Lab: 90-minutes to Hyperledger Fabric

By Mark Anthony Morris
Connect with me at LinkedIn:
https://www.linkedin.com/in/markmorrissupergeek/
Email: mark.morris@tarantulatechnology.com

In this lab, we will look a creating an Hyperledger Fabric Test Network, and deploying and testing an Hyperledger Fabric Smart Contract.

The lab is slit into 4 parts.

PART 1: Prerequisites--describes what we need installed to run the Lab.

PART 2: Install Samples, Binaries, and Docker Images--describes how to set up the environment for running the Lab.

PART 3: Using the Fabric test network--describes how to interact with the Lab test network to start and stop the network, and deploy and test the exampe smart contract.

PART 4: Screen Prints--contains (for your reference) Screen Prints of the primary commands we run in the Lab.

PART 1: Prerequisites

Before you begin, you should confirm that you have installed all the prerequisites below.

Install Git

Download the latest version of git if it is not already installed, or if you have problems running the curl commands.

Install cURL

Download the latest version of the cURL tool if it is not already installed or if you get errors running the curl commands.

Note
If you're on Windows please see the specific note on Windows extras below.

Docker and Docker Compose

You will need the following installed:

- MacOSX, *nix, or Windows 10: Docker - Docker version 17.06.2-ce or greater is required.
- Older versions of Windows: Docker Toolbox - again, Docker version Docker 17.06.2-ce or greater is required.

You can check the version of Docker you have installed with the following command from a terminal prompt:

```
docker --version
```

Note
The following applies to linux systems running systemd.

Make sure the docker daemon is running:

```
sudo systemctl start docker
```

Optional: If you want the docker daemon to start when the system starts, use the following:

```
sudo systemctl enable docker
```

Add your user to the docker group:

```
sudo usermod -a -G docker <username>
```

Note
Installing Docker for Mac or Windows, or Docker Toolbox will also install Docker Compose. If you already had Docker installed, you should check that you have Docker Compose version 1.14.0 or greater installed. If not, install a more recent version of Docker.

You can check the version of Docker Compose you have installed with the following command from a terminal prompt:

```
docker-compose --version
```

Windows extras

On Windows 10 you should use the native Docker distribution and you may use the Windows PowerShell. However, for the `binaries` command to succeed you will still need to have the `uname` command available. You can get it as part of Git but beware that only the 64bit version is supported.

Before running any `git clone` commands, run the following commands:

```
git config --global core.autocrlf false
git config --global core.longpaths true
```

You can check the setting of these parameters with the following commands:

```
git config --get core.autocrlf
git config --get core.longpaths
```

These need to be `false` and `true` respectively.

The `curl` command that comes with Git and Docker Toolbox is old and does not handle properly the redirect used. Make sure you have and use a newer version which can be downloaded from the cURL downloads page

PART 2: Install Samples, Binaries, and Docker Images

Note
If you are running on **Windows** you will want to make use of the Docker Quickstart Terminal for the upcoming terminal commands. Please see the **Prerequisites** if you haven't previously installed Docker.

If you are using Docker Toolbox or macOS, you will need to use a location under `/Users` (macOS) when installing and running the lab.

If you are using Docker for Mac, you will need to use a location under `/Users`, `/Volumes`, `/private`, or `/tmp`. To use a different location, please consult the Docker documentation for file sharing.

If you are using Docker for Windows, please consult the Docker documentation for shared drives and use a location under one of the shared drives.

Note
Make sure Docker is running. You can test is Docker is running by executing:

```
docker --version
```

In a terminal window determine a location on your machine where you want to run the Lab and `cd` (change directory command) into that directory.

The `curl` command that follows will perform the following steps:

1. If needed, clone the hyperledger/fabric-samples repository
2. Checkout the appropriate version tag
3. Install the Hyperledger Fabric platform-specific binaries and config files for the version specified into the /bin and /config directories of ***fabric-samples***
4. Download the Hyperledger Fabric docker images for the version

specified

Once you are ready, and in the Lab directory, go ahead and execute the below `curl` command.

Note
The command below pulls down the latest production release.

```
curl -sSL https://bit.ly/2ysbOFE | bash -s
```

Note
If you want a specific release, pass a version identifier for Fabric, Fabric-ca and thirdparty Docker images. The command below demonstrates how to download the latest production releases (same the above command) - **Fabric v2.1.0** and **Fabric CA v1.4.6**

```
curl -sSL https://bit.ly/2ysbOFE | bash -s -- <fabric_version>
<fabric-ca_version> <thirdparty_version>
curl -sSL https://bit.ly/2ysbOFE | bash -s -- 2.1.0 1.4.6 0.4.18
```

Note
If you get an error running the above curl command, you may have too old a version of curl that does not handle redirects or an unsupported environment. Please see the **Prerequisites** for additional information on where to find the latest version of curl and get the right environment. Alternately, you can substitute the un-shortened URL: https://raw.githubusercontent.com/hyperledger/fabric/master/scripts/bootstrap.sh

The above `curl` command downloads and executes a bash script that will download and extract all of the platform-specific binaries you will need to set up your Hyperledger Fabric network and place them into the created cloned repo. The following platform-specific binaries are retrieved:

- `configtxgen`,
- `configtxlator`,
- `cryptogen`,
- `discover`,
- `idemixgen`
- `orderer`,
- `peer`,

- `fabric-ca-client`,
- `fabric-ca-server`

and places them in the `bin` sub-directory of the ***fabric-samples*** repo directory.

[optional] You may want to add the `bin` sub-directory to your **PATH** environment variable, so the executable commands can be picked up without fully qualifying the path to each command. e.g.:

```
export PATH=<path to download location>/bin:$PATH
```

The script will also download the Hyperledger Fabric docker images from Docker Hub into your local Docker registry and tag them as 'latest'.

Finally, upon conclusion the script lists out the Docker images installed.

Look at the names for each image; these are the components that will ultimately comprise the Hyperledger Fabric network.

Also notice that you have two instances of the same image ID - one tagged as "amd64-1.x.x" and one tagged as "latest". Prior to 1.2.0, the image being downloaded was determined by `uname -m` and showed as "x86_64-1.x.x".

PART 3: Using the Fabric test network

After completing PART 2 you have downloaded the Hyperledger Fabric executables, Docker images, and samples. We will deploy a test network by using scripts that are provided in the `fabric-samples` repository. Use the test network to learn about Fabric by running nodes on your local machine. More experienced developers can use the network to test their smart contracts and applications. The network is meant to be used only as a tool for education and testing. It should not be used as a template for deploying a production network. The test network, introduced in Fabric v2.0, is the replacement for the `first-network` sample.

The test network deploys a Fabric network using Docker Compose. Because the nodes are isolated within a Docker Compose network, the test network is not configured to connect to other running fabric nodes.

Note
These instructions have been verified to work against the latest stable Docker images and the pre-compiled setup utilities within the supplied tar file. If you run these commands with images or tools from the current master branch, it is possible that you will encounter errors.

Before we begin

Before we can bring up the test network, we need to clone the `fabric-samples` repository and download the Fabric images. To contine you must have perfromed **PART 1: Prerequisites** and **PART 2: Installed the Samples, Binaries and Docker Images**.

Bring up the test network

We find the scripts to bring up the test network in the `test-network` directory of the `fabric-samples` repository directory. Change to the test network directory using the following command:

```
cd fabric-samples/test-network
```

In this directory, we find an annotated script, `network.sh`, this script starts up a Fabric network using the Docker images on your local machine.

You can run `./network.sh -h` to print the script help text seen below:

```
Usage:
  network.sh <Mode> [Flags]
    <Mode>
      - 'up' - bring up fabric orderer and peer nodes. No
channel is created
      - 'up createChannel' - bring up fabric network with one
channel
      - 'createChannel' - create and join a channel after the
network is created
      - 'deployCC' - deploy the fabcar chaincode on the channel
      - 'down' - clear the network with docker-compose down
      - 'restart' - restart the network

    Flags:
    -ca <use CAs> -  create Certificate Authorities to generate
the crypto material
    -c <channel name> - channel name to use (defaults to
"mychannel")
    -s <dbtype> - the database backend to use: goleveldb
(default) or couchdb
    -r <max retry> - CLI times out after certain number of
attempts (defaults to 5)
    -d <delay> - delay duration in seconds (defaults to 3)
    -l <language> - the programming language of the chaincode to
deploy: go (default), java, javascript, typescript
    -v <version>  - chaincode version. Must be a round number,
1, 2, 3, etc
    -i <imagetag> - the tag to be used to launch the network
(defaults to "latest")
    -cai <ca_imagetag> - the image tag to be used for CA
(defaults to "1.4.6")
    -verbose - verbose mode
  network.sh -h (print this message)

 Possible Mode and flags
  network.sh up -ca -c -r -d -s -i -verbose
  network.sh up createChannel -ca -c -r -d -s -i -verbose
  network.sh createChannel -c -r -d -verbose
  network.sh deployCC -l -v -r -d -verbose
 Taking all defaults:
    network.sh up

 Examples:
  network.sh up createChannel -ca -c mychannel -s couchdb -i
2.0.0
  network.sh createChannel -c channelName
  network.sh deployCC -l javascript
```

From inside the `test-network` directory, run the following command to remove any containers or artifacts from any previous runs:

```
./network.sh down
```

We can start the network by issuing the following command:

Note
You will experience problems if you try to run the script from another directory.

```
./network.sh up
```

This command creates a Fabric network that consists of two peer nodes, one ordering node. No channel is created when you run `./network.sh up`, we will create a channel below in: **Creating a channel**.

See **Screen Print 1** in **Screen Prints** for an example of the output from this command.

If you don't get this result, jump down to Troubleshooting for help on what might have gone wrong. By default, the network uses the cryptogen tool to bring up the network.

The components of the test network

After our test network is deployed, we can take some time to examine its components. Run the following command to list all of Docker containers that are running. We should see the three nodes that were created by the `network.sh` script:

```
docker ps -a
```

See **Screen Print 2** in **Screen Prints** for an example of the output from this command.

Each node and user that interacts with a Fabric network needs to belong to an organization that is a network member. The group of organizations that are members of a Fabric network are often referred to as the consortium. The test network has two consortium members, Org1 and Org2. The network also includes one orderer organization that maintains the ordering service of the network.

Peers are the fundamental components of any Fabric network. Peers store the blockchain ledger and validate transactions before they are committed to the ledger. Peers run the smart contracts that contain the business logic that is used to manage the assets on the blockchain ledger.

