạn như vai trò quản trị viên hoặc thành viên của một nhóm trực thuộc tổ chức. MSP cũng là nền tảng để xác định các quyền truy cập trong mạng lưới hoặc channel, bao gồm quyền quản trị, quyền đọc dữ liệu, hoặc quyền ghi dữ liệu.

MSP còn cung cấp khả năng duy trì danh sách các danh tính đã bị thu hồi (Certificate Revocation List - CRL), đảm bảo rằng các danh tính không còn giá trị hoặc không đáng tin cậy sẽ không thể tiếp tục sử dụng trong mạng lưới.

Peers:

Một Blockchain network bao gồm chủ yếu các peer. Peer là thành tố cơ bản của network vì nó lưu trữ bản sao của Smart Contract (Chaincode) và bản sao của Ledger.

Có hai loại peer:

- Endorsing Peer dùng để xác minh và thực thi các giao dịch.
- Committing Peer để thực hiện cam kết giao dịch sau khi được đồng thuận.

Một tổ chức có thể chứa cùng lúc 2 loại peer trên. Các peer có vai trò quan trọng trong việc duy trì tính toàn vẹn và bảo mật của sổ cái. Các peer có thể được tạo, start, stop, tái cấu hình, thậm chí là xóa.

Orderer (Ordering Service):

Module chịu trách nhiệm sắp xếp giao dịch và đóng gói thành block để đảm bảo tất cả các peer cập nhật sổ cái một cách nhất quán trước khi đưa chúng vào sổ cái. Do Hyperledger Fabric được thiết kế dựa trên các thuật toán "deterministic consensus", bất kỳ block nào do Ordering Service tạo ra và được các peer xác thực đều được đảm bảo tính chính xác. Điều này giúp ledger duy trì trạng thái nhất quán tuyệt đối, loại bỏ hoàn toàn khả năng xảy ra tình trạng rẽ nhánh như trong các hệ thống blockchain khác. Hỗ trợ các cơ chế như Raft và Kafka để đảm bảo tính nhất quán của chuỗi.

Channel:

Channel là cơ chế giúp các peer và các thành phần khác trong mạng blockchain có thể tương tác với nhau. Có thể hiểu, channel là một nhóm các tổ chức có chung chuỗi giá trị, cùng tham gia vào một kênh giao tiếp. Một tổ chức có thể tham gia vào nhiều channel khác nhau, trong khi peer đóng vai trò là điểm kết nối giữa tổ chức và channel tương ứng. Channel tạo điều kiện để các ứng dụng cụ thể trong mạng blockchain giao tiếp một cách hiệu quả.

Ledger:

Một ledger lưu trữ trạng thái hiện tại của một doanh nghiệp dưới dạng nhật ký giao dịch. Mỗi giao dịch trong ledger có ảnh hưởng đến trạng thái cuối cùng của hệ thống, giúp theo dõi sự thay đổi của các đối tượng kinh doanh qua thời gian. Trong Hyperledger Fabric, ledger bao gồm hai phần chính: world state và blockchain. World state lưu trữ trạng thái hiện tại của các tài sản (dựa trên CouchDB hoặc LevelDB), trong khi blockchain ghi lại lịch sử các giao dịch đã thực hiện theo chuỗi thời gian.

Chaincode (Smart Contract):

Smart Contract: Là chương trình thực thi các quy tắc kinh doanh dưới dạng mã, cho phép tự động hóa các quy trình như chuyển nhượng tài sản hoặc thanh toán. Smart contract có thể được viết bằng các ngôn ngữ như JavaScript, Go hoặc Java. Smart contract thực hiện các thao tác như truy vấn (get), tạo (put) và xóa (delete) các đối tượng trong world state, đồng thời ghi lại các giao dịch không thay đổi trong blockchain.

Chaincode: Là bộ chứa của các smart contract được đóng gói lại, giúp nhóm các hợp đồng liên quan lại với nhau để triển khai. Chaincode có thể được quản lý bởi các quản trị viên và triển khai trên các peer nodes trong mạng.

Fabric SDK:

Client SDKs (Bộ công cụ phát triển phần mềm khách hàng) trong Hyperledger Fabric là các thư viện giúp các ứng dụng tương tác với mạng Hyperledger Fabric. Chúng cung cấp API để gửi và nhận dữ liệu, thực thi giao dịch, truy vấn dữ liệu từ sổ cái (ledger), quản lý các hợp đồng thông minh (smart contracts), và tương tác với các thành phần khác của mạng blockchain. Hỗ trợ nhiều ngôn ngữ như Java, Python, và JavaScript.

3 Cơ chế đồng thuận RAFT

3.1. Giới thiệu

Raft là một cơ chế đồng thuận được thiết kế để dễ hiểu và triển khai, đặc biệt trong các hệ thống phân tán. Nó tập trung vào việc đảm bảo tính nhất quán (consistency) của dữ liệu trên các nút trong hệ thống phân tán. Raft được sử dụng rộng rãi trong các ứng dụng doanh nghiệp, bao gồm cả trong blockchain, như một giải pháp thay thế đơn giản và hiệu quả cho Paxos.

Trong Hyperledger Fabric, cơ chế Raft được sử dụng như một dịch vụ sắp xếp (ordering service) để đảm bảo thứ tự của các giao dịch trước khi chúng được ghi vào sổ cái.

3.2. Nguyên tắc hoạt động của Raft

Raft hoạt động bằng cách phân phối dữ liệu qua một tập hợp các nút, trong đó mỗi nút có một bản sao của dữ liệu. Hệ thống sử dụng một nút làm leader (lãnh đạo) để điều phối các hoạt động và đồng bộ dữ liệu với các nút khác (follower).

Quá trình chính:

1. Lựa chọn leader:

Một nút trong mạng được chọn làm leader thông qua một cuộc bầu cử.

Leader chịu trách nhiệm điều phối các thay đổi và đồng bộ dữ liệu với các follower.

2. Ghi nhận log:

Các yêu cầu từ client được gửi đến leader.

Leader ghi các thay đổi vào nhật ký (log) của mình và sau đó gửi các bản cập nhật đến follower.

3. Đồng bộ dữ liệu:

Follower nhận và áp dụng các bản cập nhật từ leader để đảm bảo tất cả các nút trong mạng có cùng trạng thái.

4 . Cam kết giao dịch:

Một giao dịch được coi là cam kết khi nó đã được ghi nhận bởi đa số các nút trong mạng (quorum).

5. Xử lý lỗi:

Nếu leader gặp sự cố, các nút còn lại sẽ bầu chọn một leader mới để tiếp tục quá trình đồng thuận.

3.3. Các thành phần chính trong Raft

Một node order có thể là 1 trong 3 trạng thái sau

Leader: Điều phối hoạt động, nhận yêu cầu từ client, và gửi các bản ghi cập nhật đến follower. Chỉ có một leader tại một thời điểm trong hệ thống.

Follower: Lắng nghe và thực hiện các hướng dẫn từ leader.

Nếu follower không nhận được tín hiệu từ leader trong một khoảng thời gian, nó sẽ chuyển về trạng thái candidate.

Candidate: Khi một node trong hệ thống không còn nhận được tín hiệu từ leader (do leader bị lỗi hoặc không phản hồi), nó sẽ tự động chuyển sang trạng thái **candidate** và bắt đầu quá trình bầu cử để tìm một **leader** mới. Một candidate trở thành leader nếu nhận được phiếu bầu từ đa số các nút trong mạng.

3.4. Quy trình hoạt động của RAFT

Bước 1: Trạng thái ban đầu - Follower

- Mỗi node bắt đầu ở trạng thái **follower**. Khi ở trạng thái này, nó lắng nghe các tín hiệu từ **leader** thông qua các **heartbeat**.
- **Leader** sẽ gửi tín hiệu **heartbeat** định kỳ đến các **follower** để đảm bảo các **follower** luôn cập nhật và duy trì trạng thái đồng bộ.
- Nếu một node **follower** không nhận được tín hiệu **heartbeat** từ **leader** trong khoảng thời gian **election timeout**, nó sẽ chuyển sang trạng thái **candidate**.

Bước 2: Quá trình bầu cử

- Khi một node **follower** chuyển sang trạng thái **candidate** do không nhận được heartbeat từ **leader**, nó sẽ tự động **tăng term** (một giá trị số ngẫu nhiên).
- **Candidate** sẽ gửi một yêu cầu bầu cử (**RequestVote**) tới tất cả các node khác trong mạng. Yêu cầu này yêu cầu các node bỏ phiếu cho nó nếu chúng chưa bỏ phiếu cho ứng cử viên khác trong **term** hiện tại.
- Các node trong mạng (bao gồm cả **follower** và **candidate**) sẽ kiểm tra tính hợp lệ của **candidate** (có term cao hơn không, có log hợp lệ không) và nếu hợp lệ, sẽ bỏ phiếu cho **candidate**.

Bước 3: Trở thành Leader

- **Candidate** cần nhận được đa số phiếu bầu từ các node trong mạng để trở thành **leader**.
- Khi **candidate** nhận được đủ số phiếu bầu (tức là đa số node đồng ý trong một **term**), nó sẽ trở thành **leader** và bắt đầu gửi **heartbeat** tới các node còn lại để giữ cho mạng ổn định.
- Sau khi **leader** được bầu, các node còn lại sẽ chuyển về trạng thái **follower** và đồng bộ với **leader**.

Bước 4: Ghi và đồng bộ dữ liệu

- **Leader** sẽ nhận và xử lý các yêu cầu ghi (write) từ client. Sau đó, **leader** sẽ ghi dữ liệu vào log của mình và gửi yêu cầu sao chép (replicate) log này tới các **follower**.
- Các **follower** sẽ nhận và lưu trữ các log này. Sau khi tất cả các **follower** đã đồng bộ log, **leader** sẽ gửi tín hiệu xác nhận để các giao dịch được chính thức ghi vào sổ cái.

Bước 5: Quản lý lỗi và phục hồi

- Nếu **leader** gặp sự cố và không còn phản hồi (hoặc bị lỗi), các **follower** sẽ không nhận được **heartbeat** nữa và chuyển sang trạng thái **candidate**.
- Một quá trình bầu cử mới sẽ được bắt đầu để chọn ra **leader** mới. Quá trình này sẽ đảm bảo rằng một **leader** mới được bầu nhanh chóng, duy trì tính khả dụng của hệ thống.
- Nếu một **follower** bị lỗi hoặc không thể liên lạc với **leader**, nó sẽ không tham gia vào quá trình đồng bộ cho đến khi được phục hồi và đồng bộ lại với **leader**.

Bước 6: Quản lý log và sự đồng bộ

- **Raft** duy trì tính nhất quán của log trên tất cả các node thông qua việc sử dụng cơ chế **quorum** (đa số phiếu).
- Nếu có sự không đồng bộ giữa các node về log, **leader** sẽ yêu cầu các **follower** đồng bộ lại để đảm bảo tính nhất quán.

