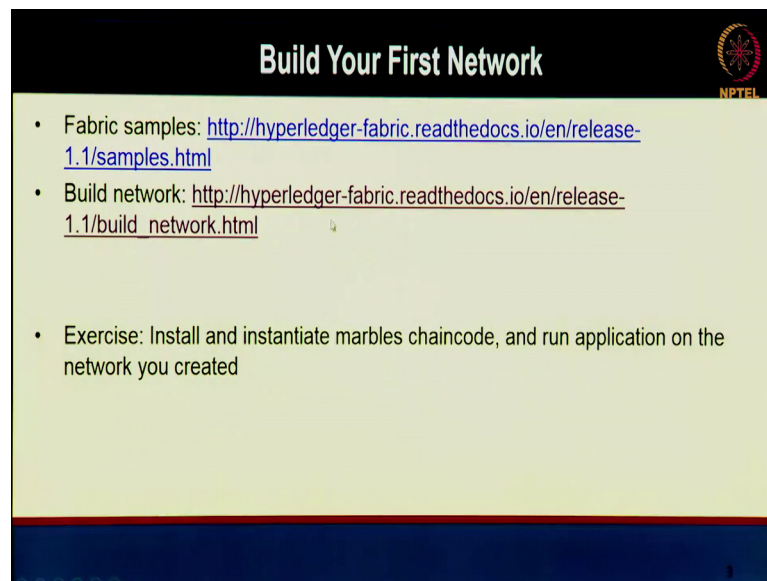


Blockchains Architecture, Design and Use Cases
Prof. Sandip Chakraborty
Prof. Praveen Jayachandran
Department of Computer Science Engineering
Indian Institute of Technology, Kharagpur

Lecture – 27
Fabric Demo, deploy from scratch – III

Hello everyone, good to have you back we are going to be we did a couple of demos on hyperledger fabric using IBM blockchain cloud, but now we are going to look at how you could do it yourself in your own virtual machine for instance or your own laptop, for instance you can you do it wherever you feel like. But I am going to be using Ubuntu vm and ask for certain sort of prerequisites to be installed in the previous lecture. So, let us get started so, a couple of links for how you can build your own network.

(Refer Slide Time: 00:43)

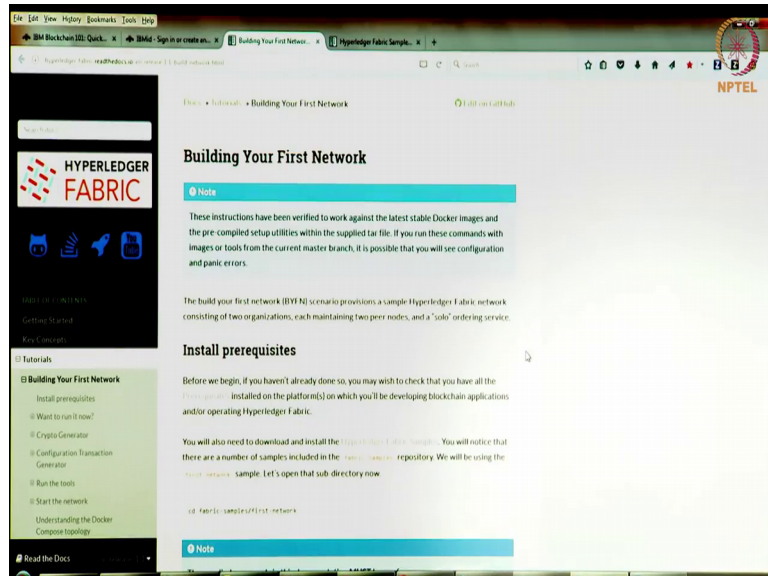


The slide is titled "Build Your First Network" and features the NPTEL logo in the top right corner. It contains a bulleted list of resources and an exercise:

- Fabric samples: <http://hyperledger-fabric.readthedocs.io/en/release-1.1/samples.html>
- Build network: http://hyperledger-fabric.readthedocs.io/en/release-1.1/build_network.html
- Exercise: Install and instantiate marbles chaincode, and run application on the network you created