Every peer in the network needs to belong to a member of the consortium. In the test network, each organization operates one peer, `peer0.org1.example.com` and `peer0.org2.example.com`.

Every Fabric network also includes an ordering service. While peers validate transactions and add blocks of transactions to the blockchain ledger, they do not decide on the order of transactions or include them into new blocks. On a distributed network, peers may be running far away from each other and not have a common view of when a transaction was created. Coming to consensus on the order of transactions is a costly process that would create overhead for the peers.

An ordering service allows peers to focus on validating transactions and committing them to the ledger. After ordering nodes receive endorsed transactions from clients, they come to consensus on the order of transactions and then add them to blocks. The blocks are then distributed to peer nodes, which add the blocks to the blockchain ledger. Ordering nodes also operate the system channel that defines the capabilities of a Fabric network, such as how blocks are made and which version of Fabric that nodes can use. The system channel defines which organizations are members of the consortium.

The test network uses a single node Raft ordering service that is operated by the ordering organization. You can see the ordering node running on our machine as `orderer.example.com` in **Screen Print 2** in **Screen Prints**. While the test network only uses a single node ordering service, a real network would have multiple ordering nodes, operated by one or multiple orderer organizations. The different ordering nodes would use the Raft consensus algorithm to come to agreement on the order of transactions across the network.

Creating a channel

Now that we have peer and orderer nodes running on our machine, we can use the script to create a Fabric channel for transactions between Org1 and Org2. Channels are a private layer of communication between specific network members. Channels can be used only by organizations that are invited to the channel, and are invisible to other members of the network. Each channel has a separate blockchain ledger. Organizations that have been invited "join" their peers to the channel to store the channel ledger and validate transactions on the channel.

We can use the `network.sh` script to create a channel between Org1 and Org2 and join their peers to the channel. Run the following command to create a channel with the default name of `mychannel`:

```
./network.sh createChannel
```

See **Screen Print 3** in **Screen Prints** for an example of the output from this command.

If the command was successful, we see the following message printed at the end of our output:

```
========= Channel successfully joined ===========
```

We can also use the channel flag to create a channel with a custom name. As an example, the following command would create a channel named `channel1`:

```
./network.sh createChannel -c channel1
```

The channel flag also allows us to create multiple channels by specifying different channel names. After we create `mychannel`, we can use the command below to create a second channel named `mychannel2`:

```
./network.sh createChannel -c mychannel2
```

If you want to bring up the network and create a channel in a single step, you can use the `up` and `createChannel` modes together:

```
./network.sh up createChannel
```

Starting a chaincode on the channel

After we have created a channel, we can start using smart contracts to interact with the channel ledger. Smart contracts contain the business logic that governs assets on the blockchain ledger. Applications run by members of the network can invoke smart contracts to create assets on the ledger, as well as change and transfer those assets. Applications also query smart contracts to read data on the ledger.

To ensure that transactions are valid, transactions created using smart contracts typically need to be signed by multiple organizations to be committed to the channel ledger. Multiple signatures are integral to the trust model of Fabric. Requiring multiple endorsements for a transaction prevents one organization on a channel from tampering with the ledger on their peer or using business logic that was not agreed to. To sign a transaction, each organization needs to invoke and execute the smart contract on their peer, which then signs the output of the transaction. If the output is consistent and has been signed by enough organizations, the transaction can be committed to the ledger. The policy that specifies the set of organizations on the channel that need to execute the smart contract is referred to as the endorsement policy, which is set for each chaincode (smart contract) as part of the chaincode definition.

In Fabric, smart contracts are deployed on the network in packages referred to as chaincode. A Chaincode is installed on the peers of an organization and then deployed to a channel, where it can then be used to endorse transactions and interact with the blockchain ledger. Before a chaincode can be deployed to a channel, the members of the channel need to agree on a chaincode definition that establishes chaincode governance. When the required number of organizations agree, the chaincode definition can be committed to the channel, and the chaincode is ready to be used.

After we have used the `network.sh` to create a channel, we can start a chaincode on the channel using the following command:

```
./network.sh deployCC
```

See **Screen Print 4** in **Screen Prints** for an example of the output from this command.

The `deployCC` subcommand will install the **fabcar** chaincode on `peer0.org1.example.com` and `peer0.org2.example.com` and then deploy the chaincode on the channel specified using the channel flag (or `mychannel` if no channel is specified).

If we are deploying chaincode for the first time, the script will install the chaincode dependencies. By default, The script installs the Go version of the fabcar chaincode. However, we can use the language flag, `-l`, to install the Java or javascript versions of the chaincode.

We can find the Fabcar chaincode in the `chaincode` folder of the `fabric-samples` directory. This folder contains sample chaincode that are provided as examples and used by other tutorials to highlight Fabric features.

After the **fabcar** chaincode definition has been committed to the channel, the script initializes the chaincode by invoking the `init` function and then invokes the chaincode to put an initial list of cars on the ledger. The script then queries the chaincode to verify the that the data was added. If the chaincode was installed, deployed, and invoked correctly, you should see the following list of cars printed at the end of the command output:

```
[{"Key":"CAR0", "Record":
{"make":"Toyota","model":"Prius","colour":"blue","owner":"Tomoko
"}},
{"Key":"CAR1", "Record":
{"make":"Ford","model":"Mustang","colour":"red","owner":"Brad"}}
,
{"Key":"CAR2", "Record":
{"make":"Hyundai","model":"Tucson","colour":"green","owner":"Jin
Soo"}},
{"Key":"CAR3", "Record":
{"make":"Volkswagen","model":"Passat","colour":"yellow","owner":
"Max"}},
{"Key":"CAR4", "Record":
{"make":"Tesla","model":"S","colour":"black","owner":"Adriana"}}
,
{"Key":"CAR5", "Record":
{"make":"Peugeot","model":"205","colour":"purple","owner":"Miche
l"}},
{"Key":"CAR6", "Record":
{"make":"Chery","model":"S22L","colour":"white","owner":"Aarav"}
},
{"Key":"CAR7", "Record":
{"make":"Fiat","model":"Punto","colour":"violet","owner":"Pari"}
},
{"Key":"CAR8", "Record":
{"make":"Tata","model":"Nano","colour":"indigo","owner":"Valeria
"}},
{"Key":"CAR9", "Record":
{"make":"Holden","model":"Barina","colour":"brown","owner":"Shot
aro"}}]
===================== Query successful on peer0.org1 on channel
'mychannel' =====================
```

Interacting with the network

After we bring up the test network, create a channel, and deploy chaincode we can use the `peer` CLI (command line interface) to interact with our network.

The `peer` CLI allows us to invoke deployed smart contracts, update channels, or install and deploy new smart contracts from the CLI.

Note
Make sure we are in the `test-network` directory when executing the below peer CLI commands.

The `peer` binary command is in the `bin` folder of the `fabric-samples` repository directory.

Run the following command to add the binary commands to your Path:

```
export PATH=${PWD}/../bin:$PATH
```

We also need to set the `FABRIC_CFG_PATH` to point to the `core.yaml` file in the `fabric-samples` repository, so run this command:

```
export FABRIC_CFG_PATH=$PWD/../config/
```

We can now set the environment variables that allow us to operate the `peer` CLI as Org1. Run the following commands:

```
# Environment variables for Org1

export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/
peerOrganizations/org1.example.com/peers/peer0.org1.example.com/
tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/
peerOrganizations/org1.example.com/users/Admin@org1.example.com/
msp
export CORE_PEER_ADDRESS=localhost:7051
```

The `CORE_PEER_TLS_ROOTCERT_FILE` and `CORE_PEER_MSPCONFIGPATH` environment variables point to the Org1 crypto material in the `organizations` folder.

Run the following command to get the list of cars that were added to our channel ledger:

```
peer chaincode query -C mychannel -n fabcar -c '{"Args":
["queryAllCars"]}'
```

If the command is successful, you can see the same list of cars that were printed in the logs when you ran the script:

```
[{"Key":"CAR0", "Record":
{"make":"Toyota","model":"Prius","colour":"blue","owner":"Tomoko
"}},
{"Key":"CAR1", "Record":
{"make":"Ford","model":"Mustang","colour":"red","owner":"Brad"}}
,
{"Key":"CAR2", "Record":
{"make":"Hyundai","model":"Tucson","colour":"green","owner":"Jin
Soo"}},
{"Key":"CAR3", "Record":
{"make":"Volkswagen","model":"Passat","colour":"yellow","owner":
"Max"}},
{"Key":"CAR4", "Record":
{"make":"Tesla","model":"S","colour":"black","owner":"Adriana"}}
,
{"Key":"CAR5", "Record":
{"make":"Peugeot","model":"205","colour":"purple","owner":"Miche
l"}},
{"Key":"CAR6", "Record":
{"make":"Chery","model":"S22L","colour":"white","owner":"Aarav"}
},
{"Key":"CAR7", "Record":
{"make":"Fiat","model":"Punto","colour":"violet","owner":"Pari"}
},
{"Key":"CAR8", "Record":
{"make":"Tata","model":"Nano","colour":"indigo","owner":"Valeria
"}},
{"Key":"CAR9", "Record":
{"make":"Holden","model":"Barina","colour":"brown","owner":"Shot
aro"}}]
```

Chaincodes are invoked when a network member wants to transfer or change an asset on the ledger. Run the following command to change the owner of a car on the ledger by invoking the fabcar chaincode:

```
peer chaincode invoke -o localhost:7050 --
ordererTLSHostnameOverride orderer.example.com --tls true --
cafile ${PWD}/organizations/ordererOrganizations/example.com/
orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-
cert.pem -C mychannel -n fabcar --peerAddresses localhost:7051
```

```
--tlsRootCertFiles ${PWD}/organizations/peerOrganizations/
org1.example.com/peers/peer0.org1.example.com/tls/ca.crt --
peerAddresses localhost:9051 --tlsRootCertFiles ${PWD}/
organizations/peerOrganizations/org2.example.com/peers/
peer0.org2.example.com/tls/ca.crt -c
'{"function":"changeCarOwner","Args":["CAR9","Dave"]}'
```

If the command is successful, you should see the following response:

```
2019-12-04 17:38:21.048 EST [chaincodeCmd]
chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful.
result: status:200
```

Note
If we were to deploy the Java chaincode, we would run the invoke command with the following arguments instead:

```
'{"function":"changeCarOwner","Args":["CAR009","Dave"]}'
```

The Fabcar chaincode written in Java uses a different index than the chaincode written in Javascipt or Go.