3.5. Ưu điểm của RAFT trong Hyperledger Fabric

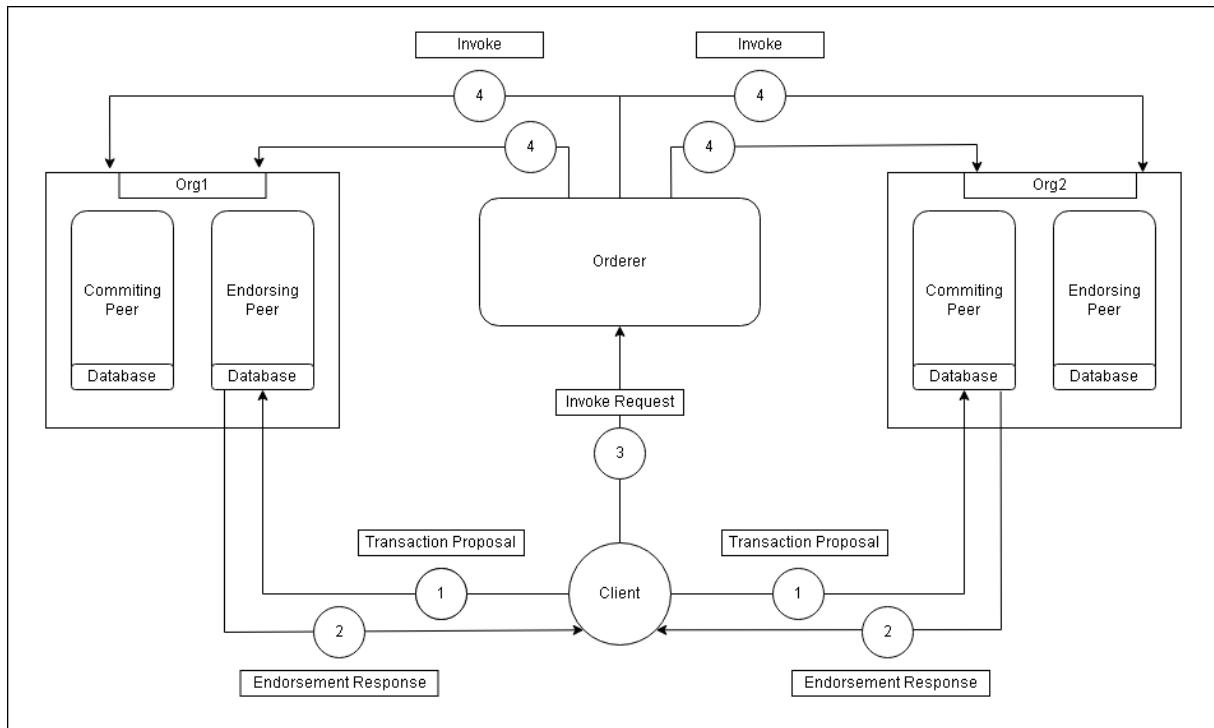
- **Đơn giản và dễ triển khai:** Raft được thiết kế để dễ hiểu và triển khai so với các cơ chế đồng thuận phức tạp khác như Paxos.
- **Hiệu suất cao:** Raft có thể xử lý hàng loạt các giao dịch nhanh chóng nhờ vào việc chỉ yêu cầu đồng thuận từ đa số nút.
- **Chịu lỗi (Fault Tolerance):** Hệ thống Raft có thể tiếp tục hoạt động ngay cả khi một số nút bị lỗi.
- **Tính nhất quán mạnh (Strong Consistency):** Đảm bảo rằng tất cả các nút trong hệ thống đều có cùng một trạng thái.
- **Tích hợp tốt với Hyperledger Fabric:** Raft trong Hyperledger Fabric được sử dụng để sắp xếp giao dịch, đảm bảo tính toàn vẹn và thứ tự giao dịch trong mạng blockchain.

3.6. So sánh RAFT và Kafka (cơ chế đồng thuận khác trong Fabric)

Đặc điểm	RAFT	Kafka
Phức tạp	Đơn giản, dễ triển khai	Phức tạp hơn, yêu cầu thêm Zookeeper
Tính chịu lỗi	Chịu lỗi cao, leader dễ thay thế	Chịu lỗi tốt nhưng phụ thuộc vào Zookeeper
Hiệu suất	Tốt trong hệ thống nhỏ và vừa	Phù hợp với khối lượng giao dịch lớn
Cấu hình	Ít cấu hình hơn	Yêu cầu cấu hình phức tạp hơn

III. Triển khai

1. Mô hình triển khai:



2. Các thành phần của mô hình

Client Application (Ứng dụng khách)

- **Vai trò:**
- Ứng dụng khách là điểm tương tác của người dùng với mạng blockchain. Nó thực hiện các yêu cầu giao dịch, chẳng hạn như thêm, sửa đổi, hoặc truy vấn dữ liệu.
- **Chức năng chính:**
 - Gửi yêu cầu giao dịch đến các Endorser Peers.
 - Nhận phản hồi từ mạng và xử lý thông tin kết quả giao dịch.

Peer Nodes (Nút mạng)

Các Peer Nodes là thành phần cốt lõi chịu trách nhiệm lưu trữ sổ cái (ledger), thực thi hợp đồng thông minh (chaincode), và duy trì trạng thái mạng. Có hai loại chính:

- **Endorser Peer:**
 - Thực hiện xác minh giao dịch bằng cách chạy chaincode.
 - Gửi phản hồi đề xuất (proposal response) đến Client Application.
- **Committer Peer:**
 - Ghi các giao dịch đã xác thực vào sổ cái (ledger).
 - Đảm bảo tính toàn vẹn và cập nhật trạng thái thế giới (world state).

Orderer Nodes (Nút đặt hàng)

- **Vai trò:**

Orderer Nodes chịu trách nhiệm sắp xếp và đồng thuận giao dịch. Đây là thành phần chính để đảm bảo tất cả các giao dịch trong mạng được thực thi theo thứ tự và nhất quán.

- **Chức năng chính:**

- Sử dụng cơ chế đồng thuận (RAFT, Kafka, hoặc BFT) để xử lý giao dịch.
- Đóng gói giao dịch thành các khối và phân phối đến các Peers.

Smart Contracts (Hợp đồng thông minh)

- **Vai trò:**

Smart Contracts, còn gọi là Chaincode, là logic nghiệp vụ được định nghĩa bởi người dùng.

- **Chức năng chính:**

- Xử lý logic nghiệp vụ trong giao dịch.
- Đảm bảo tính tự động hóa và bảo mật trong việc thực hiện các quy tắc trên mạng blockchain.

Ledger (Sổ cái)

- **Vai trò:**

Ledger là nơi lưu trữ tất cả các giao dịch trong mạng blockchain.

- **Cấu trúc chính:**

- **Blockchain Ledger:** Ghi lại tất cả các giao dịch dưới dạng bất biến.
- **World State:** Lưu trạng thái hiện tại của dữ liệu dưới dạng dễ dàng truy vấn.

Membership Service Provider (MSP)

- **Vai trò:**

MSP là thành phần quản lý danh tính và quyền hạn của các thành viên trong mạng.

- **Chức năng chính:**

- Cung cấp cơ chế xác thực và phân quyền.

Certificate Authority (CA)

- **Vai trò:**

CA chịu trách nhiệm cấp phát, quản lý, và thu hồi chứng chỉ số cho các thành viên trong mạng.

- **Chức năng chính:**

- Cấp chứng chỉ X.509 để định danh các thành phần (Peers, Orderer, Client).
- Đảm bảo tính an toàn và đáng tin cậy trong việc truyền dữ liệu.

Channel (Kênh giao tiếp)

- **Vai trò:**

Channels là cách tổ chức các giao tiếp riêng tư trong mạng Hyperledger Fabric.

- **Chức năng chính:**

- Phân chia mạng thành các không gian giao tiếp riêng biệt giữa các tổ chức.
- Mỗi channel có ledger riêng, đảm bảo tính riêng tư và bảo mật cho các giao dịch trong kênh.

Policy (Chính sách mạng)

- **Vai trò:**

Chính sách định nghĩa các quy tắc và điều kiện trong mạng blockchain.

- **Chức năng chính:**

- Quản lý chính sách xác nhận giao dịch (Endorsement Policy).
- Điều khiển quyền truy cập vào các thành phần và chức năng trong mạng.

3. Quy trình giao dịch được tạo

Quy trình này bao gồm các bước chính sau:

Bước 1: Khởi tạo giao dịch

- **Client Application** (ứng dụng khách) bắt đầu bằng cách gửi yêu cầu giao dịch đến các **Endorser Peers** (các nút xác thực). Yêu cầu giao dịch chứa thông tin cần thực hiện, chẳng hạn như tạo, sửa đổi hoặc xóa một bản ghi trên blockchain.
- Các giao dịch không được gửi trực tiếp đến blockchain; thay vào đó, chúng được đánh giá trước bởi các Endorser.

Bước 2: Endorsement (Xác nhận giao dịch)

- **Endorser Peers** nhận yêu cầu và thực hiện mô phỏng giao dịch thông qua **Smart Contract** (Chaincode).
- Các Endorser không ghi dữ liệu vào blockchain mà chỉ thực hiện kiểm tra logic và xác thực tính đúng đắn của giao dịch. Kết quả của bước này là một **Proposal Response** (phản hồi đề xuất), bao gồm chữ ký số của Endorser và kết quả thực thi.

Bước 3: Phân phối phản hồi

- Client Application thu thập phản hồi từ các Endorser.
- Nếu phản hồi từ đủ số lượng Endorser được yêu cầu (theo **Endorsement Policy** - Chính sách xác thực), giao dịch sẽ tiếp tục. Nếu không, giao dịch sẽ bị từ chối.

Bước 4: Submit Transaction (Nộp giao dịch)

- Client gửi giao dịch được xác nhận (transaction proposal) đến **Orderer** (cơ chế đặt hàng). Orderer chịu trách nhiệm đảm bảo tính đồng thuận của mạng và sắp xếp các giao dịch.

Bước 5: Ordering (Đặt hàng và đồng thuận)

- **Orderer Nodes** sử dụng cơ chế đồng thuận (như RAFT) để sắp xếp giao dịch theo thứ tự thời gian.
- Sau khi đồng thuận, các giao dịch được đóng gói thành một khối (block) và phân phối đến tất cả các **Committer Peers**.

Bước 6: Commit Transaction (Ghi giao dịch vào blockchain)

- **Committer Peers** nhận khối từ Orderer và thực hiện hai nhiệm vụ:
 - **Validation:** Kiểm tra tính hợp lệ của giao dịch dựa trên chính sách xác thực và phiên bản dữ liệu hiện tại (MVCC - Multi-Version Concurrency Control).

- **Commit:** Ghi các giao dịch hợp lệ vào sổ cái và cập nhật trạng thái thế giới (World State).

Bước 7: Hoàn tất giao dịch

- Sau khi giao dịch được ghi nhận, Committer Peers gửi thông báo trạng thái giao dịch (success/failure) về ứng dụng khách.
- Ứng dụng khách có thể truy vấn sổ cái để kiểm tra giao dịch đã được xác nhận **thành công hay chưa.**

4. Các bước để tạo 1 mạng private blockchain dựa trên Hyperledger Fabric

Code: https://github.com/huyhieple/private_blockchain.git

4.1 Giới thiệu các công cụ có sẵn của Hyperledger Fabric



1. configtxgen

Công cụ này được sử dụng để tạo các tập tin cấu hình cho việc tạo kênh và cấu hình ordering service.