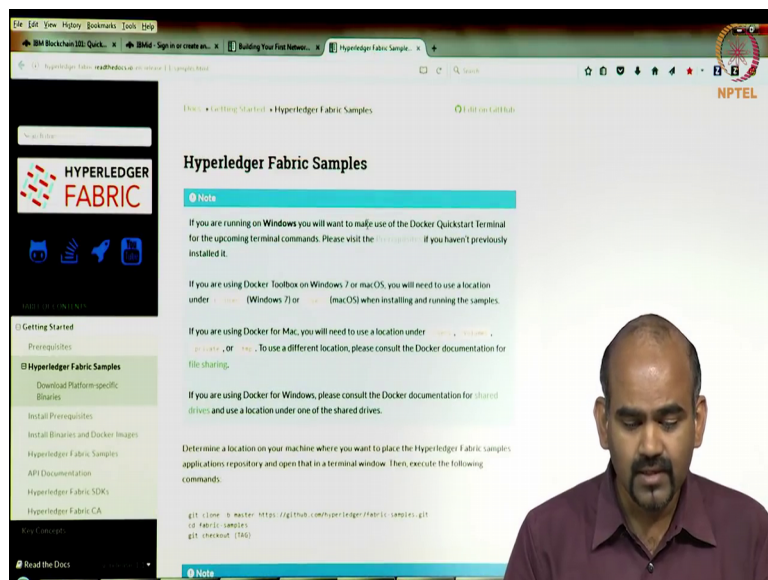
So, we are going to start from that I am going to start from this link on building a network. It once we are done here I would encourage you to try and install an application on top of the network that we are going to build. So, the instance I will show you where the marbles chain code is it and you can go deploy that same chain code on the custom network that you have developed on your or you have built on your virtual machine.

(Refer Slide Time: 01:10)



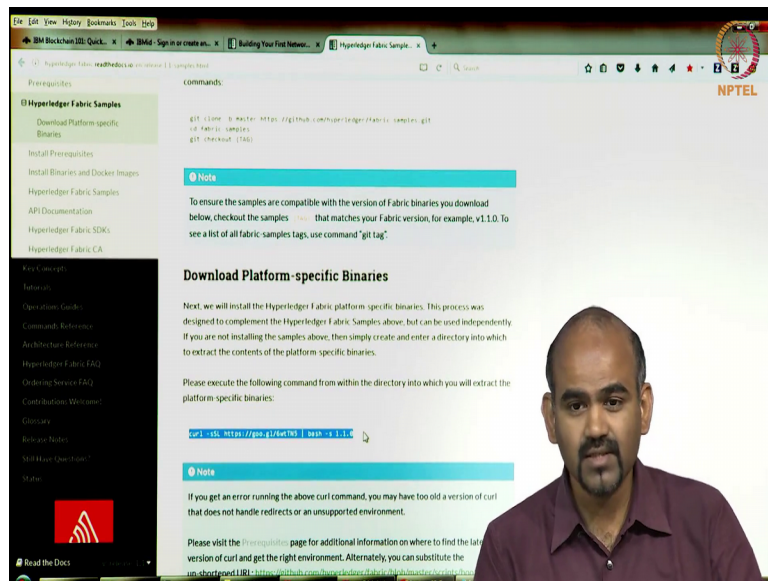
So, let us get there so, let us start with the instructions on how to build your network. So, this is the page that the pointer to. So, first you have to start with the installing some prerequisites, in the prerequisites as a link to the prerequisites you can go that install those that are needed for this for this exercise.

(Refer Slide Time: 01:29)



And there are what I have done as apart from setting of the prerequisites just to save time in the video.

(Refer Slide Time: 01:38)



What have done is, I have also gone ahead and cloned this part this step. So, first you need to go clone piece of code these are the fabric samples. So, we are going to use that as the starting point so, go please clone this GitHub repository and let us get to the fabric samples folder.

So, just follow these steps here, this will what we will do is we will go and pick up a lot of docker containers in the background. So, you will get the peer container, reorder container, the ca container. So, you will get all of those a docker images and it will have that setup in your vm. So, for me took about 10 minutes for this to run, but for you for you it depends on your network right. So, it might be plus or minus of few minutes.

So, apart from once you done those steps where those fabric samples get the binaries as well, actually the docker containers are going to be fetched in the in the binaries. So, this step here you have to go fetch the binaries so, this is the peer order binary. So, this is where actually I will take the 10 minutes sorry about that. So, this getting the clone github itself is just a code so, that will be fast. So, these 2 steps are something you have to run it probably will take you 10 - 12 minutes to do that, but I have already done that for my vm so, we are going to start with that ok.