Because the endorsement policy for the fabcar chaincode requires the transaction to be signed by Org1 and Org2, the chaincode invoke command needs to target both `peer0.org1.example.com` and `peer0.org2.example.com` using the `--peerAddresses` flag.

Because TLS is enabled for the test network, the command needs to reference the TLS certificate for each peer using the `--tlsRootCertFiles` flag.

After we invoke the chaincode, we can use another query to see how the invoke changed the assets on the blockchain ledger.

Since we already queried the Org1 peer, we can take this opportunity to query the chaincode running on the Org2 peer.

Run the below commands to set the environment variables to operate as Org2:

```
# Environment variables for Org2

export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org2MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/
peerOrganizations/org2.example.com/peers/peer0.org2.example.com/
tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/
peerOrganizations/org2.example.com/users/Admin@org2.example.com/
msp
export CORE_PEER_ADDRESS=localhost:9051
```

We can now query the fabcar chaincode running on `peer0.org2.example.com`:

```
peer chaincode query -C mychannel -n fabcar -c '{"Args":
["queryCar","CAR9"]}'
```

The result will show that `"CAR9"` was transferred to Dave:

```
{"make":"Holden","model":"Barina","colour":"brown","owner":"Dave
"}
```

Bring down the network

When we are finished using the test network, we can bring down the network with the following command:

```
./network.sh down
```

See **Screen Print 5** in **Screen Prints** for an example of the output from this command.

The command will stop and remove the nodes and chaincode containers, delete the organization crypto material, and remove the chaincode images from your Docker Registry. The command also removes the channel artifacts and docker volumes from previous runs, allowing us to run `./network.sh up` again if we encountered any problems.

Troubleshooting

If you have any problems with the Lab, review the following:

You should always start your network fresh. You can use the following command to remove the artifacts, crypto material, containers, volumes, and chaincode images from previous runs:

```
./network.sh down
```

You **will** see errors if you do not remove old containers, images, and volumes.

If you see Docker errors, first check your Docker version (PART 1: Prerequisites), and then try restarting your Docker process. Problems with Docker are oftentimes not immediately recognizable. For example, you may see errors that are the result of your node not being able to access the crypto material mounted within a container.

If problems persist, you can remove your images and start from scratch:

```
docker rm -f $(docker ps -aq)
docker rmi -f $(docker images -q)
```

If you see errors on your create, approve, commit, invoke or query commands, make sure you have properly updated the channel name and chaincode name. There are placeholder values in the supplied sample commands.

If you see the error below:

```
Error: Error endorsing chaincode: rpc error: code = 2 desc =
Error installing chaincode code mycc:1.0(chaincode /var/
hyperledger/production/chaincodes/mycc.1.0 exits)
```

You likely have chaincode images (e.g. `dev-peer1.org2.example.com-fabcar-1.0` or `dev-peer0.org1.example.com-fabcar-1.0`) from prior runs. Remove them and try again.

```
docker rmi -f $(docker images | grep dev-peer[0-9] | awk '{print
$3}')
```

If you see the below error:

```
[configtx/tool/localconfig] Load -> CRIT 002 Error reading
configuration: Unsupported Config Type ""
panic: Error reading configuration: Unsupported Config Type ""
```

Then you did not set the `FABRIC_CFG_PATH` environment variable properly. The configtxgen tool needs this variable in order to locate the configtx.yaml. Go back and execute an `export FABRIC_CFG_PATH=$PWD/configtx/configtx.yaml`, then recreate your channel artifacts.

If you see an error stating that you still have “active endpoints”, then prune your Docker networks. This will wipe your previous networks and start you with a fresh environment:

```
docker network prune
```

You will see the following message:

```
WARNING! This will remove all networks not used by at least one
container.
Are you sure you want to continue? [y/N]
```

Select `y`.

If you see an error similar to the following:

```
/bin/bash: ./scripts/createChannel.sh: /bin/bash^M: bad
interpreter: No such file or directory
```

Ensure that the file in question (**createChannel.sh** in this Lab) is encoded in the Unix format. This was most likely caused by not setting `core.autocrlf` to `false` in your Git configuration (see Windows extras in PART 1). There are several ways of fixing this. If you have access to the vim editor for instance, open the file:

```
vim ./fabric-samples/test-network/scripts/createChannel.sh
```

Then change its format by executing the following vim command:

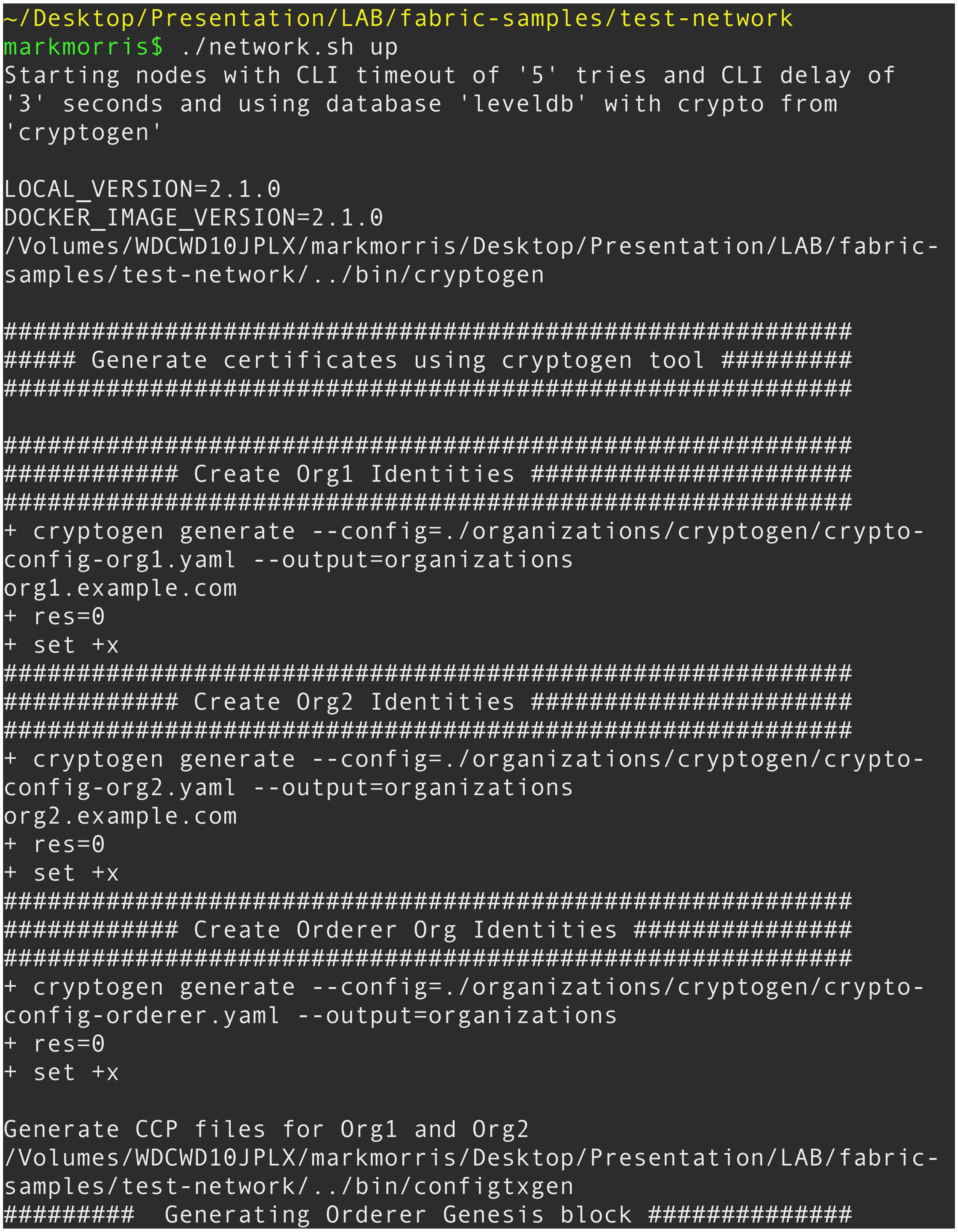
```
:set ff=unix
```

If your orderer exits upon creation or if you see that the create channel command fails due to an inability to connect to your ordering service, use the `docker logs` command to read the logs from the ordering node. You may see the following message:

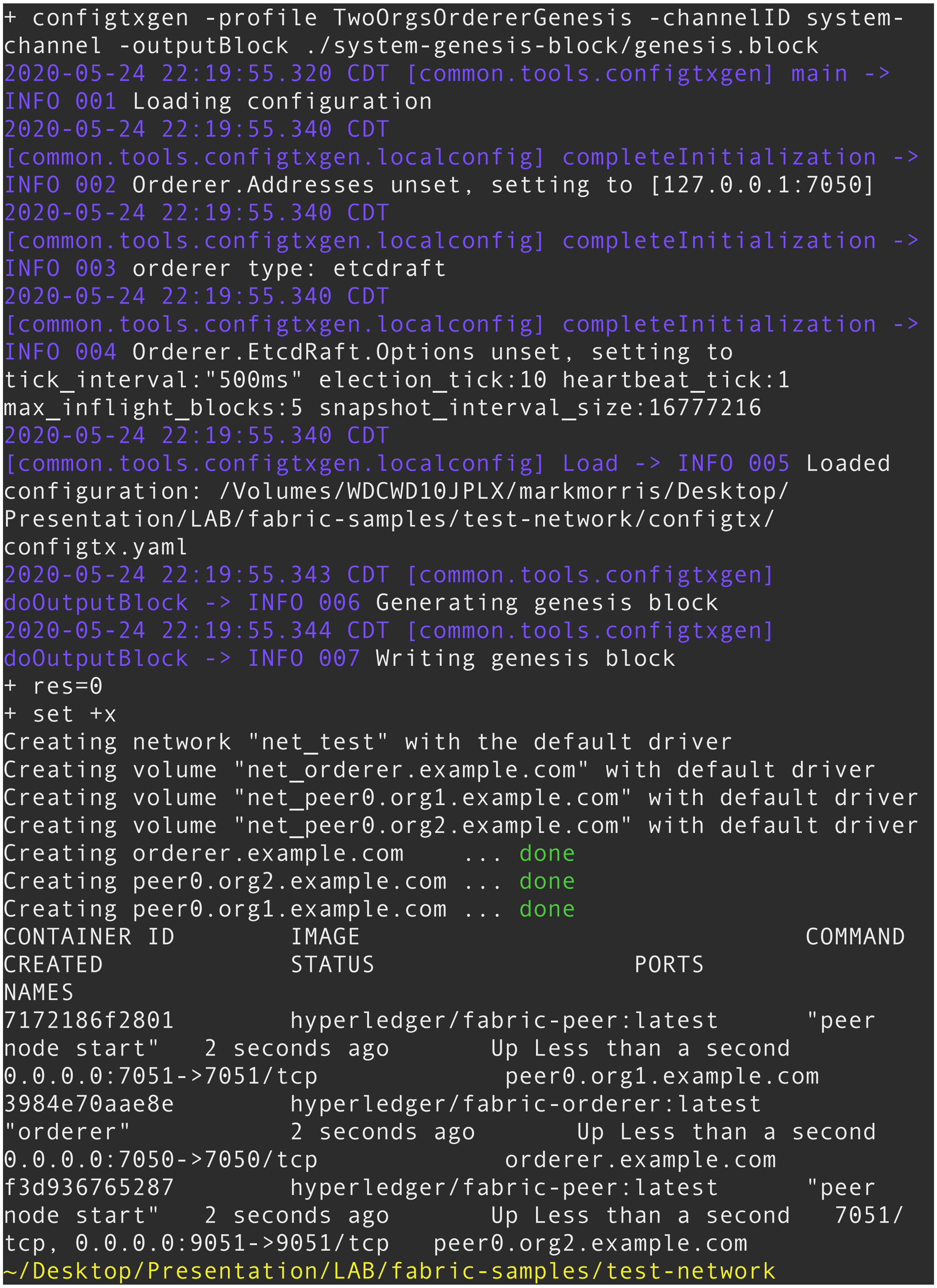
```
PANI 007 [channel system-channel] config requires unsupported
orderer capabilities: Orderer capability V2_0 is required but
not supported: Orderer capability V2_0 is required but not
supported
```