Chức năng:

- Tạo các tệp cấu hình channel.tx để khởi tạo kênh mới trong mạng.
- Tạo genesis block cho ordering service.
- Cung cấp các lệnh để cấu hình và quản lý kênh, ví dụ như tạo kênh hoặc thay đổi cấu hình của kênh.

2. configtxlator

Công cụ này dùng để dịch các tệp cấu hình giữa các định dạng khác nhau trong mạng Hyperledger Fabric.

Chức năng:

- Dùng để chuyển đổi giữa các định dạng JSON và protobuf của cấu hình kênh hoặc ordering service.
- Giúp người dùng dễ dàng kiểm tra và sửa đổi các cấu hình trong khi duy trì tính nhất quán.

3. cryptogen

Công cụ này dùng để tạo các chứng chỉ và khóa cho các tổ chức, peer, và orderer trong mạng.

Chức năng:

- Sinh các chứng chỉ (certificates) và khóa (keys) cho các tổ chức tham gia vào mạng blockchain.
- Được sử dụng trong các bước khởi tạo mạng Hyperledger Fabric để đảm bảo an toàn và xác thực các thành viên trong mạng.

4. discover

Công cụ này được sử dụng để phát hiện và khám phá các peer nodes trong mạng.

- Chức năng:
- Dùng để tìm kiếm các peer nodes và dịch vụ trên mạng blockchain.
- Giúp các node nhận diện và đồng bộ hóa với nhau trong mạng phân tán.

5. idemixgen

Công cụ này dùng để tạo các chứng chỉ và khóa cho hệ thống Identity Mixer (Idemix) trong Hyperledger Fabric.

Chức năng:

- Sinh các chứng chỉ và khóa cho hệ thống Idemix, giúp quản lý danh tính và quyền riêng tư cho các giao dịch trong blockchain.

6. orderer

Đây là công cụ điều phối (ordering service) trong mạng Hyperledger Fabric.

Chức năng:

- Quản lý việc tạo và phân phối các block chứa giao dịch đến các peer nodes.
- Đảm bảo rằng các giao dịch được thực hiện theo đúng thứ tự và duy trì tính nhất quán của mạng.
- Điều phối các giao dịch giữa các peer và lưu trữ các block giao dịch.

7. osnadmin

Công cụ này là dành cho việc quản lý và điều hành ordering service trong Hyperledger Fabric.

Chức năng:

- Quản lý cấu hình và trạng thái của ordering service (OSN).
- Cung cấp các lệnh để điều hành và quản lý các cấu hình, kiểm tra tình trạng của ordering service.

8. peer

Đây là công cụ chủ yếu để tương tác với các peer nodes trong mạng Hyperledger Fabric.

Chức năng:

- Quản lý các giao dịch, truy vấn dữ liệu và thực thi các smart contract (chaincode).
- Tạo, truy vấn và xử lý các kênh, chaincode.
- Tương tác với các chaincode để thực hiện các yêu cầu đọc và ghi dữ liệu vào blockchain.

9. fabric-ca-client

Đây là công cụ dùng để tương tác với Fabric CA (Certificate Authority), giúp cấp phát và quản lý chứng chỉ cho các node và thành viên trong mạng.

Chức năng:

- Đăng ký và cấp chứng chỉ cho các peer, client, và tổ chức tham gia vào mạng.
- Quản lý chứng chỉ và khóa cần thiết để xác thực các giao dịch và thành viên trong mạng.
- Cho phép các thành viên đăng nhập và nhận chứng chỉ từ CA.

10. fabric-ca-server

Đây là công cụ chạy Fabric Certificate Authority (CA), chịu trách nhiệm cấp phát chứng chỉ cho mạng Hyperledger Fabric.

Chức năng:

- Cung cấp dịch vụ CA cho mạng blockchain để cấp chứng chỉ cho các peer, orderer và client.
- Quản lý việc phát hành và thu hồi chứng chỉ cho các thành viên trong mạng blockchain.
- Đảm bảo tính an toàn và xác thực các giao dịch và thành viên tham gia mạng.

4.2. Các bước xây dựng

Bước 1: Tạo network artifact (các thành phần của mạng)

Dùng 1 file cấu hình có tên là crypto-config.yaml và công cụ **cryptogen** để tạo chứng chỉ cho mạng.

Tạo chứng chỉ cho orderer

```
OrdererOrgs:
# -----
# Orderer
# -----
- Name: Orderer
  Domain: example.com
  EnableNodeOUs: true

# -----
# "Specs" - See PeerOrgs below for complete description
# -----
Specs:
- Hostname: orderer
  SANS:
    - "localhost"
    - "127.0.0.1"
- Hostname: orderer2
  SANS:
    - "localhost"
    - "127.0.0.1"
- Hostname: orderer3
  SANS:
    - "localhost"
    - "127.0.0.1"

# -----
```

Tạo chứng chỉ cho tổ chức (Org)

```
PeerOrgs:
# -----
- Name: Org1
  Domain: org1.example.com
  EnableNodeOUs: true

  Template:
    Count: 2
    # Start: 5
    # Hostname: {{.Prefix}}{{.Index}} # default
    SANS:
      - "localhost"

  Users:
    Count: 1

- Name: Org2
  Domain: org2.example.com
  EnableNodeOUs: true

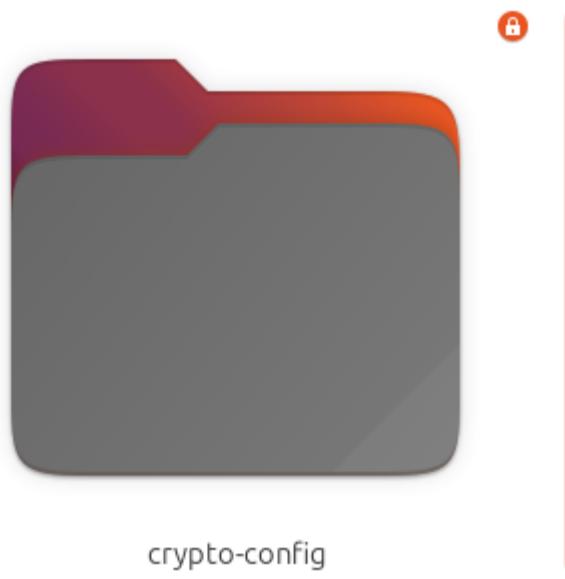
  Template:
    Count: 2
    # Start: 5
    # Hostname: {{.Prefix}}{{.Index}} # default
    SANS:
      - "localhost"

  Users:
    Count: 1
```

Sử dụng câu lệnh sau để tạo

cryptogen generate --config=./crypto-config.yaml --output=./crypto-config/

Kết quả tạo ra thư mục crypto-config chứa các chứng chỉ



```

root@hh1-VMware:/home/hhl/Desktop/hhl/basicNetwork_2.0/BasicNetwork-2.0/artifacts/channel/crypto-config# tree -d
.
└── ordererOrganizations
    └── example.com
        ├── ca
        ├── msp
        │   ├── admincerts
        │   ├── cacerts
        │   └── tlscacerts
        └── orderers
            ├── orderer2.example.com
            │   ├── msp
            │   │   ├── admincerts
            │   │   ├── cacerts
            │   │   ├── keystore
            │   │   ├── signcerts
            │   │   └── tlscacerts
            │   └── tls
            ├── orderer3.example.com
            │   ├── msp
            │   │   ├── admincerts
            │   │   ├── cacerts
            │   │   ├── keystore
            │   │   ├── signcerts
            │   │   └── tlscacerts
            │   └── tls
            └── orderer.example.com
                ├── msp
                │   ├── admincerts
                │   ├── cacerts
                │   ├── keystore
                │   ├── signcerts
                │   └── tlscacerts
                └── tls
        └── tlscacerts
        └── users

```

Bước 2:

Tiếp theo dùng 1 file cấu hình có tên là cryptotx.yaml và công cụ configtxgen để tạo genesis block và cấu hình mạng ban đầu.

```

#
# System channel
SYS_CHANNEL="sys-channel"

# channel name defaults to "mychannel"
CHANNEL_NAME="mychannel"

echo $CHANNEL_NAME

# Generate System Genesis block
configtxgen -profile OrdererGenesis -configPath . -channelID $SYS_CHANNEL -outputBlock ./genesis.block

# Generate channel configuration block
configtxgen -profile BasicChannel -configPath . -outputCreateChannelTx ./mychannel.tx -channelID $CHANNEL_NAME

echo "##### Generating anchor peer update for Org1MSP ######"
configtxgen -profile BasicChannel -configPath . -outputAnchorPeersUpdate ./Org1MSPanchor.tx -channelID $CHANNEL_NAME -asOrg Org1MSP

echo "##### Generating anchor peer update for Org2MSP ######"
configtxgen -profile BasicChannel -configPath . -outputAnchorPeersUpdate ./Org2MSPanchor.tx -channelID $CHANNEL_NAME -asOrg Org2MSP

```

Kết quả



Tạo ra genesis block chứa cấu hình ban đầu của các orderer trong đó có chứa cơ chế đồng thuận sẽ được sử dụng là RAFT

```

Profiles:
  BasicChannel: ...

  OrdererGenesis:
    <<: *ChannelDefaults
    Capabilities:
      <<: *ChannelCapabilities
  Orderer:
    <<: *OrdererDefaults
    OrdererType: etcdraft
    EtcdRaft:
      Consenters:
        - Host: orderer.example.com
          Port: 7050
          ClientTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.crt
          ServerTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.crt
        - Host: orderer2.example.com
          Port: 8050
          ClientTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer2.example.com/tls/server.crt
          ServerTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer2.example.com/tls/server.crt
        - Host: orderer3.example.com
          Port: 9050
          ClientTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer3.example.com/tls/server.crt
          ServerTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer3.example.com/tls/server.crt

```

Xem khối genesis.block

```

root@hh1-VMware:/home/hhl/Desktop/hhl/basicNetwork_2.0/BasicNetwork-2.0/artifacts/channel# configtxlator proto_decode --input genesis.block --type common.Block --output genesis.json
root@hh1-VMware:/home/hhl/Desktop/hhl/basicNetwork_2.0/BasicNetwork-2.0/artifacts/channel# jq . genesis.json
{
  "data": [
    "data": [
      {
        "payload": {
          "data": {
            "config": {
              "channel_group": {
                "groups": {
                  "Consortiums": {
                    "groups": {
                      "SampleConsortium": {
                        "groups": {
                          "Org1MSP": {
                            "groups": {},
                            "mod_policy": "Admins",
                            "policies": {
                              "Admins": {
                                "mod_policy": "Admins",
                                "policy": {
                                  "type": 1,
                                  "value": {
                                    "identities": [
                                      {
                                        "principal": {
                                          "msp_identifier": "Org1MSP",
                                          "role": "ADMIN"
                                        },
                                        "principal_classification": "ROLE"
                                      }
                                    ],
                                    "rule": {
                                      "n_out_of": 1
                                    }
                                  }
                                }
                              }
                            }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    ]
  }
}

```

Khối mychannel.tx chứa cấu hình của kênh bao gồm tên kênh là “mychannel” và các thành phần tham gia kênh này

Bước 3:

docker-compose.yaml là một file cấu hình Docker Compose. Sử dụng nó để triển khai các thành phần mạng Hyperledger Fabric trong môi trường Docker. File này định nghĩa các container và cấu hình cần thiết cho các thành phần khác nhau của mạng blockchain Hyperledger Fabric.

```

BasicNetwork-2.0 > artifacts > docker-compose.yaml
 1  version: "2"
 2
 3  networks:
 4    test:
 5
 6  services:
 7    ca-org1:
 8    ca-org2:
 9    orderer.example.com:
10    orderer2.example.com:
11    orderer3.example.com:
12    couchdb0:
13    couchdb1:
14    couchdb2:
15    couchdb3:
16    peer0.org1.example.com:
17    peer1.org1.example.com:
18    peer0.org2.example.com:
19    peer1.org2.example.com:
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359

```

Cấu hình 1 mạng docker có tên là test giúp các docker kết nối với nhau và làm việc trong một môi trường

Phạm vi: Chỉ các container thuộc cùng mạng test mới có thể giao tiếp trực tiếp qua tên container hoặc địa chỉ IP trong mạng đó.

Chạy lệnh để khởi tạo và chạy các container

Docker-compose up -d

```

root@hh1-VirtualBox:/home/hhl/Desktop/hhl/basicNetwork_2.0/BasicNetwork-2.0/artifacts# docker-compose up -d
cbdbe7a5bc2a: Pull complete
c1275de45a84: Pull complete
f588f9bba3d6: Pull complete
8f1a3c5e0a14: Pull complete
080b660b2196: Pull complete
1c1cf954d27a: Pull complete
Digest: sha256:163b245e18c5361520e10260093eb9271a091bf71ace8a55e0190658575e3037
Status: Downloaded newer image for hyperledger/fabric-orderer:2.1
Pulling couchdb0 (hyperledger/fabric-couchdb:...):
latest: Pulling from hyperledger/fabric-couchdb
8f91359f1fff: Pull complete
25382a7438ae: Pull complete
043089fd442c: Pull complete
51901bb19330: Pull complete
7e45b1a430cf: Pull complete
c0c197a7fd22: Pull complete
beb39dc557f4: Pull complete
1ada71a639d6: Pull complete
952d5b6650fc: Pull complete
37552aed0e1: Pull complete
45df897db071: Pull complete
Digest: sha256:626cc21b7d614f0dbc9e3577c4e6a7bbb72eae5e14cf75d1fe516f1bb2684dc
Status: Downloaded newer image for hyperledger/fabric-couchdb:latest
Pulling peer0.org1.example.com (hyperledger/fabric-peer:2.1)...
2.1: Pulling from hyperledger/fabric-peer
cbdbe7a5bc2a: Already exists
c1275de45a84: Already exists
533397ecbccb: Pull complete
366c0e7ddacd: Pull complete
66658cc1667a: Pull complete

```

```

root@hh1-VMware:/home/hh1/Desktop/hh1/basicNetwork_2.0/BasicNetwork-2.0/artifacts#
8f91359f1fff: Pull complete
25382a7438ae: Pull complete
043089fd442c: Pull complete
51901bb19330: Pull complete
7e45b1a430cf: Pull complete
c0c197a7fd22: Pull complete
beb39dc557fa: Pull complete
1ada71a639d6: Pull complete
952d5b6650fc: Pull complete
37552ae4d0e1: Pull complete
45df897d071: Pull complete
Digest: sha256:626cc21b7d614f0dbc9e3577c4e6a7bbb72eaee5e14cf75d1fe516f1bb2684dc
Status: Downloaded newer image for hyperledger/fabric-couchdb:latest
Pulling peer0.org1.example.com (hyperledger/fabric-peer:2.1)...
2.1: Pulling from hyperledger/fabric-peer
cbdbe7a5bc2a: Already exists
c1275de45a84: Already exists
533397ecbccb: Pull complete
366c0e7ddacd: Pull complete
66658cc1667a: Pull complete
Digest: sha256:86f4c15607d33bff44f965631a966bc142ebe8e7c18a25a41bea425cefa6dfd7
Status: Downloaded newer image for hyperledger/fabric-peer:2.1
Creating orderer.example.com ... done
Creating peer1.org1.example.com ... done
Creating couchdb2 ... done
Creating ca.org1.example.com ... done
Creating couchdb1 ... done
Creating peer0.org2.example.com ... done
Creating couchdb3 ... done
Creating couchdb0 ... done
Creating ca.org2.example.com ... done
Creating peer1.org2.example.com ... done
Creating orderer3.example.com ... done
Creating orderer2.example.com ... done
Creating peer0.org1.example.com ... done
root@hh1-VMware:/home/hh1/Desktop/hh1/basicNetwork_2.0/BasicNetwork-2.0/artifacts# 

```

Kiểm tra lại các container đang chạy

```

root@hh1-VMware:/home/hh1/Desktop/hh1/basicNetwork_2.0/BasicNetwork-2.0/artifacts# docker ps -a
CONTAINER ID IMAGE NAMES COMMAND CREATED STATUS PORTS
f061648fc5d5 hyperledger/fabric-peer:2.1 "peer node start" 8 minutes ago Up 8 minutes 0.0.0.0:7051->7051/tcp, :::7051->7051/tcp
54be14a6628e hyperledger/fabric-orderer:2.1 "orderer" 8 minutes ago Up 8 minutes 7050/tcp, 0.0.0.0:8050->8050/tcp, :::8050->8050/tcp, 0.0.0.0:8444->844
3/tcp, [::]:8444->8443/tcp orderer2.example.com
b5d9993e328a hyperledger/fabric-peer:2.1 "peer node start" 8 minutes ago Up 8 minutes 7051/tcp, 0.0.0.0:10051->10051/tcp, :::10051->10051/tcp
bf0d9df5f39ec hyperledger/fabric-ca "sh -c 'fabric-ca-se..." 8 minutes ago Up 8 minutes 0.0.0.0:8054->7054/tcp, [::]:8054->7054/tcp
93dce52024fa hyperledger/fabric-orderer:2.1 "orderer" 8 minutes ago Up 8 minutes 7050/tcp, 0.0.0.0:9050->9050/tcp, :::9050->9050/tcp, 0.0.0.0:8445->844
3/tcp, [::]:8445->8443/tcp orderer3.example.com
4c060b8501be hyperledger/fabric-couchdb "tini -- /docke... 8 minutes ago Up 8 minutes 4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp, :::5984->5984/tcp
couchdb0
ee41cc4b8800 hyperledger/fabric-couchdb "tini -- /docke... 8 minutes ago Up 8 minutes 4369/tcp, 9100/tcp, 0.0.0.0:8984->5984/tcp, [::]:8984->5984/tcp
couchdb3
6e7fef0c9a24 hyperledger/fabric-couchdb "tini -- /docke... 8 minutes ago Up 8 minutes 4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp, [::]:6984->5984/tcp
couchdb1
8125791dc71d hyperledger/fabric-peer:2.1 "peer node start" 8 minutes ago Up 8 minutes 7051/tcp, 0.0.0.0:9051->9051/tcp, :::9051->9051/tcp
ca.org1.example.com
d7536dacf484 hyperledger/fabric-ca "sh -c 'fabric-ca-se..." 8 minutes ago Up 8 minutes 0.0.0.0:7054->7054/tcp, :::7054->7054/tcp
01fbcd6ad2e18 hyperledger/fabric-couchdb "tini -- /docke... 8 minutes ago Up 8 minutes 4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp, [::]:7984->5984/tcp
couchdb2
db256ec3c19 hyperledger/fabric-peer:2.1 "peer node start" 8 minutes ago Up 8 minutes 7051/tcp, 0.0.0.0:8051->8051/tcp, :::8051->8051/tcp
peer0.org1.example.com
918da17de2db hyperledger/fabric-orderer:2.1 "orderer" 8 minutes ago Up 8 minutes 0.0.0.0:7050->7050/tcp, :::7050->7050/tcp, 0.0.0.0:8443->8443/tcp, :::
8443->8443/tcp orderer.example.com
af5b49e6de22 hello-world "/hello" 2 days ago Exited (0) 2 days ago
suspicuous_dhawan
root@hh1-VMware:/home/hh1/Desktop/hh1/basicNetwork_2.0/BasicNetwork-2.0/artifacts# 

```

Bước 4: Khởi tạo và tham gia vào kênh

Công cụ **peer** sẽ được sử dụng để khởi tạo kênh

Sử dụng

```

setGlobalsForPeer0Org1(){
    export CORE_PEER_LOCALMSPID="Org1MSP"
    export CORE_PEER_TLS_ROOTCERT_FILE=$PEER0_ORG1_CA
    export CORE_PEER_MSPCONFIGPATH=${PWD}/artifacts/channel/crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
    export CORE_PEER_ADDRESS=localhost:7051
}

```

Để thiết lập các biến cần thiết cho việc khởi tạo kênh bằng Peer0 Org1

Tên kênh

```
export CHANNEL_NAME=mychannel
```

Đây là hàm khởi tạo kênh

```

createChannel(){
    rm -rf ./channel-artifacts/*
    setGlobalsForPeer0Org1

    peer channel create -o localhost:7050 -c $CHANNEL_NAME \
        --ordererTLSHostnameOverride orderer.example.com \
        -f ./artifacts/channel/${CHANNEL_NAME}.tx --outputBlock ./channel-artifacts/${CHANNEL_NAME}.block \
        --tls $CORE_PEER_TLS_ENABLED --cafile $ORDERER_CA
}

```

Đây là hàm để các org tham gia kênh

```

joinChannel(){
    setGlobalsForPeer0Org1
    peer channel join -b ./channel-artifacts/${CHANNEL_NAME}.block

    setGlobalsForPeer1Org1
    peer channel join -b ./channel-artifacts/${CHANNEL_NAME}.block

    setGlobalsForPeer0Org2
    peer channel join -b ./channel-artifacts/${CHANNEL_NAME}.block

    setGlobalsForPeer1Org2
    peer channel join -b ./channel-artifacts/${CHANNEL_NAME}.block

}

```

Kết quả khi tạo kênh

```

root@hhl-VMware:/home/hhl/Desktop/hhl/basicNetwork_2.0/BasicNetwork-2.0# ./createChannel.sh
2024-12-09 20:42:14.503 +07 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2024-12-09 20:42:14.523 +07 0002 INFO [cli.common] readBlock -> Expect block, but got status: &{NOT_FOUND}
2024-12-09 20:42:14.526 +07 0003 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2024-12-09 20:42:14.729 +07 0004 INFO [cli.common] readBlock -> Expect block, but got status: &{SERVICE_UNAVAILABLE}
2024-12-09 20:42:14.731 +07 0005 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2024-12-09 20:42:14.934 +07 0006 INFO [cli.common] readBlock -> Expect block, but got status: &{SERVICE_UNAVAILABLE}
2024-12-09 20:42:14.937 +07 0007 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2024-12-09 20:42:15.139 +07 0008 INFO [cli.common] readBlock -> Expect block, but got status: &{SERVICE_UNAVAILABLE}
2024-12-09 20:42:15.144 +07 0009 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2024-12-09 20:42:15.346 +07 000a INFO [cli.common] readBlock -> Expect block, but got status: &{SERVICE_UNAVAILABLE}
2024-12-09 20:42:15.349 +07 000b INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2024-12-09 20:42:15.552 +07 000c INFO [cli.common] readBlock -> Received block: 0