This occurs when you are trying to run the network using Fabric version 1.4.x docker images. The test network needs to run using Fabric version 2.x.

Screen Print 1



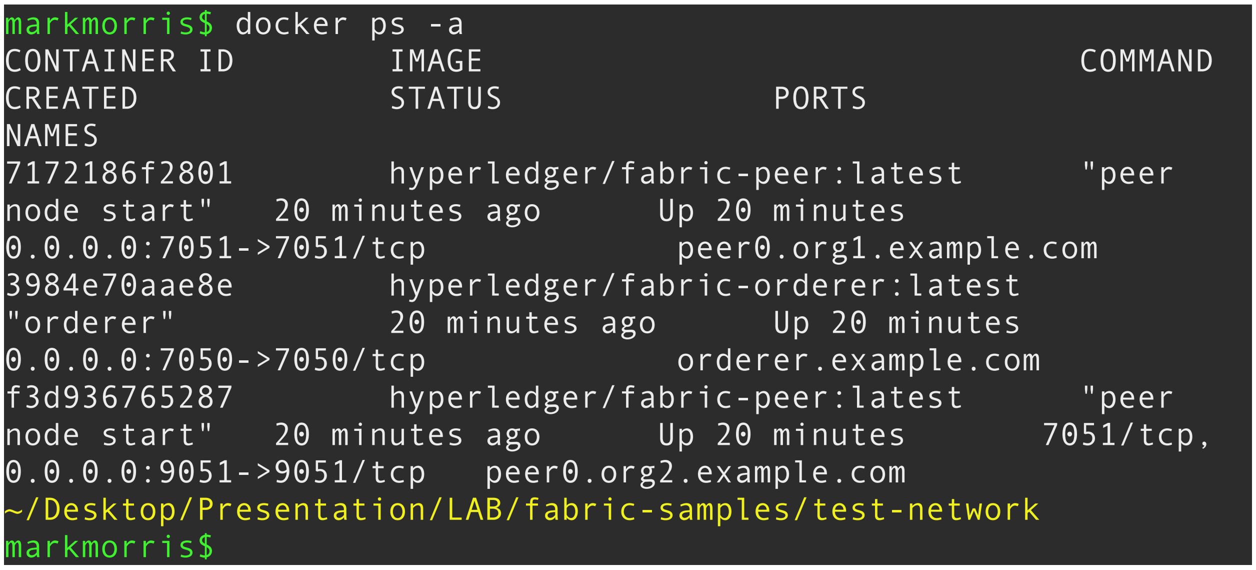
```
~/Desktop/Presentation/LAB/fabric-samples/test-network
markmorris$ ./network.sh up
Starting nodes with CLI timeout of '5' tries and CLI delay of
'3' seconds and using database 'leveldb' with crypto from
'cryptogen'

LOCAL_VERSION=2.1.0
DOCKER_IMAGE_VERSION=2.1.0
/Volumes/WDCWD10JPLX/markmorris/Desktop/Presentation/LAB/fabric-
samples/test-network/../bin/cryptogen

##########################################################
##### Generate certificates using cryptogen tool #########
##########################################################

##########################################################
############ Create Org1 Identities ######################
##########################################################
+ cryptogen generate --config=./organizations/cryptogen/crypto-
config-org1.yaml --output=organizations
org1.example.com
+ res=0
+ set +x
##########################################################
############ Create Org2 Identities ######################
##########################################################
+ cryptogen generate --config=./organizations/cryptogen/crypto-
config-org2.yaml --output=organizations
org2.example.com
+ res=0
+ set +x
##########################################################
############ Create Orderer Org Identities ###############
##########################################################
+ cryptogen generate --config=./organizations/cryptogen/crypto-
config-orderer.yaml --output=organizations
+ res=0
+ set +x

Generate CCP files for Org1 and Org2
/Volumes/WDCWD10JPLX/markmorris/Desktop/Presentation/LAB/fabric-
samples/test-network/../bin/configtxgen
#########  Generating Orderer Genesis block ##############
```



```
+ configtxgen -profile TwoOrgsOrdererGenesis -channelID system-
channel -outputBlock ./system-genesis-block/genesis.block
2020-05-24 22:19:55.320 CDT [common.tools.configtxgen] main ->
INFO 001 Loading configuration
2020-05-24 22:19:55.340 CDT
[common.tools.configtxgen.localconfig] completeInitialization ->
INFO 002 Orderer.Addresses unset, setting to [127.0.0.1:7050]
2020-05-24 22:19:55.340 CDT
[common.tools.configtxgen.localconfig] completeInitialization ->
INFO 003 orderer type: etcdraft
2020-05-24 22:19:55.340 CDT
[common.tools.configtxgen.localconfig] completeInitialization ->
INFO 004 Orderer.EtcdRaft.Options unset, setting to
tick_interval:"500ms" election_tick:10 heartbeat_tick:1
max_inflight_blocks:5 snapshot_interval_size:16777216
2020-05-24 22:19:55.340 CDT
[common.tools.configtxgen.localconfig] Load -> INFO 005 Loaded
configuration: /Volumes/WDCWD10JPLX/markmorris/Desktop/
Presentation/LAB/fabric-samples/test-network/configtx/
configtx.yaml
2020-05-24 22:19:55.343 CDT [common.tools.configtxgen]
doOutputBlock -> INFO 006 Generating genesis block
2020-05-24 22:19:55.344 CDT [common.tools.configtxgen]
doOutputBlock -> INFO 007 Writing genesis block
+ res=0
+ set +x
Creating network "net_test" with the default driver
Creating volume "net_orderer.example.com" with default driver
Creating volume "net_peer0.org1.example.com" with default driver
Creating volume "net_peer0.org2.example.com" with default driver
Creating orderer.example.com    ... done
Creating peer0.org2.example.com ... done
Creating peer0.org1.example.com ... done
CONTAINER ID        IMAGE                               COMMAND
CREATED             STATUS                  PORTS
NAMES
7172186f2801        hyperledger/fabric-peer:latest      "peer
node start"   2 seconds ago       Up Less than a second
0.0.0.0:7051->7051/tcp             peer0.org1.example.com
3984e70aae8e        hyperledger/fabric-orderer:latest
"orderer"           2 seconds ago       Up Less than a second
0.0.0.0:7050->7050/tcp             orderer.example.com
f3d936765287        hyperledger/fabric-peer:latest      "peer
node start"   2 seconds ago       Up Less than a second   7051/
tcp, 0.0.0.0:9051->9051/tcp   peer0.org2.example.com
~/Desktop/Presentation/LAB/fabric-samples/test-network
```