```

Kết quả sau khi tham gia vào kênh:

```

root@hhl-VMware:/home/hhl/Desktop/hhl/basicNetwork_2.0/BasicNetwork-2.0# ./createChannel.sh
2024-12-09 20:54:51.451 +07 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2024-12-09 20:54:51.535 +07 0002 INFO [channelCmd] executeJoin -> Successfully submitted proposal to join channel
2024-12-09 20:54:51.593 +07 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2024-12-09 20:54:51.667 +07 0002 INFO [channelCmd] executeJoin -> Successfully submitted proposal to join channel
2024-12-09 20:54:51.718 +07 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2024-12-09 20:54:51.799 +07 0002 INFO [channelCmd] executeJoin -> Successfully submitted proposal to join channel
2024-12-09 20:54:51.853 +07 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connections initialized
2024-12-09 20:54:51.923 +07 0002 INFO [channelCmd] executeJoin -> Successfully submitted proposal to join channel
root@hhl-VMware:/home/hhl/Desktop/hhl/basicNetwork_2.0/BasicNetwork-2.0#

```

Vào trong các container để xem

docker exec -it peer0.org1.example.com sh

Để vào shell của container

Sau đó dùng lệnh *peer channel list* để xem

```

root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# docker exec -it peer0.org1.example.com sh
/opt/gopath/src/github.com/hyperledger/fabric/peer # peer channel list
2024-12-10 07:16:27.741 UTC [main] InitCmd -> WARN 001 CORE_LOGGING_LEVEL is no longer supported, please use the FABRIC_LOGGING_SPEC environment variable
2024-12-10 07:16:27.743 UTC [main] SetOrdererEnv -> WARN 002 CORE_LOGGING_LEVEL is no longer supported, please use the FABRIC_LOGGING_SPEC environment variable
2024-12-10 07:16:27.747 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
Channels peers has joined:
mychannel

```

Peer0 org1 đã tham gia vào kênh mychannel

Bước 5: Thiết lập chaincode trong các tổ chức

Đây là một bước quan trọng vì khi các tổ chức cài đặt được chaincode thì mới có thể tiến hành giao dịch và ghi vào sổ cái

Cài đặt chain code vào peer của các tổ chức bao gồm các bước sau:

Package (đóng gói):

```

packageChaincode() {
    rm -rf ${CC_NAME}.tar.gz
    setGlobalsForPeer0Org1
    peer lifecycle chaincode package ${CC_NAME}.tar.gz \
        --path ${CC_SRC_PATH} --lang ${CC_RUNTIME_LANGUAGE} \
        --label ${CC_NAME}_${VERSION}
    echo "===== Chaincode is packaged on peer0.org1 ====="
}

```

Install (cài đặt):

```

installChaincode() {
    setGlobalsForPeer0Org1
    peer lifecycle chaincode install ${CC_NAME}.tar.gz
    echo "===== Chaincode is installed on peer0.org1 ====="

    setGlobalsForPeer0Org2
    peer lifecycle chaincode install ${CC_NAME}.tar.gz
    echo "===== Chaincode is installed on peer0.org2 ====="
}

```

Query Installed:

```

queryInstalled() {
    setGlobalsForPeer0Org1
    peer lifecycle chaincode queryinstalled >&log.txt
    cat log.txt
    PACKAGE_ID=$(sed -n "/${CC_NAME}_${VERSION}/ {s/^Package ID: //; s/, Label:.*/$//; p;}" log.txt)
    echo PackageID is ${PACKAGE_ID}
    echo "===== Query installed successful on peer0.org1 on channel ====="
}

```

Approve (Chấp thuận):

```
approveForMyOrg1() {
    setGlobalsForPeer0Org1
    # set -x
    peer lifecycle chaincode approveformyorg -o localhost:7050 \
        --ordererTLSHostnameOverride orderer.example.com --tls \
        --collections-config $PRIVATE_DATA_CONFIG \
        --caf file $ORDERER_CA --channelID $CHANNEL_NAME --name ${CC_NAME} --version ${VERSION} \
        --init-required --package-id ${PACKAGE_ID} \
        --sequence ${VERSION}
    # set +x
    echo "===== chaincode approved from org 1 ===== "
```

Check commit readiness (kiểm tra sẵn sàng commit):

```
checkCommitReadyness() {
    setGlobalsForPeer0Org1
    peer lifecycle chaincode checkcommitreadiness \
        --collections-config $PRIVATE_DATA_CONFIG \
        --channelID $CHANNEL_NAME --name ${CC_NAME} --version ${VERSION} \
        --sequence ${VERSION} --output json --init-required
    echo "===== checking commit readiness from org 1 ====="
}
```

Committee

```
commitChaincodeDefination() {
    setGlobalsForPeer0Org1
    peer lifecycle chaincode commit -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com \
        --tls $CORE_PEER_TLS_ENABLED --cafile $ORDERER_CA \
        --channelID $CHANNEL_NAME --name ${CC_NAME} \
        --collections-config $PRIVATE_DATA_CONFIG \
        --peerAddresses localhost:7051 --tlsRootCertFiles $PEER0_ORG1_CA \
        --peerAddresses localhost:9051 --tlsRootCertFiles $PEER0_ORG2_CA \
        --version ${VERSION} --sequence ${VERSION} --init-required
}
```

Querry Commit: Kiểm tra trạng thái commit

```
queryCommitted() {
    setGlobalsForPeer0Org1
    peer lifecycle chaincode querycommitted --channelID $CHANNEL_NAME --name ${CC_NAME}
}
```

Invoke Init: Khởi động smart contract

```
chaincodeInvokeInit() {
    setGlobalsForPeer0Org1
    peer chaincode invoke -o localhost:7050 \
        --ordererTLSHostnameOverride orderer.example.com \
        --tls $CORE_PEER_TLS_ENABLED --cafile $ORDERER_CA \
        -C $CHANNEL_NAME -n ${CC_NAME} \
        --peerAddresses localhost:7051 --tlsRootCertFiles $PEER0_ORG1_CA \
        --peerAddresses localhost:9051 --tlsRootCertFiles $PEER0_ORG2_CA \
        --isInit -c '{"Args":[]}'
}
```

Sau khi chạy xong các bước, kiểm tra lại các container

Có 2 container được tạo ra chính là chaincode thực thi trên peer 0 của các tổ chức Org1 và Org2

4.3. Tương tác với chaincode

- Khởi tạo sổ cái ban đầu với peer0 của Org1

```
chaincodeInvokeInitLedger() {  
  
    setGlobalsForPeer0Org1  
  
    ## Init ledger  
    peer chaincode invoke -o localhost:7050 \  
        --ordererTLSHostnameOverride orderer.example.com \  
        --tls $CORE_PEER_TLS_ENABLED \  
        --cafile $ORDERER_CA \  
        -C $CHANNEL_NAME -n ${CC_NAME} \  
        --peerAddresses localhost:7051 --tlsRootCertFiles $PEER0_ORG1_CA \  
        --peerAddresses localhost:9051 --tlsRootCertFiles $PEER0_ORG2_CA \  
        -c '{"function": "initLedger", "Args":[]}'  
}  
chaincodeInvokeInitLedger
```

Đây là hàm khởi tạo trong chaincode:

```
func (s *SmartContract) initLedger(APIstub shim.ChaincodeStubInterface) sc.Response {  
    cars := []Car{  
        Car{Make: "Toyota", Model: "Prius", Colour: "blue", Owner: "Huy Hiep Le"},  
        Car{Make: "Ford", Model: "Mustang", Colour: "red", Owner: "LTKN"},  
        Car{Make: "Hyundai", Model: "Tucson", Colour: "green", Owner: "LMT"},  
        Car{Make: "Volkswagen", Model: "Passat", Colour: "yellow", Owner: "Max"},  
        Car{Make: "Tesla", Model: "S", Colour: "black", Owner: "Adriana"},  
        Car{Make: "Peugeot", Model: "205", Colour: "purple", Owner: "Michel"},  
        Car{Make: "Chery", Model: "S22L", Colour: "white", Owner: "Aarav"},  
        Car{Make: "Fiat", Model: "Punto", Colour: "violet", Owner: "Pari"},  
        Car{Make: "Tata", Model: "Nano", Colour: "indigo", Owner: "Valeria"},  
        Car{Make: "Holden", Model: "Barina", Colour: "brown", Owner: "Shotaro"},  
    }  
  
    i := 0  
    for i < len(cars) {  
        carAsBytes, _ := json.Marshal(cars[i])  
        APIstub.PutState("CAR"+strconv.Itoa(i), carAsBytes)  
        i = i + 1  
    }  
  
    return shim.Success(nil)  
}
```

Kết quả khi chạy:

```
2024-12-10 19:46:51.083 +07 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200  
root@hhl-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0#
```

- Truy vấn sổ cái với peer0 của Org2:

```
setGlobalsForPeer0Org2() {  
    export CORE_PEER_LOCALMSPID="Org2MSP"  
    export CORE_PEER_TLS_ROOTCERT_FILE=$PEER0_ORG2_CA  
    export CORE_PEER_MSPCONFIGPATH=${PWD}/artifacts/channel/crypto-config/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp  
    export CORE_PEER_ADDRESS=localhost:9051  
}  
  
# chaincodeQuery  
chaincodeQueryAllCar() [  
    setGlobalsForPeer0Org2  
    peer chaincode query -C $CHANNEL_NAME -n ${CC_NAME} -c '{"function": "queryAllCars"}'  
]  
chaincodeQueryAllCar  
# Run this function if you add any new dependency in chaincode
```

Kết quả:

```

root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# ./deployChaincode.sh
[{"Key": "CAR0", "Record": {"colour": "blue", "make": "Toyota", "model": "Prius", "owner": "Tomoko"}}, {"Key": "CAR1", "Record": {"colour": "red", "make": "Ford", "model": "Mustang", "owner": "Brad"}}, {"Key": "CAR2", "Record": {"colour": "green", "make": "Hyundai", "model": "Tucson", "owner": "Jin Soo"}}, {"Key": "CAR3", "Record": {"colour": "yellow", "make": "Volkswagen", "model": "Passat", "owner": "Max"}}, {"Key": "CAR4", "Record": {"colour": "black", "make": "Tesla", "model": "S", "owner": "Adriana"}}, {"Key": "CAR5", "Record": {"colour": "purple", "make": "Peugeot", "model": "205", "owner": "Michel"}}, {"Key": "CAR6", "Record": {"colour": "white", "make": "Cherry", "model": "S22L", "owner": "Aarav"}}, {"Key": "CAR7", "Record": {"colour": "violet", "make": "Fiat", "model": "Punto", "owner": "Pari"}}, {"Key": "CAR8", "Record": {"colour": "indigo", "make": "Tata", "model": "Nano", "owner": "Valeria"}}, {"Key": "CAR9", "Record": {"colour": "brown", "make": "Holden", "model": "Barina", "owner": "Shotaro"}]}
root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0#

```

5. Chính sửa cơ chế đồng thuận

```

Options:
# TickInterval is the time interval between two Node.Tick invocations.
TickInterval: 500ms

# ElectionTick is the number of Node.Tick invocations that must pass
# between elections. That is, if a follower does not receive any
# message from the leader of current term before ElectionTick has
# elapsed, it will become candidate and start an election.
# ElectionTick must be greater than HeartbeatTick.
ElectionTick: 10

# HeartbeatTick is the number of Node.Tick invocations that must
# pass between heartbeats. That is, a leader sends heartbeat
# messages to maintain its leadership every HeartbeatTick ticks.
HeartbeatTick: 1

# MaxInflightBlocks limits the max number of in-flight append messages
# during optimistic replication phase.
MaxInflightBlocks: 5

# SnapshotIntervalSize defines number of bytes per which a snapshot is taken
SnapshotIntervalSize: 16 MB

```

Có thể chỉnh sửa cơ chế đồng thuận RAFT bằng cách chỉnh sửa các trường sau

Giải thích:

TickInterval: là khoảng thời gian giữa 2 lần mà các orderer tiến hành kiểm tra trạng thái và cập nhật giao dịch

ElectionTick: Số lần gọi Node.Tick cần thiết giữa các cuộc bầu cử. Nếu một follower không nhận được thông điệp từ leader trong khoảng thời gian này, nó sẽ trở thành candidate và bắt đầu một cuộc bầu cử mới. ElectionTick phải lớn hơn HeartbeatTick.

HeartbeatTick: Leader sẽ gửi thông điệp qua mỗi lần Node.Tick. Ở trên HeartbeatTick bằng 1 có nghĩa là 1 lần Node.Tick Leader sẽ gửi 1 thông điệp đến các node orderer còn lại.

MaxInflightBlocks: số block tối đa được chờ để xử lý. Ở trên MaxInflightBlocks khi 5 block được gửi tới các order khác và vẫn đang chờ để xử lý thì block thứ 6 phải đợi cho đến khi 1 block được xác nhận mới có thể được gửi và xử lý.

SnapshotIntervalSize: Khi tổng dung lượng của log đạt hoặc vượt 16 MB, cụm Raft sẽ:

- **Tạo snapshot:** Lưu trạng thái hiện tại của toàn hệ thống.
- **Xóa log cũ:** Giải phóng dung lượng bằng cách loại bỏ log trước thời điểm snapshot

Những tùy chọn này rất quan trọng để cân bằng giữa hiệu suất, độ tin cậy và tính sẵn sàng trong cụm Raft.

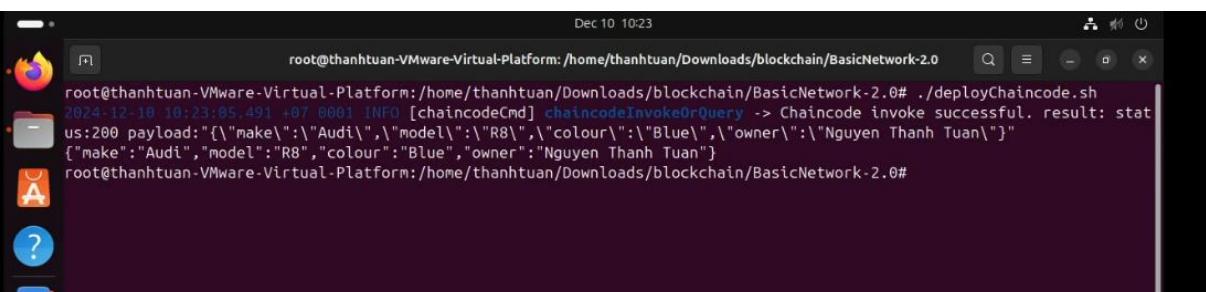
IV. Demo

1. Demo 1: Thực hiện transaction và kiểm tra trên ledger sau khi commit thành công

Link demo: <https://drive.google.com/drive/folders/1rofN2L-Ma4N4KmwUK2L7odm0Q9lOoNME?usp=sharing>

```
## Create Car
peer chaincode invoke -o localhost:7050 \
--ordererTLSHostnameOverride orderer.example.com \
--tls $CORE_PEER_TLS_ENABLED \
--cafile $ORDERER_CA \
-C $CHANNEL_NAME -n ${CC_NAME} \
--peerAddresses localhost:7051 \
--tlsRootCertFiles $PEER0_ORG1_CA \
--peerAddresses localhost:9051 --tlsRootCertFiles $PEER0_ORG2_CA \
-c '{"function": "createCar", "Args":["Car-Nhom10", "Audi", "R8", "Blue", "Nguyen Thanh Tuan"]}'
```

Truy vấn lại sổ cái



```
Dec 10 10:23:05.491 +07 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:{"\\"make\\":\\"Audi\\",\\"model\\":\\"R8\\",\\"colour\\":\\"Blue\\",\\"owner\\":\\"Nguyen Thanh Tuan\\"}" {"make": "Audi", "model": "R8", "colour": "Blue", "owner": "Nguyen Thanh Tuan"} root@thanhtuan-VMware-Virtual-Platform:/home/thanhtuan/Downloads/blockchain/BasicNetwork-2.0#
```

2. Demo 2: Thực hiện tắt 1 trong 3 orderer và xem kết quả thực thi đồng thời xem log của các node orderer

Link demo: <https://drive.google.com/drive/folders/1qzzHbBxmGlC8Ce1Ec-QbXG8pPO58hM4k3?usp=sharing>

- Thực hiện giao dịch khi chưa tắt orderer:

```
createCar() []
setGlobalsForPeer0Org2
peer chaincode invoke -o localhost:7050 \
--ordererTLSHostnameOverride orderer.example.com \
--tls $CORE_PEER_TLS_ENABLED \
--cafile $ORDERER_CA \
-C $CHANNEL_NAME -n ${CC_NAME} \
--peerAddresses localhost:7051 --tlsRootCertFiles $PEER0_ORG1_CA \
--peerAddresses localhost:9051 --tlsRootCertFiles $PEER0_ORG2_CA \
-c '{"function": "createCar", "Args":["CAR22", "Ford", "Mustang", "Red", "LMT"]}' \
# | c '{"function": "createCar", "Args":["CAR31", "Mazda", "CX-5", "Red", "LTKN"]}'
```

Kết quả

```

root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# ./deployChaincode.sh
root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# ./deployChaincode.sh
2024-12-11 08:01:17.348 +07 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful. result: status:200 payload:>{"make":"Ford","model":"Mustang","color":"Red","owner":"LMT"}"
```

• Tiến hành tắt 1 orderer

Tắt container đang chạy orderer.com.example

```

root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# docker stop b85090f127b1
b85090f127b1
root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0#
```

Thực hiện lại lệnh thêm tài sản:

```

root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# ./deployChaincode.sh
Error: error getting broadcast client: orderer client failed to connect to localhost:7050: failed to create new connection: connection error: desc = "transport: error while dialing: dial tcp 127.0.0.1:7050: connect: connection refused"
```

Kết quả không được

Lý do: Cơ chế đồng thuận RAFT để bầu được leader thì phải có trên 50% số phiếu mới bầu được leader nên là khi tắt 1 orderer thì tỉ lệ là 50% khiến cho cơ chế đồng thuận không sử dụng được