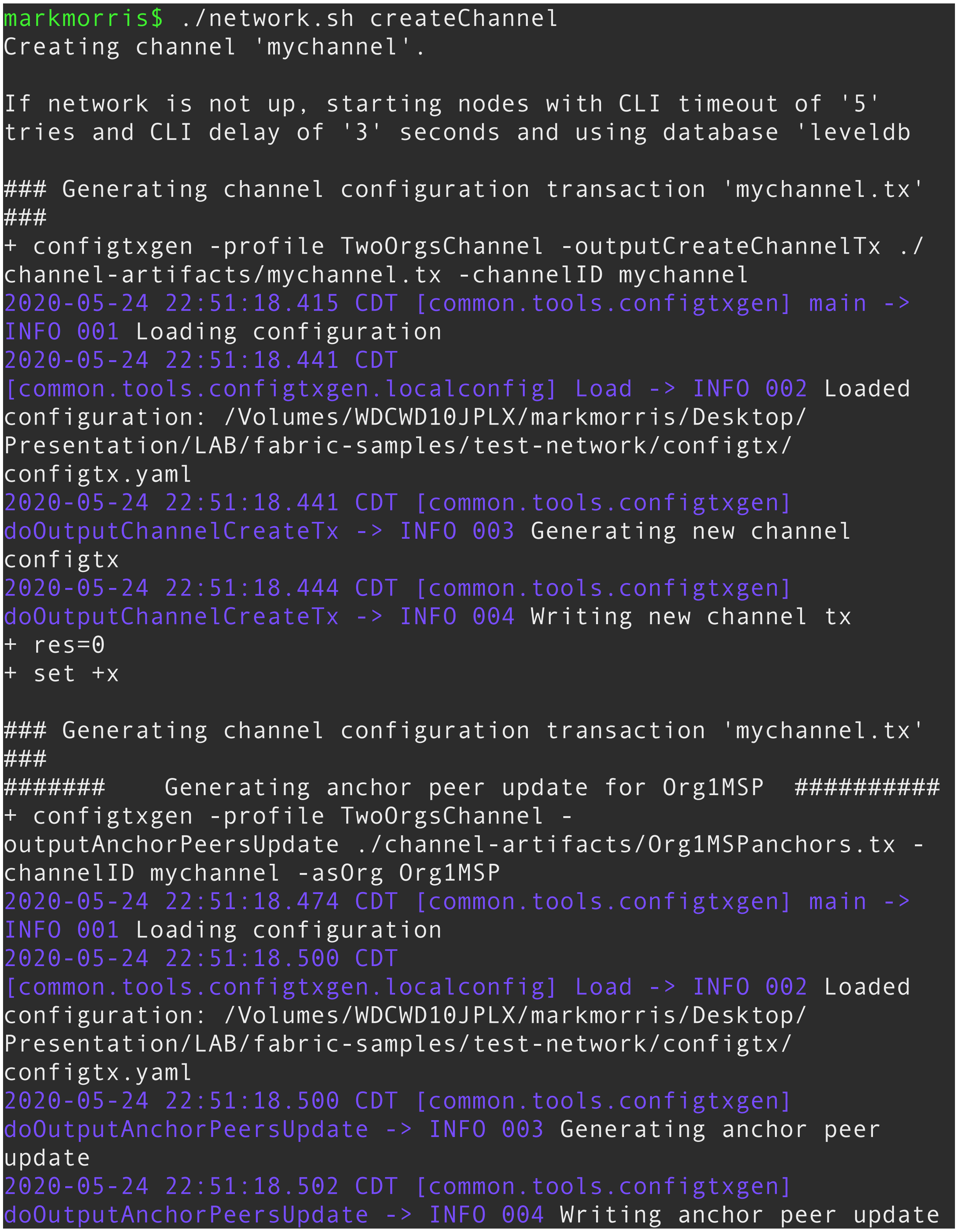
```
markmorris$
```

Screen Print 2



```
markmorris$ docker ps -a
CONTAINER ID        IMAGE                               COMMAND
CREATED             STATUS              PORTS
NAMES
7172186f2801        hyperledger/fabric-peer:latest      "peer
node start"   20 minutes ago      Up 20 minutes
0.0.0.0:7051->7051/tcp             peer0.org1.example.com
3984e70aae8e        hyperledger/fabric-orderer:latest
"orderer"           20 minutes ago      Up 20 minutes
0.0.0.0:7050->7050/tcp             orderer.example.com
f3d936765287        hyperledger/fabric-peer:latest      "peer
node start"   20 minutes ago      Up 20 minutes      7051/tcp,
0.0.0.0:9051->9051/tcp   peer0.org2.example.com
~/Desktop/Presentation/LAB/fabric-samples/test-network
markmorris$
```

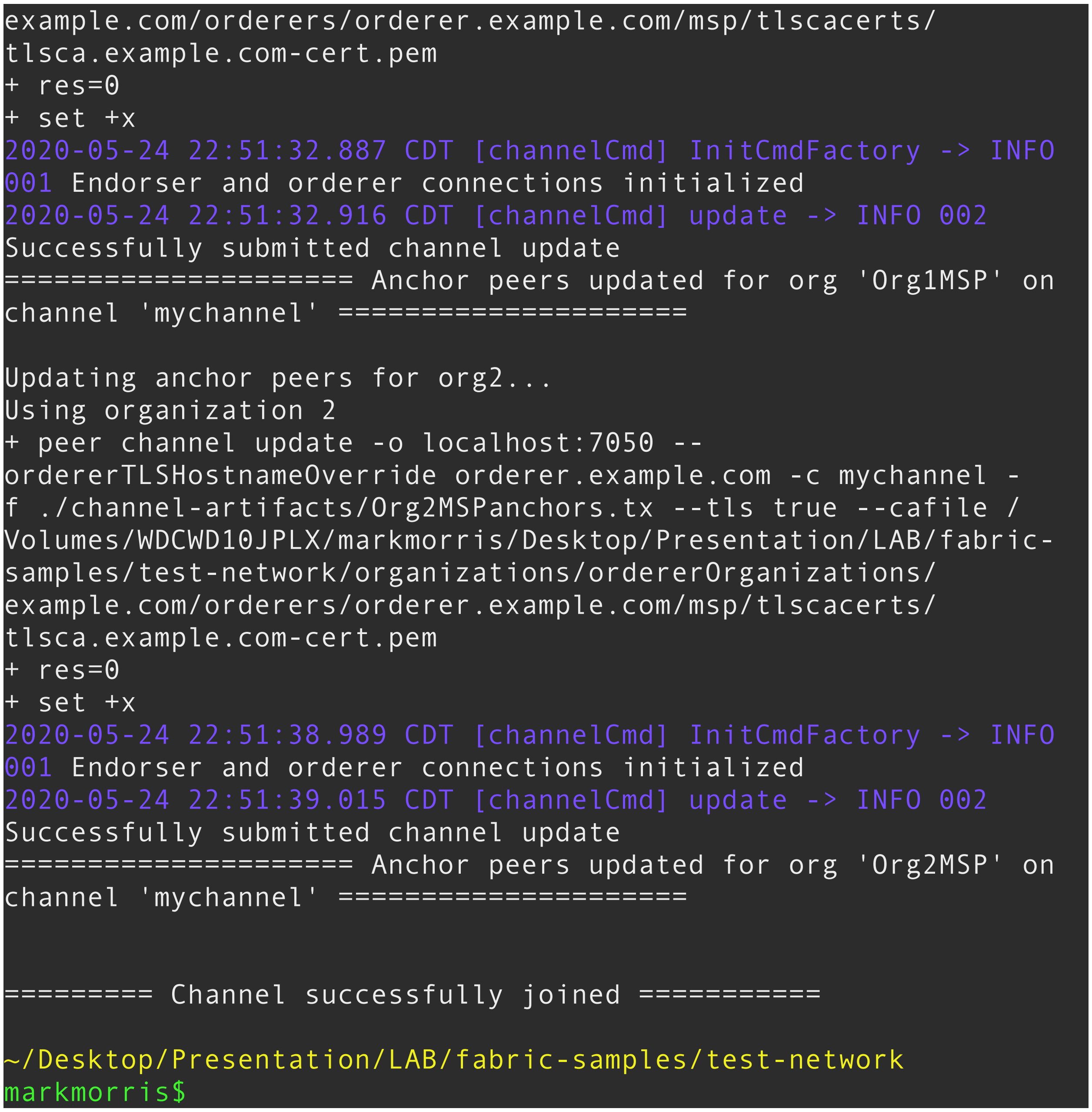
Screen Print 3



```
markmorris$ ./network.sh createChannel
Creating channel 'mychannel'.

If network is not up, starting nodes with CLI timeout of '5'
tries and CLI delay of '3' seconds and using database 'leveldb

### Generating channel configuration transaction 'mychannel.tx'
###
+ configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./
channel-artifacts/mychannel.tx -channelID mychannel
2020-05-24 22:51:18.415 CDT [common.tools.configtxgen] main ->
INFO 001 Loading configuration
2020-05-24 22:51:18.441 CDT
[common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded
configuration: /Volumes/WDCWD10JPLX/markmorris/Desktop/
Presentation/LAB/fabric-samples/test-network/configtx/
configtx.yaml
2020-05-24 22:51:18.441 CDT [common.tools.configtxgen]
doOutputChannelCreateTx -> INFO 003 Generating new channel
configtx
2020-05-24 22:51:18.444 CDT [common.tools.configtxgen]
doOutputChannelCreateTx -> INFO 004 Writing new channel tx
+ res=0
+ set +x

### Generating channel configuration transaction 'mychannel.tx'
###
#######    Generating anchor peer update for Org1MSP  ##########
+ configtxgen -profile TwoOrgsChannel -
outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -
channelID mychannel -asOrg Org1MSP
2020-05-24 22:51:18.474 CDT [common.tools.configtxgen] main ->
INFO 001 Loading configuration
2020-05-24 22:51:18.500 CDT
[common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded
configuration: /Volumes/WDCWD10JPLX/markmorris/Desktop/
Presentation/LAB/fabric-samples/test-network/configtx/
configtx.yaml
2020-05-24 22:51:18.500 CDT [common.tools.configtxgen]
doOutputAnchorPeersUpdate -> INFO 003 Generating anchor peer
update
2020-05-24 22:51:18.502 CDT [common.tools.configtxgen]
doOutputAnchorPeersUpdate -> INFO 004 Writing anchor peer update
```