Log của orderer 2 và orderer 3:

```

root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# 
2024-12-10 13:29:43.147 UTC [orderer.consensus.etcdraft] run -> INFO 004 Raft leader changed: 0 -> 2 channel=mychannel node=2
2024-12-10 13:29:43.147 UTC [orderer.consensus.etcdraft] run -> INFO 04F Start accepting requests as Raft leader at block [0] channel=mychannel node=2
2024-12-10 13:30:07.224 UTC [orderer.consensus.etcdraft] propose -> INFO 050 Created block [1], there are 6 blocks in flight channel=mychannel node=2
2024-12-10 13:30:07.225 UTC [orderer.consensus.etcdraft] run -> INFO 051 Received config transaction, pause accepting transaction till it is committed channel=mychannel node=2
2024-12-10 13:30:07.226 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 052 Writing block [1] (Raft index: 5) to ledger channel=mychannel node=2
2024-12-10 13:30:07.386 UTC [orderer.consensus.etcdraft] propose -> INFO 053 Created block [2], there are 6 blocks in flight channel=mychannel node=2
2024-12-10 13:30:07.386 UTC [orderer.consensus.etcdraft] propose -> INFO 054 Received config transaction, pause accepting transaction till it is committed channel=mychannel node=2
2024-12-10 13:30:07.310 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 055 Writing block [2] (Raft index: 6) to ledger channel=mychannel node=2
2024-12-10 13:35:09.254 UTC [orderer.consensus.etcdraft] propose -> INFO 056 Created block [3], there are 6 blocks in flight channel=mychannel node=2
2024-12-10 13:35:09.257 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 057 Writing block [3] (Raft index: 7) to ledger channel=mychannel node=2
2024-12-10 13:35:11.486 UTC [orderer.consensus.etcdraft] propose -> INFO 058 Created block [4], there are 6 blocks in flight channel=mychannel node=2
2024-12-10 13:35:11.489 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 059 Writing block [4] (Raft index: 8) to ledger channel=mychannel node=2
2024-12-10 13:35:13.783 UTC [orderer.consensus.etcdraft] propose -> INFO 060 Created block [5], there are 6 blocks in flight channel=mychannel node=2
2024-12-10 13:35:13.783 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 061 Writing block [5] (Raft index: 9) to ledger channel=mychannel node=2
2024-12-10 13:35:16.353 UTC [orderer.consensus.etcdraft] propose -> INFO 062 Created block [6], there are 6 blocks in flight channel=mychannel node=2
2024-12-10 13:35:16.353 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 063 Writing block [6] (Raft index: 10) to ledger channel=mychannel node=2
2024-12-10 13:35:21.432 UTC [orderer.consensus.etcdraft] propose -> INFO 064 Created block [7], there are 6 blocks in flight channel=mychannel node=2
2024-12-10 13:42:58.448 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 065 Writing block [7] (Raft index: 11) to ledger channel=mychannel node=2
2024-12-10 13:42:58.458 UTC [orderer.consensus.etcdraft] propose -> INFO 066 Created block [8], there are 6 blocks in flight channel=mychannel node=2
2024-12-10 23:46:06.839 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 067 Writing block [8] (Raft index: 12) to ledger channel=mychannel node=2
2024-12-10 23:46:06.839 UTC [orderer.consensus.etcdraft] propose -> INFO 068 Created block [9], there are 6 blocks in flight channel=mychannel node=2
2024-12-10 23:46:06.839 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 069 Writing block [9] (Raft index: 13) to ledger channel=mychannel node=2
2024-12-10 23:48:38.445 UTC [orderer.consensus.etcdraft] propose -> INFO 070 Created block [10], there are 6 blocks in flight channel=mychannel node=2
2024-12-10 23:48:38.458 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 071 Writing block [10] (Raft index: 14) to ledger channel=mychannel node=2
2024-12-11 01:05:11.361 UTC [orderer.consensus.etcdraft] propose -> INFO 072 Created block [11], there are 6 blocks in flight channel=mychannel node=2
2024-12-11 01:05:11.361 UTC [orderer.common.cluster] handleMessage -> WARN 068 Stream read from 172.18.0.3:54956 failed: rpc error: code = Canceled desc = context canceled
2024-12-11 01:05:11.297 UTC [orderer.common.cluster] handleMessage -> WARN 069 Stream read from 172.18.0.3:54956 failed: rpc error: code = Canceled desc = context canceled
2024-12-11 01:05:11.297 UTC [com.google.grpc.server] 1 -> INFO 069 streaming call completed grpc.service=orderer.Cluster grpc.method=Step grpc.peer_address=172.18.0.3:54956 grpc.peer_subject=t=CN=orderer.example.com,L=San Francisco,ST=California,C=US error=rpc error: code = Canceled desc = context canceled" grpc.code=Canceled grpc.call_duration=8ms@0m20.491456187s
2024-12-11 01:05:11.298 UTC [com.google.grpc.server] 1 -> INFO 066 streaming call completed grpc.service=orderer.Cluster grpc.method=Step grpc.peer_address=172.18.0.3:54956 grpc.peer_subject=t=CN=orderer.example.com,L=San Francisco,ST=California,C=US error=rpc error: code = Canceled desc = context canceled" grpc.code=Canceled grpc.call_duration=8ms@0m44.5769933638s
2024-12-11 01:05:12.226 UTC [orderer.consensus.etcdraft] logSendFailure -> ERRO 066 failed to send StepRequest to 1, because: aborted channel=mychannel node=2
```

```

root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# root@hh1-VMware:/home/hhl/Desktop/hhl/BasicNetwork-2.0# 
2024-12-10 13:29:42.141 UTC [orderer.consensus.etcdraft] start -> INFO 047 Starting raft mode as part of a new channel channel=mychannel node=3
2024-12-10 13:29:42.141 UTC [orderer.consensus.etcdraft] becomefollower -> INFO 048 3 became follower at term 0 channel=mychannel node=3
2024-12-10 13:29:42.141 UTC [orderer.consensus.etcdraft] newraft -> INFO 049 newraft 3 [peers: [], term: 0, commit: 0, applied: 0, lastIndex: 0, lastTerm: 0] channel=mychannel node=3
2024-12-10 13:29:42.141 UTC [orderer.consensus.etcdraft] becomefollower -> INFO 049 3 became follower at term 1 channel=mychannel node=3
2024-12-10 13:29:42.143 UTC [orderer.consensus.etcdraft] apply -> INFO 050 Applied config change to add node 1, current nodes in channel: [1 2 3] channel=mychannel node=3
2024-12-10 13:29:42.143 UTC [orderer.consensus.etcdraft] apply -> INFO 04C Applied config change to add node 2, current nodes in channel: [1 2 3] channel=mychannel node=3
2024-12-10 13:29:42.143 UTC [orderer.consensus.etcdraft] apply -> INFO 051 Applied config change to add node 3, current nodes in channel: [1 2 3] channel=mychannel node=3
2024-12-10 13:29:43.145 UTC [orderer.consensus.etcdraft] Step -> INFO 040 3 [logTerm: 1, index: 3, vote: 0] cast MsgPreVote at term 1 channel=mychannel node=3
2024-12-10 13:29:43.145 UTC [orderer.consensus.etcdraft] Step -> INFO 041 3 [logTerm: 1, index: 3, vote: 1] cast MsgPreCandidate at term 1 channel=mychannel node=3
2024-12-10 13:29:43.147 UTC [orderer.consensus.etcdraft] run -> INFO 052 raft.node: 3 elected leader 2 at term 2 channel=mychannel node=3
2024-12-10 13:29:43.147 UTC [orderer.consensus.etcdraft] run -> INFO 053 Raft leader changed: 0 -> 2 channel=mychannel node=3
2024-12-10 13:36:07.228 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 055 Writing block [1] (Raft index: 5) to ledger channel=mychannel node=3
2024-12-10 13:36:07.310 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 055 Writing block [2] (Raft index: 6) to ledger channel=mychannel node=3
2024-12-10 13:35:09.257 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 056 Writing block [3] (Raft index: 7) to ledger channel=mychannel node=3
2024-12-10 13:35:11.489 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 057 Writing block [4] (Raft index: 8) to ledger channel=mychannel node=3
2024-12-10 13:35:13.707 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 058 Writing block [5] (Raft index: 9) to ledger channel=mychannel node=3
2024-12-10 13:35:16.356 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 059 Writing block [6] (Raft index: 10) to ledger channel=mychannel node=3
2024-12-10 13:35:21.434 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 060 Writing block [7] (Raft index: 11) to ledger channel=mychannel node=3
2024-12-10 13:42:58.450 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 065 Writing block [8] (Raft index: 12) to ledger channel=mychannel node=3
2024-12-10 23:48:38.451 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 066 Writing block [9] (Raft index: 13) to ledger channel=mychannel node=3
2024-12-10 01:04:19.376 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 067 Writing block [10] (Raft index: 14) to ledger channel=mychannel node=3
2024-12-11 01:05:11.294 UTC [orderer.common.cluster] handleMessage -> WARN 068 Stream read from 172.18.0.3:50030 failed: rpc error: code = Canceled desc = context canceled
2024-12-11 01:05:11.294 UTC [orderer.common.cluster] handleMessage -> WARN 069 Stream read from 172.18.0.3:50030 failed: rpc error: code = Canceled desc = context canceled
2024-12-11 01:05:11.295 UTC [com.google.grpc.server] 1 -> INFO 063 streaming call completed grpc.service=orderer.Cluster grpc.method=Step grpc.peer_address=172.18.0.3:50030 grpc.peer_subject=t=CN=orderer.example.com,L=San Francisco,ST=California,C=US error=rpc error: code = Canceled desc = context canceled" grpc.code=Canceled grpc.call_duration=8ms@0m45.589555135s
2024-12-11 01:05:11.298 UTC [com.google.grpc.server] 1 -> INFO 062 streaming call completed grpc.service=orderer.Cluster grpc.method=Step grpc.peer_address=172.18.0.3:50030 grpc.peer_subject=t=CN=orderer.example.com,L=San Francisco,ST=California,C=US error=rpc error: code = Canceled desc = context canceled" grpc.code=Canceled grpc.call_duration=8ms@0m44.5769933638s
2024-12-11 01:05:12.189 UTC [orderer.consensus.etcdraft] logSendFailure -> ERRO 063 Failed to send StepRequest to 1, because: aborted channel=sys-channel node=3
```

- Bật lại orderer và tiến hành giao dịch

Bật lại node orderer đã tắt

```
Dec 11 08:06
root@hh1-VMware:/home/hh/Desktop/hh/BasicNetwork-2.0# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
1e12ef231e6 dev-peer.0.org1.example.com:fan-fabcar-1.25df8261cf9dbffbe5c8ae2d8780661205b8c483c0c7c7e0998c223421b772ac-842bf449d28bc25a8e391eb28bc31b3bce32b56c616a421e254844fc612f22 dev-peer.0.org1.example.com:fabcar-1.25df8261cf9dbffbe5c8ae2d8780661205b8c483c0c7c7e0998c223421b772ac-842bf449d28bc25a8e391eb28bc31b3bce32b56c616a421e254844fc612f22 dev-peer.0.org2.example.com:fabcar-1.25df8261cf9dbffbe5c8ae2d8780661205b8c483c0c7c7e0998c223421b772ac-842bf449d28bc25a8e391eb28bc31b3bce32b56c616a421e254844fc612f22 dev-peer.0.org2.example.com:fabcar-1.25df8261cf9dbffbe5c8ae2d8780661205b8c483c0c7c7e0998c223421b772ac-842bf449d28bc25a8e391eb28bc31b3bce32b56c616a421e254844fc612f22 dev-peer.0.org3.example.com:fabric-peer.2.1 peer node start - 12 hours ago Up 12 hours 0.0.0.0:7051->7051/tcp, :::7051->7051/tcp peer0.org1.example.com
d4b413891ec hyperledger/fabric-orderer:2.1 orderer - 12 hours ago Up 12 hours 7050/tcp, 0.0.0.0:9050->9050/tcp, :::9050->9050/tcp, 0.0.0.0:8445->8443/tcp, [::]:8445->8443/tcp orderer3.example.com
c16d9ff6d39 hyperledger/fabric-couchdb tini -- /docker-entrypoint-init.d/couchdb - 12 hours ago Up 12 hours 4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp, :::5984->5984/tcp couchdb
cz51923162t hyperledger/fabric-peer.2.1 peer node start - 12 hours ago Up 12 hours 7051/tcp, 0.0.0.0:9051->9051/tcp, :::9051->9051/tcp peer0.org2.example.com
437aa3a167a1 hyperledger/fabric-peer.2.1 peer node start - 12 hours ago Up 12 hours 7051/tcp, 0.0.0.0:10051->10051/tcp, :::10051->10051/tcp peer1.org1.example.com
ba3b940e01f1 hyperledger/fabric-couchdb tini -- /docker-entrypoint-init.d/couchdb - 12 hours ago Up 12 hours 4369/tcp, 9100/tcp, 0.0.0.0:9984->5984/tcp, [::]:9984->5984/tcp couchdb3
b710848bc598 hyperledger/fabric-couchdb tini -- /docker-entrypoint-init.d/couchdb - 12 hours ago Up 12 hours 4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp, [::]:6984->5984/tcp couchdb1
3d3fb8a687e0 hyperledger/fabric-couchdb tini -- /docker-entrypoint-init.d/couchdb - 12 hours ago Up 12 hours 4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp, [::]:7984->5984/tcp couchdb2
34ca7c6939da hyperledger/fabric-cica sh -c "fabric-ca-server" - 12 hours ago Up 12 hours 0.0.0.0:8054->7854/tcp, [::]:8054->7854/tcp ca.org2.example.com
c5c51da3c32 hyperledger/fabric-peer.2.1 peer node start - 12 hours ago Up 12 hours 7051/tcp, 0.0.0.0:8051->8051/tcp, :::8051->8051/tcp peer1.org1.example.com
0261faad9b2 hyperledger/fabric-orderer:2.1 orderer - 12 hours ago Up 12 hours 7050/tcp, 0.0.0.0:8050->8050/tcp, :::8050->8050/tcp, 0.0.0.0:8444->8443/tcp, [::]:8444->8443/tcp orderer2.example.com
6602c1884d0a hyperledger/fabric-cica sh -c "fabric-ca-server" - 12 hours ago Up 12 hours 0.0.0.0:7054->7054/tcp, :::7054->7054/tcp ca.org1.example.com

To return to your computer, move the mouse pointer outside or press Ctrl+Alt
```