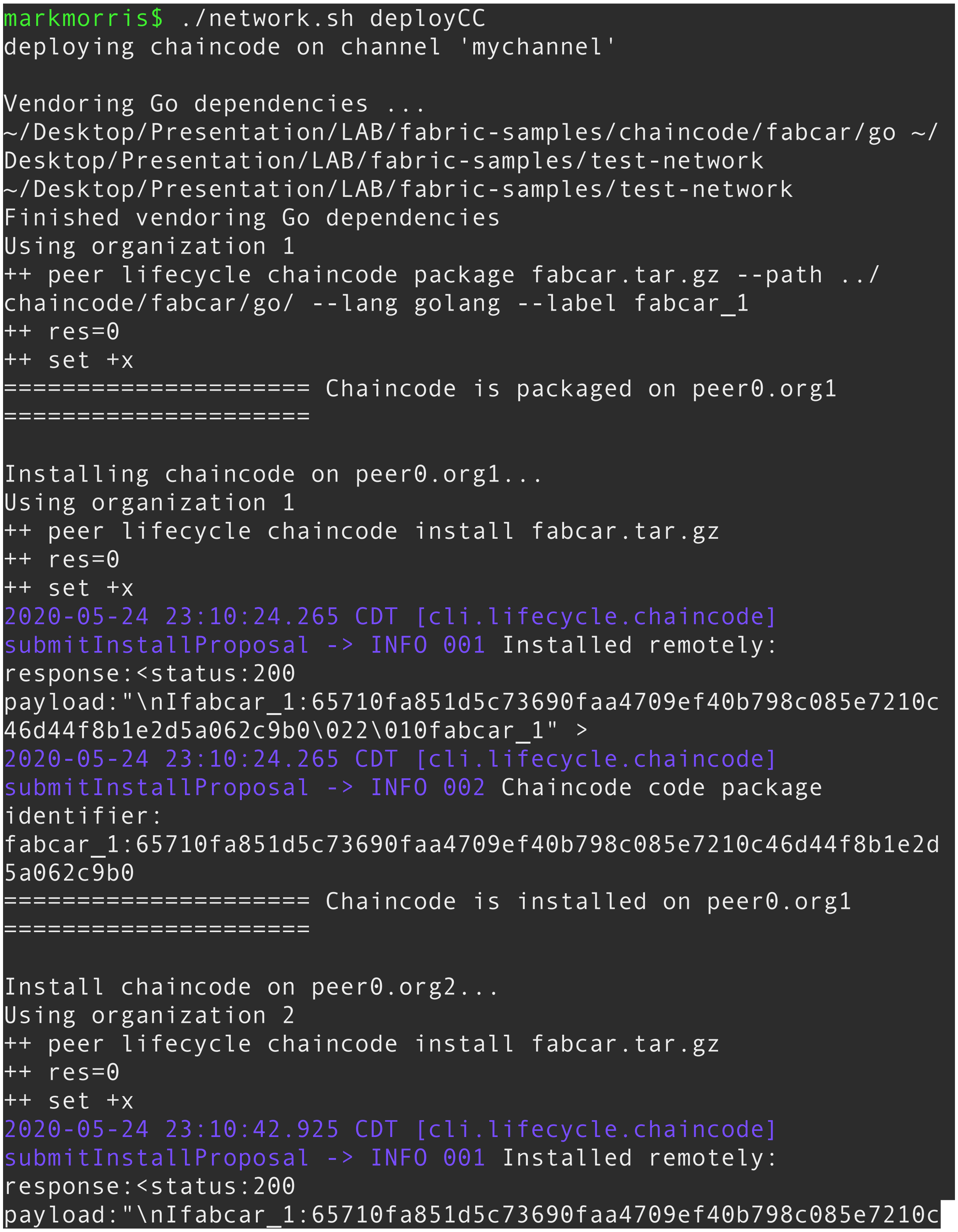
```
+ res=0
+ set +x

#######    Generating anchor peer update for Org2MSP  ##########
+ configtxgen -profile TwoOrgsChannel -
outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -
channelID mychannel -asOrg Org2MSP
2020-05-24 22:51:18.533 CDT [common.tools.configtxgen] main ->
INFO 001 Loading configuration
2020-05-24 22:51:18.555 CDT
[common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded
configuration: /Volumes/WDCWD10JPLX/markmorris/Desktop/
Presentation/LAB/fabric-samples/test-network/configtx/
configtx.yaml
2020-05-24 22:51:18.555 CDT [common.tools.configtxgen]
doOutputAnchorPeersUpdate -> INFO 003 Generating anchor peer
update
2020-05-24 22:51:18.558 CDT [common.tools.configtxgen]
doOutputAnchorPeersUpdate -> INFO 004 Writing anchor peer update
+ res=0
+ set +x

Creating channel mychannel
Using organization 1
+ peer channel create -o localhost:7050 -c mychannel --
ordererTLSHostnameOverride orderer.example.com -f ./channel-
artifacts/mychannel.tx --outputBlock ./channel-artifacts/
mychannel.block --tls true --cafile /Volumes/WDCWD10JPLX/
markmorris/Desktop/Presentation/LAB/fabric-samples/test-network/
organizations/ordererOrganizations/example.com/orderers/
orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
+ res=0
+ set +x
2020-05-24 22:51:21.807 CDT [channelCmd] InitCmdFactory -> INFO
001 Endorser and orderer connections initialized
2020-05-24 22:51:21.856 CDT [cli.common] readBlock -> INFO 002
Expect block, but got status: &{NOT_FOUND}
2020-05-24 22:51:21.865 CDT [channelCmd] InitCmdFactory -> INFO
003 Endorser and orderer connections initialized
2020-05-24 22:51:22.070 CDT [cli.common] readBlock -> INFO 004
Expect block, but got status: &{SERVICE_UNAVAILABLE}
2020-05-24 22:51:22.078 CDT [channelCmd] InitCmdFactory -> INFO
005 Endorser and orderer connections initialized
2020-05-24 22:51:22.284 CDT [cli.common] readBlock -> INFO 006
Expect block, but got status: &{SERVICE_UNAVAILABLE}
2020-05-24 22:51:22.293 CDT [channelCmd] InitCmdFactory -> INFO
```

```
007 Endorser and orderer connections initialized
2020-05-24 22:51:22.499 CDT [cli.common] readBlock -> INFO 008
Expect block, but got status: &{SERVICE_UNAVAILABLE}
2020-05-24 22:51:22.509 CDT [channelCmd] InitCmdFactory -> INFO
009 Endorser and orderer connections initialized
2020-05-24 22:51:22.714 CDT [cli.common] readBlock -> INFO 00a
Expect block, but got status: &{SERVICE_UNAVAILABLE}
2020-05-24 22:51:22.724 CDT [channelCmd] InitCmdFactory -> INFO
00b Endorser and orderer connections initialized
2020-05-24 22:51:22.927 CDT [cli.common] readBlock -> INFO 00c
Expect block, but got status: &{SERVICE_UNAVAILABLE}
2020-05-24 22:51:22.936 CDT [channelCmd] InitCmdFactory -> INFO
00d Endorser and orderer connections initialized
2020-05-24 22:51:23.144 CDT [cli.common] readBlock -> INFO 00e
Received block: 0

===================== Channel 'mychannel' created
=====================

Join Org1 peers to the channel...
Using organization 1
+ peer channel join -b ./channel-artifacts/mychannel.block
+ res=0
+ set +x
2020-05-24 22:51:26.358 CDT [channelCmd] InitCmdFactory -> INFO
001 Endorser and orderer connections initialized
2020-05-24 22:51:26.440 CDT [channelCmd] executeJoin -> INFO 002
Successfully submitted proposal to join channel

Join Org2 peers to the channel...
Using organization 2
+ peer channel join -b ./channel-artifacts/mychannel.block
+ res=0
+ set +x
2020-05-24 22:51:29.518 CDT [channelCmd] InitCmdFactory -> INFO
001 Endorser and orderer connections initialized
2020-05-24 22:51:29.820 CDT [channelCmd] executeJoin -> INFO 002
Successfully submitted proposal to join channel

Updating anchor peers for org1...
Using organization 1
+ peer channel update -o localhost:7050 --
ordererTLSHostnameOverride orderer.example.com -c mychannel -
f ./channel-artifacts/Org1MSPanchors.tx --tls true --cafile /
Volumes/WDCWD10JPLX/markmorris/Desktop/Presentation/LAB/fabric-
samples/test-network/organizations/ordererOrganizations/
```



```
example.com/orderers/orderer.example.com/msp/tlscacerts/
tlsca.example.com-cert.pem
+ res=0
+ set +x
2020-05-24 22:51:32.887 CDT [channelCmd] InitCmdFactory -> INFO
001 Endorser and orderer connections initialized
2020-05-24 22:51:32.916 CDT [channelCmd] update -> INFO 002
Successfully submitted channel update
===================== Anchor peers updated for org 'Org1MSP' on
channel 'mychannel' =====================

Updating anchor peers for org2...
Using organization 2
+ peer channel update -o localhost:7050 --
ordererTLSHostnameOverride orderer.example.com -c mychannel -
f ./channel-artifacts/Org2MSPanchors.tx --tls true --cafile /
Volumes/WDCWD10JPLX/markmorris/Desktop/Presentation/LAB/fabric-
samples/test-network/organizations/ordererOrganizations/
example.com/orderers/orderer.example.com/msp/tlscacerts/
tlsca.example.com-cert.pem
+ res=0
+ set +x
2020-05-24 22:51:38.989 CDT [channelCmd] InitCmdFactory -> INFO
001 Endorser and orderer connections initialized
2020-05-24 22:51:39.015 CDT [channelCmd] update -> INFO 002
Successfully submitted channel update
===================== Anchor peers updated for org 'Org2MSP' on
channel 'mychannel' =====================


========= Channel successfully joined ===========

~/Desktop/Presentation/LAB/fabric-samples/test-network
markmorris$
```

Screen Print 4



```
markmorris$ ./network.sh deployCC
deploying chaincode on channel 'mychannel'

Vendoring Go dependencies ...
~/Desktop/Presentation/LAB/fabric-samples/chaincode/fabcar/go ~/
Desktop/Presentation/LAB/fabric-samples/test-network
~/Desktop/Presentation/LAB/fabric-samples/test-network
Finished vendoring Go dependencies
Using organization 1
++ peer lifecycle chaincode package fabcar.tar.gz --path ../
chaincode/fabcar/go/ --lang golang --label fabcar_1
++ res=0
++ set +x
===================== Chaincode is packaged on peer0.org1
=====================

Installing chaincode on peer0.org1...
Using organization 1
++ peer lifecycle chaincode install fabcar.tar.gz
++ res=0
++ set +x
2020-05-24 23:10:24.265 CDT [cli.lifecycle.chaincode]
submitInstallProposal -> INFO 001 Installed remotely:
response:<status:200
payload:"\nIfabcar_1:65710fa851d5c73690faa4709ef40b798c085e7210c
46d44f8b1e2d5a062c9b0\022\010fabcar_1" >
2020-05-24 23:10:24.265 CDT [cli.lifecycle.chaincode]
submitInstallProposal -> INFO 002 Chaincode code package
identifier:
fabcar_1:65710fa851d5c73690faa4709ef40b798c085e7210c46d44f8b1e2d
5a062c9b0
===================== Chaincode is installed on peer0.org1
=====================

Install chaincode on peer0.org2...
Using organization 2
++ peer lifecycle chaincode install fabcar.tar.gz
++ res=0
++ set +x
2020-05-24 23:10:42.925 CDT [cli.lifecycle.chaincode]
submitInstallProposal -> INFO 001 Installed remotely:
response:<status:200
payload:"\nIfabcar_1:65710fa851d5c73690faa4709ef40b798c085e7210c
```

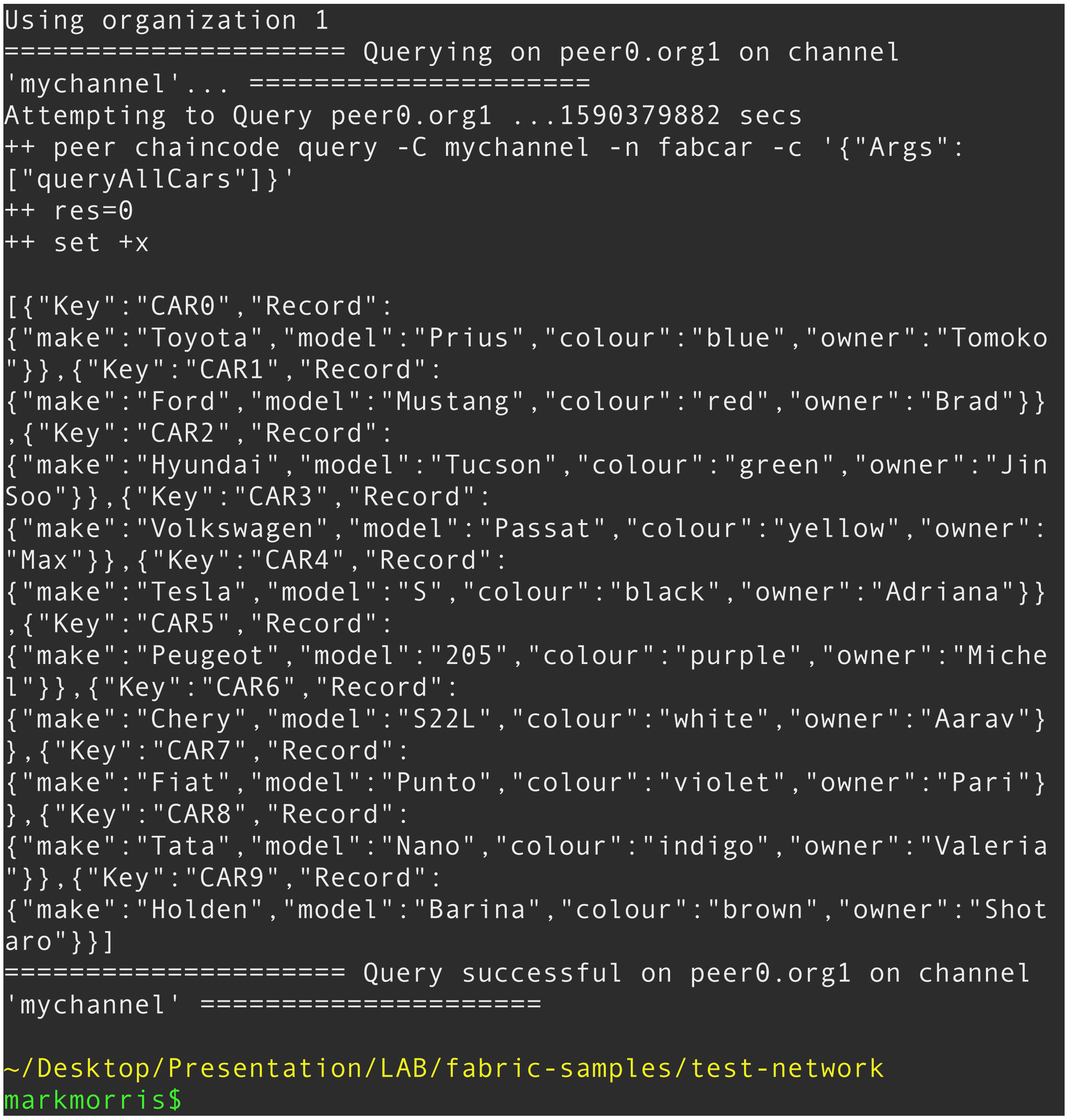
```
46d44f8b1e2d5a062c9b0\022\010fabcar_1" >
2020-05-24 23:10:42.925 CDT [cli.lifecycle.chaincode]
submitInstallProposal -> INFO 002 Chaincode code package
identifier:
fabcar_1:65710fa851d5c73690faa4709ef40b798c085e7210c46d44f8b1e2d
5a062c9b0
===================== Chaincode is installed on peer0.org2
=====================

Using organization 1
++ peer lifecycle chaincode queryinstalled
++ res=0
++ set +x
Installed chaincodes on peer:
Package ID:
fabcar_1:65710fa851d5c73690faa4709ef40b798c085e7210c46d44f8b1e2d
5a062c9b0, Label: fabcar_1
PackageID is
fabcar_1:65710fa851d5c73690faa4709ef40b798c085e7210c46d44f8b1e2d
5a062c9b0
===================== Query installed successful on peer0.org1
on channel =====================

Using organization 1
++ peer lifecycle chaincode approveformyorg -o localhost:7050 --
ordererTLSHostnameOverride orderer.example.com --tls true --
cafile /Volumes/WDCWD10JPLX/markmorris/Desktop/Presentation/LAB/
fabric-samples/test-network/organizations/ordererOrganizations/
example.com/orderers/orderer.example.com/msp/tlscacerts/
tlsca.example.com-cert.pem --channelID mychannel --name fabcar
--version 1 --init-required --package-id
fabcar_1:65710fa851d5c73690faa4709ef40b798c085e7210c46d44f8b1e2d
5a062c9b0 --sequence 1
++ set +x
2020-05-24 23:10:45.132 CDT [chaincodeCmd] ClientWait -> INFO
001 txid
[3017d8f0c706bef3a8a78cd60dfae9e397dcc781a762df22793233555fb69ea
4] committed with status (VALID) at
===================== Chaincode definition approved on
peer0.org1 on channel 'mychannel' =====================

Using organization 1
===================== Checking the commit readiness of the
chaincode definition on peer0.org1 on channel 'mychannel'...
=====================
Attempting to check the commit readiness of the chaincode
```

```
definition on peer0.org1 secs
++ peer lifecycle chaincode checkcommitreadiness --channelID
mychannel --name fabcar --version 1 --sequence 1 --output json
--init-required
++ res=0
++ set +x
{
    "approvals": {
        "Org1MSP": true,
        "Org2MSP": false
    }
}
===================== Checking the commit readiness of the
chaincode definition successful on peer0.org1 on channel
'mychannel' =====================
Using organization 2
===================== Checking the commit readiness of the
chaincode definition on peer0.org2 on channel 'mychannel'...
=====================
Attempting to check the commit readiness of the chaincode
definition on peer0.org2 secs
++ peer lifecycle chaincode checkcommitreadiness --channelID
mychannel --name fabcar --version 1 --sequence 1 --output json
--init-required
++ res=0
++ set +x
{
    "approvals": {
        "Org1MSP": true,
        "Org2MSP": false
    }
}
===================== Checking the commit readiness of the
chaincode definition successful on peer0.org2 on channel
'mychannel' =====================
Using organization 2
++ peer lifecycle chaincode approveformyorg -o localhost:7050 --
ordererTLSHostnameOverride orderer.example.com --tls true --
cafile /Volumes/WDCWD10JPLX/markmorris/Desktop/Presentation/LAB/
fabric-samples/test-network/organizations/ordererOrganizations/
example.com/orderers/orderer.example.com/msp/tlscacerts/
tlsca.example.com-cert.pem --channelID mychannel --name fabcar
--version 1 --init-required --package-id
fabcar_1:65710fa851d5c73690faa4709ef40b798c085e7210c46d44f8b1e2d
5a062c9b0 --sequence 1
++ set +x
```

```
2020-05-24 23:10:53.753 CDT [chaincodeCmd] ClientWait -> INFO
001 txid
[31a487ca3cc8f0af49edab85078890d4a2b16c85536867290d90a16e5d7db05
5] committed with status (VALID) at
===================== Chaincode definition approved on
peer0.org2 on channel 'mychannel' =====================

Using organization 1
===================== Checking the commit readiness of the
chaincode definition on peer0.org1 on channel 'mychannel'...
=====================
Attempting to check the commit readiness of the chaincode
definition on peer0.org1 secs
++ peer lifecycle chaincode checkcommitreadiness --channelID
mychannel --name fabcar --version 1 --sequence 1 --output json
--init-required
++ res=0
++ set +x
{
    "approvals": {
        "Org1MSP": true,
        "Org2MSP": true
    }
}
===================== Checking the commit readiness of the
chaincode definition successful on peer0.org1 on channel
'mychannel' =====================
Using organization 2
===================== Checking the commit readiness of the
chaincode definition on peer0.org2 on channel 'mychannel'...
=====================
Attempting to check the commit readiness of the chaincode
definition on peer0.org2 secs
++ peer lifecycle chaincode checkcommitreadiness --channelID
mychannel --name fabcar --version 1 --sequence 1 --output json
--init-required
++ res=0
++ set +x
{
    "approvals": {
        "Org1MSP": true,
        "Org2MSP": true
    }
}
===================== Checking the commit readiness of the
chaincode definition successful on peer0.org2 on channel
```