Log của order được bật lại

```
Dec 11 06:09
root@hh1-VMware:~/home/hh1/Desktop/hh1/BasicNetwork-2.0
root@hh1-VMware:~/home/hh1/Desktop/hh1/BasicNetwork-2.0
root@hh1-VMware:~/home/hh1/Desktop/hh1/BasicNetwork-2.0
root@hh1-VMware:~/home/hh1/Desktop/hh1/BasicNetwork-2.0

2024-12-11 01:09:00.881 UTC [orderer.common.cluster] updateStubInMapping -> INFO 024 Deactivating node 2 in channel sys-channel with endpoint of orderer2.example.com:8050 due to TLS certificate change
2024-12-11 01:09:00.882 UTC [orderer.common.cluster] updateStubInMapping -> INFO 025 Allocating a new stub for node 3 with endpoint of orderer3.example.com:9050 for channel sys-channel
2024-12-11 01:09:00.882 UTC [orderer.common.cluster] updateStubInMapping -> INFO 026 Deactivating node 3 in channel sys-channel with endpoint of orderer3.example.com:9050 due to TLS certificate change
2024-12-11 01:09:00.882 UTC [orderer.common.cluster] applyMembershipConfig -> INFO 027 Z2 exists in both old and new membership for channel sys-channel , skipping its deactivation
2024-12-11 01:09:00.882 UTC [orderer.common.cluster] applyMembershipConfig -> INFO 028 Z3 exists in both old and new membership for channel sys-channel , skipping its deactivation
2024-12-11 01:09:00.887 UTC [orderer.common.cluster] Configure -> INFO 029 Exiting
2024-12-11 01:09:00.882 UTC [orderer.consensus.etcdraft] start -> INFO 02a Restarting raft node channel-sys-channel node=1
2024-12-11 01:09:00.882 UTC [orderer.consensus.etcdraft] becomeFollower -> INFO 02b I became follower at term 2 channel=sys-channel node=1
2024-12-11 01:09:00.882 UTC [orderer.consensus.etcdraft] newRaft -> INFO 02c newRaft 1 [peers: [], term: 2, commit: 4, applied: 0, lastIndex: 5, lastTerm: 2] channel=sys-channel node=1
2024-12-11 01:09:00.883 UTC [orderer.common.Server] Main -> INFO 02d Starting orderer:
Version: 2.1.1
Commit SHA: 6393adb
Go version: go1.14.1
OS/Arch: linux/amd64
2024-12-11 01:09:00.883 UTC [orderer.common.server] Main -> INFO 02e Beginning to serve requests
2024-12-11 01:09:00.885 UTC [orderer.consensus.etcdraft] apply -> INFO 02f Applied config change to add node 1, current nodes in channel: [1] channel=sys-channel node=1
2024-12-11 01:09:00.885 UTC [orderer.consensus.etcdraft] apply -> INFO 030 Applied config change to add node 2, current nodes in channel: [1 2] channel=sys-channel node=1
2024-12-11 01:09:00.886 UTC [orderer.consensus.etcdraft] apply -> INFO 031 Applied config change to add node 3, current nodes in channel: [1 2 3] channel=sys-channel node=1
2024-12-11 01:09:05.892 UTC [orderer.consensus.etcdraft] Step -> INFO 032 I 1 is starting a new election at term 2 channel=sys-channel node=1
2024-12-11 01:09:05.892 UTC [orderer.consensus.etcdraft] becomePreCandidate -> INFO 033 I 1 became pre-candidate at term 2 channel=sys-channel node=1
2024-12-11 01:09:05.892 UTC [orderer.consensus.etcdraft] poll -> INFO 034 I received MsgPreVoteRes from 1 at term 2 channel=sys-channel node=1
2024-12-11 01:09:05.892 UTC [orderer.consensus.etcdraft] campaign -> INFO 035 1 [logterm: 2, index: 5] sent MsgPreVote request to 2 at term 2 channel=sys-channel node=1
2024-12-11 01:09:05.892 UTC [orderer.consensus.etcdraft] campaign -> INFO 036 1 [logterm: 2, index: 5] sent MsgPreVote request to 3 at term 2 channel=sys-channel node=1
2024-12-11 01:09:06.851 UTC [orderer.consensus.etcdraft] Step -> INFO 037 I 1 is starting a new election at term 2 channel=mychannel node=1
2024-12-11 01:09:06.851 UTC [orderer.consensus.etcdraft] becomePreCandidate -> INFO 038 I 1 became pre-candidate at term 2 channel=mychannel node=1
2024-12-11 01:09:06.852 UTC [orderer.consensus.etcdraft] poll -> INFO 039 I received MsgPreVoteRes from 1 at term 2 channel=mychannel node=1
2024-12-11 01:09:06.852 UTC [orderer.consensus.etcdraft] campaign -> INFO 040 1 [logterm: 2, index: 15] sent MsgPreVote request to 2 at term 2 channel=mychannel node=1
2024-12-11 01:09:06.852 UTC [orderer.consensus.etcdraft] campaign -> INFO 041 1 [logterm: 2, index: 15] sent MsgPreVote request to 3 at term 2 channel=mychannel node=1
um to your computer, move the mouse pointer outside or press Ctrl+Alt.
```

Log của orderer 2

```
Dec 11 08:09
root@hh1-VMware:~/home/hh1/Desktop/hh1/BasicNetwork-2.0

root@hh1-VMware:~/home/hh1/Desktop/hh1/BasicNetwork-2.0 x root@hh1-VMware:~/home/hh1/Desktop/hh1/BasicNetwork-2.0 x root@hh1-VMware:~/home/hh1/Desktop/hh1/BasicNetwork-2.0 x root@hh1-VMware:~/home/hh1/Desktop/hh1/BasicNetwork-2.0

[2024-12-10 13:35:09.254 UTC [orderer.consensus.etcdraft] propose -> INFO 056 Created block [3], there are 0 blocks in flight channel=mychannel node=2
[2024-12-10 13:35:09.257 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 057 Writing block [3] (Raft index: 7) to ledger channel=mychannel node=2
[2024-12-10 13:35:11.484 UTC [orderer.consensus.etcdraft] propose -> INFO 058 Created block [4], there are 0 blocks in flight channel=mychannel node=2
[2024-12-10 13:35:11.488 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 059 Writing block [4] (Raft index: 8) to ledger channel=mychannel node=2
[2024-12-10 13:35:13.703 UTC [orderer.consensus.etcdraft] propose -> INFO 060 Created block [5], there are 0 blocks in flight channel=mychannel node=2
[2024-12-10 13:35:13.707 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 060 Writing block [5] (Raft index: 9) to ledger channel=mychannel node=2
[2024-12-10 13:35:16.353 UTC [orderer.consensus.etcdraft] propose -> INFO 060 Created block [6], there are 0 blocks in flight channel=mychannel node=2
[2024-12-10 13:35:16.356 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 060 Writing block [6] (Raft index: 10) to ledger channel=mychannel node=2
[2024-12-10 13:35:21.431 UTC [orderer.consensus.etcdraft] propose -> INFO 065 Created block [7], there are 0 blocks in flight channel=mychannel node=2
[2024-12-10 13:35:21.433 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 065 Writing block [7] (Raft index: 11) to ledger channel=mychannel node=2
[2024-12-10 13:42:58.448 UTC [orderer.consensus.etcdraft] propose -> INFO 069 Created block [8], there are 0 blocks in flight channel=mychannel node=2
[2024-12-10 13:42:58.450 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 061 Writing block [8] (Raft index: 12) to ledger channel=mychannel node=2
[2024-12-10 23:46:06.839 UTC [orderer.consensus.etcdraft] propose -> INFO 062 Created block [9], there are 0 blocks in flight channel=mychannel node=2
[2024-12-10 23:46:06.851 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 063 Writing block [9] (Raft index: 13) to ledger channel=mychannel node=2
[2024-12-10 23:48:38.445 UTC [orderer.consensus.etcdraft] propose -> INFO 064 Created block [10], there are 0 blocks in flight channel=mychannel node=2
[2024-12-10 23:48:38.450 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 065 Writing block [10] (Raft index: 14) to ledger channel=mychannel node=2
[2024-12-11 01:08:49.361 UTC [orderer.consensus.etcdraft] propose -> INFO 066 Created block [11], there are 0 blocks in flight channel=mychannel node=2
[2024-12-11 01:08:49.375 UTC [orderer.consensus.etcdraft] writeBlock -> INFO 067 Writing block [11] (Raft index: 15) to ledger channel=mychannel node=2
[2024-12-11 01:08:49.376 UTC [orderer.common.cluster] handleMessage -> WARN 068 Stream read from 172.18.0.3:54956 failed; rpc error: code = Canceled desc = context canceled
[2024-12-11 01:08:51.297 UTC [orderer.common.cluster] handleMessage -> WARN 069 Stream read from 172.18.0.3:54956 failed; rpc error: code = Canceled desc = context canceled
[2024-12-11 01:08:51.297 UTC [com.grpc.server] 1 -> INFO 069 streaming call completed grpc.service=orderer.Cluster grpc.method=Step grpc.peer_address=172.18.0.3:54956 grpc.peer_subject=CN=orderer.example.com,OU=San Francisco,ST=California,CUS="" error="rpc error: code = Canceled desc = context canceled" grpc.code=Canceled grpc.call.duration=895m20.491456187s
[2024-12-11 01:08:51.298 UTC [com.grpc.server] 1 -> INFO 069 streaming call completed grpc.service=orderer.Cluster grpc.method=Step grpc.peer_address=172.18.0.3:54956 grpc.peer_subject=CN=orderer.example.com,OU=San Francisco,ST=California,CUS="" error="rpc error: code = Canceled desc = context canceled" grpc.code=Canceled grpc.call.duration=895m44.570993363s
[2024-12-11 01:08:51.298 UTC [orderer.consensus.etcdraft] logSendFailure -> ERROR 066 Failed to send StepRequest to 1, because: aborted channel=mychannel node=2
[2024-12-11 01:08:53.899 UTC [orderer.consensus.etcdraft] Step -> INFO 080 2 [logterm: 2, index: 5, vote: 3] ignored MsgPreVote from 1 [logterm: 2, index: 5] at term 2: lease is not expired (remaining ticks: 9) channel=sys.channel node=2
[2024-12-11 01:08:56.855 UTC [orderer.consensus.etcdraft] Step -> INFO 086 2 [logterm: 2, index: 15, vote: 2] ignored MsgPreVote from 1 [logterm: 2, index: 15] at term 2: lease is not expired (remaining ticks: 3) channel=mychannel node=2
[2024-12-11 01:08:57.887 UTC [orderer.consensus.etcdraft] Step -> INFO 086 2 [logterm: 2, index: 5, vote: 3] ignored MsgPreVote from 1 [logterm: 2, index: 5] at term 2: lease is not expired (remaining ticks: 9) channel=sys.channel node=2
[2024-12-11 01:09:12.057 UTC [orderer.consensus.etcdraft] Step -> INFO 070 2 [logterm: 2, index: 15, vote: 2] ignored MsgPreVote from 1 [logterm: 2, index: 15] at term 2: lease is not expired (remaining ticks: 1) channel=mychannel node=2
```

Log của oderer 3:

Thực hiện lai thêm tài sản mới

```
createCar() {
    setGlobalsForPeer0Org2
    peer chaincode invoke -o localhost:7050 \
        --ordererTLSHostnameOverride orderer.example.com \
        --tls $CORE_PEER_TLS_ENABLED \
        --cafile $ORDERER_CA \
        -C $CHANNEL_NAME -n ${CC_NAME} \
        --peerAddresses localhost:7051 --tlsRootCertFiles $PEER0_ORG1_CA \
        --peerAddresses localhost:9051 --tlsRootCertFiles $PEER0_ORG2_CA \
        -c '{"function": "createCar","Args":["CAR31", "Mazda", "CX-5", "Red", "LTKN"]}'
```

Quá trình này vẫn **chưa bộc lộ** ra được leader nên chưa thực hiện được giao dịch

Sau một thời gian thực hiện giao dịch đã thành công

V. Tài liệu tham khảo

<https://hyperledger-fabric.readthedocs.io/en/release-2.5/>

<https://www.youtube.com/watch?v=SJTdJt6N6Ow&list=PLSBNVhWU6KjW4qo1RlmR7cvvV8XIILub6>