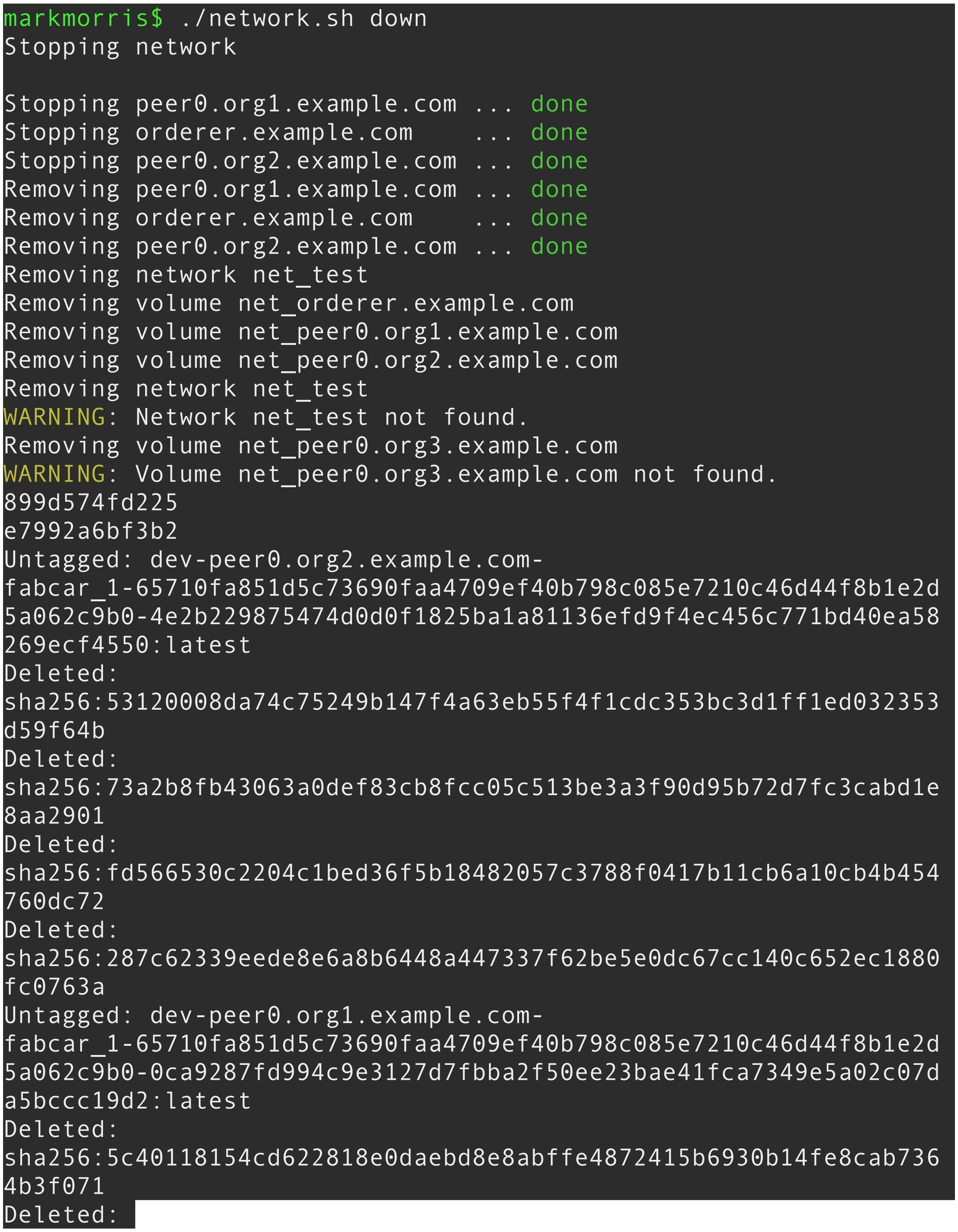
```
'mychannel' ======================
Using organization 1
Using organization 2
++ peer lifecycle chaincode commit -o localhost:7050 --
ordererTLSHostnameOverride orderer.example.com --tls true --
cafile /Volumes/WDCWD10JPLX/markmorris/Desktop/Presentation/LAB/
fabric-samples/test-network/organizations/ordererOrganizations/
example.com/orderers/orderer.example.com/msp/tlscacerts/
tlsca.example.com-cert.pem --channelID mychannel --name fabcar
--peerAddresses localhost:7051 --tlsRootCertFiles /Volumes/
WDCWD10JPLX/markmorris/Desktop/Presentation/LAB/fabric-samples/
test-network/organizations/peerOrganizations/org1.example.com/
peers/peer0.org1.example.com/tls/ca.crt --peerAddresses
localhost:9051 --tlsRootCertFiles /Volumes/WDCWD10JPLX/
markmorris/Desktop/Presentation/LAB/fabric-samples/test-network/
organizations/peerOrganizations/org2.example.com/peers/
peer0.org2.example.com/tls/ca.crt --version 1 --sequence 1 --
init-required
++ res=0
++ set +x
2020-05-24 23:11:02.589 CDT [chaincodeCmd] ClientWait -> INFO
001 txid
[b6db6a85f949a97e0947f6b50830b19f531b732fdeb10b7c913d1f584dad2fb
2] committed with status (VALID) at localhost:7051
2020-05-24 23:11:02.648 CDT [chaincodeCmd] ClientWait -> INFO
002 txid
[b6db6a85f949a97e0947f6b50830b19f531b732fdeb10b7c913d1f584dad2fb
2] committed with status (VALID) at localhost:9051
====================== Chaincode definition committed on channel
'mychannel' ======================

Using organization 1
====================== Querying chaincode definition on
peer0.org1 on channel 'mychannel'... ======================
Attempting to Query committed status on peer0.org1, Retry after
3 seconds.
++ peer lifecycle chaincode querycommitted --channelID mychannel
--name fabcar
++ res=0
++ set +x

Committed chaincode definition for chaincode 'fabcar' on channel
'mychannel':
Version: 1, Sequence: 1, Endorsement Plugin: escc, Validation
Plugin: vscc, Approvals: [Org1MSP: true, Org2MSP: true]
====================== Query chaincode definition successful on
```

```
peer0.org1 on channel 'mychannel' =====================

Using organization 2
===================== Querying chaincode definition on
peer0.org2 on channel 'mychannel'... =====================
Attempting to Query committed status on peer0.org2, Retry after
3 seconds.
++ peer lifecycle chaincode querycommitted --channelID mychannel
--name fabcar
++ res=0
++ set +x

Committed chaincode definition for chaincode 'fabcar' on channel
'mychannel':
Version: 1, Sequence: 1, Endorsement Plugin: escc, Validation
Plugin: vscc, Approvals: [Org1MSP: true, Org2MSP: true]
===================== Query chaincode definition successful on
peer0.org2 on channel 'mychannel' =====================

Using organization 1
Using organization 2
++ peer chaincode invoke -o localhost:7050 --
ordererTLSHostnameOverride orderer.example.com --tls true --
cafile /Volumes/WDCWD10JPLX/markmorris/Desktop/Presentation/LAB/
fabric-samples/test-network/organizations/ordererOrganizations/
example.com/orderers/orderer.example.com/msp/tlscacerts/
tlsca.example.com-cert.pem -C mychannel -n fabcar --
peerAddresses localhost:7051 --tlsRootCertFiles /Volumes/
WDCWD10JPLX/markmorris/Desktop/Presentation/LAB/fabric-samples/
test-network/organizations/peerOrganizations/org1.example.com/
peers/peer0.org1.example.com/tls/ca.crt --peerAddresses
localhost:9051 --tlsRootCertFiles /Volumes/WDCWD10JPLX/
markmorris/Desktop/Presentation/LAB/fabric-samples/test-network/
organizations/peerOrganizations/org2.example.com/peers/
peer0.org2.example.com/tls/ca.crt --isInit -c
'{"function":"initLedger","Args":[]}'
++ res=0
++ set +x
2020-05-24 23:11:08.967 CDT [chaincodeCmd]
chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful.
result: status:200
===================== Invoke transaction successful on
peer0.org1 peer0.org2 on channel 'mychannel'
=====================

Querying chaincode on peer0.org1...
```



```
Using organization 1
===================== Querying on peer0.org1 on channel
'mychannel'... =====================
Attempting to Query peer0.org1 ...1590379882 secs
++ peer chaincode query -C mychannel -n fabcar -c '{"Args":
["queryAllCars"]}'
++ res=0
++ set +x

[{"Key":"CAR0","Record":
{"make":"Toyota","model":"Prius","colour":"blue","owner":"Tomoko
"}},{"Key":"CAR1","Record":
{"make":"Ford","model":"Mustang","colour":"red","owner":"Brad"}}
,{"Key":"CAR2","Record":
{"make":"Hyundai","model":"Tucson","colour":"green","owner":"Jin
Soo"}},{"Key":"CAR3","Record":
{"make":"Volkswagen","model":"Passat","colour":"yellow","owner":
"Max"}},{"Key":"CAR4","Record":
{"make":"Tesla","model":"S","colour":"black","owner":"Adriana"}}
,{"Key":"CAR5","Record":
{"make":"Peugeot","model":"205","colour":"purple","owner":"Miche
l"}},{"Key":"CAR6","Record":
{"make":"Chery","model":"S22L","colour":"white","owner":"Aarav"}
},{"Key":"CAR7","Record":
{"make":"Fiat","model":"Punto","colour":"violet","owner":"Pari"}
},{"Key":"CAR8","Record":
{"make":"Tata","model":"Nano","colour":"indigo","owner":"Valeria
"}},{"Key":"CAR9","Record":
{"make":"Holden","model":"Barina","colour":"brown","owner":"Shot
aro"}}]
===================== Query successful on peer0.org1 on channel
'mychannel' =====================

~/Desktop/Presentation/LAB/fabric-samples/test-network
markmorris$
```

Screen Print 5



```
markmorris$ ./network.sh down
Stopping network

Stopping peer0.org1.example.com ... done
Stopping orderer.example.com    ... done
Stopping peer0.org2.example.com ... done
Removing peer0.org1.example.com ... done
Removing orderer.example.com    ... done
Removing peer0.org2.example.com ... done
Removing network net_test
Removing volume net_orderer.example.com
Removing volume net_peer0.org1.example.com
Removing volume net_peer0.org2.example.com
Removing network net_test
WARNING: Network net_test not found.
Removing volume net_peer0.org3.example.com
WARNING: Volume net_peer0.org3.example.com not found.
899d574fd225
e7992a6bf3b2
Untagged: dev-peer0.org2.example.com-
fabcar_1-65710fa851d5c73690faa4709ef40b798c085e7210c46d44f8b1e2d
5a062c9b0-4e2b229875474d0d0f1825ba1a81136efd9f4ec456c771bd40ea58
269ecf4550:latest
Deleted:
sha256:53120008da74c75249b147f4a63eb55f4f1cdc353bc3d1ff1ed032353
d59f64b
Deleted:
sha256:73a2b8fb43063a0def83cb8fcc05c513be3a3f90d95b72d7fc3cabd1e
8aa2901
Deleted:
sha256:fd566530c2204c1bed36f5b18482057c3788f0417b11cb6a10cb4b454
760dc72
Deleted:
sha256:287c62339eede8e6a8b6448a447337f62be5e0dc67cc140c652ec1880
fc0763a
Untagged: dev-peer0.org1.example.com-
fabcar_1-65710fa851d5c73690faa4709ef40b798c085e7210c46d44f8b1e2d
5a062c9b0-0ca9287fd994c9e3127d7fbba2f50ee23bae41fca7349e5a02c07d
a5bccc19d2:latest
Deleted:
sha256:5c40118154cd622818e0daebd8e8abffe4872415b6930b14fe8cab736
4b3f071
Deleted:
```

```
sha256:25f8a606ea2ca768ca8dbe30cc9afead63e2e8c1250f2bb5c0605d893
3cbb8a9
Deleted:
sha256:7090075389efce64a29068194579b39f98867518630f8dcf515f7b2ba
f8c6fc8
Deleted:
sha256:97aa32c72ab01741347938788d5188389be3a93691509fd5426588cca
90d24ba
~/Desktop/Presentation/LAB/fabric-samples/test-network
markmorris$
```