(Refer Slide Time: 03:09)

```

root@fabric-jp:~/fabric-samples/first-network/channel-artifacts#
Main Menu: 1) Options 2) Facts

root@fabric-jp:~# ls
Applications fabric-samples install_nov.sh Insurance modules network_setup.log nodesource_setup.sh papers post_install.sh
root@fabric-jp:~# cd fabric-samples/
root@fabric-jp:~/fabric-samples# ls
Balance transfer bin channel-decker-downloads fabric-ca first-network LICENSE README.ad
Basic network channelcode config
root@fabric-jp:~/fabric-samples# cd first-network/
root@fabric-jp:~/fabric-samples/first-network# ls
base channel-artifacts crypto-config.yaml docker-compose-couch-org1.yaml docker-compose-e2e-template.yaml org1-artifacts scripts
root@fabric-jp:~/fabric-samples/first-network# ls
base channel-artifacts crypto-config.yaml docker-compose-couch-org1.yaml docker-compose-couch.yaml docker-compose-org1.yaml
root@fabric-jp:~/fabric-samples/first-network# cd ..
root@fabric-jp:~/fabric-samples# ls
Balance transfer bin channel-decker-downloads fabric-ca first-network LICENSE README.ad
Basic network channelcode config
root@fabric-jp:~/fabric-samples# cd channelcode
root@fabric-jp:~/fabric-samples/channelcode# ls
base channel-artifacts crypto-config.yaml docker-compose-couch-org1.yaml docker-compose-couch.yaml docker-compose-org1.yaml
root@fabric-jp:~/fabric-samples/channelcode# cd ../first-network/
root@fabric-jp:~/fabric-samples/first-network# ls
base channel-artifacts crypto-config.yaml docker-compose-couch-org1.yaml docker-compose-e2e-template.yaml org1-artifacts scripts
root@fabric-jp:~/fabric-samples/first-network# cd ..
root@fabric-jp:~/fabric-samples# ls
Balance transfer bin channel-decker-downloads fabric-ca first-network LICENSE README.ad
Basic network channelcode config
root@fabric-jp:~/fabric-samples# cd channelcode
root@fabric-jp:~/fabric-samples/channelcode# cd ../first-network/
root@fabric-jp:~/fabric-samples/first-network# ls
base channel-artifacts crypto-config.yaml docker-compose-couch-org1.yaml docker-compose-couch.yaml docker-compose-org1.yaml
root@fabric-jp:~/fabric-samples/first-network# cd ..
root@fabric-jp:~/fabric-samples# ls
Balance transfer bin channel-decker-downloads fabric-ca first-network LICENSE README.ad
Basic network channelcode config
root@fabric-jp:~/fabric-samples/first-network# cd channel-artifacts/
root@fabric-jp:~/fabric-samples/first-network/channel-artifacts# ls -al
total 8
drwxr-xr-x 2 root root 4096 Apr 10 18:28 .
drwxr-xr-x 6 root root 4096 Apr 13 11:13 ..
-rw-r--r-- 1 root root 0 Apr 10 17:47 gitkeep
root@fabric-jp:~/fabric-samples/first-network/channel-artifacts#

```

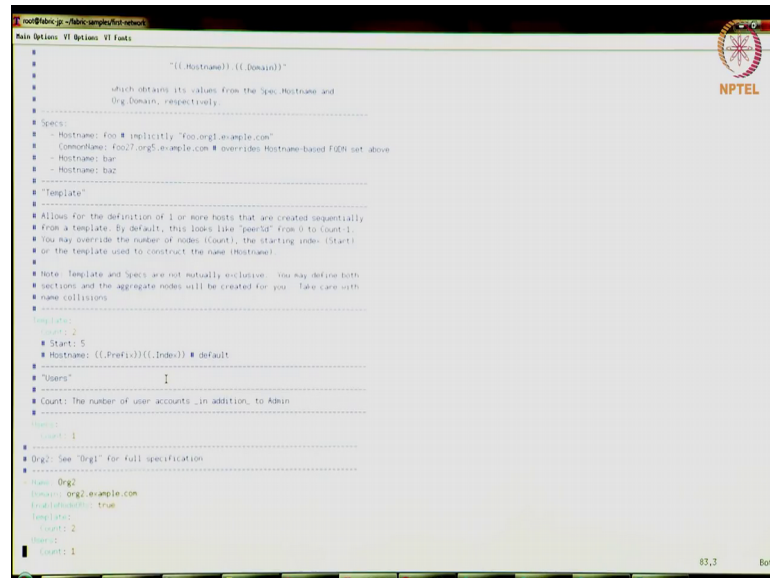
So, let us go back to the network now. So, what are we going to do, what are we let us get to the, this is the command line interface for my virtual machine. So, I am in this fabric samples slash first network. So, I have got let me get back to the fabric samples folder. So, in this there are many folders here, but I am going to use the first network, but there are other examples as well. So, there is a basic network I will give you a different topology will give you certain other aspects of setting up a network.

Let us start with this first network ok. So, first of all before I start let me show that I have a clean state now. So, I am looking at all the docker containers I have on this system there are no docker containers. So, this is the code that I am going to use to setup my network I have already cloned that code and I told you how to do that.

Now, is a next step I will first show you what is the network that we are going to build right. So, this network we are going to configure this in a file called crypto config dot yaml. So, this is a file in your folder I am going to open that file.

And then there it says template count colon 2, what this tells is there are going to be 2 peers belonging to organization. So, there are 2 peers so, count colon 2 belonging to organization 1.

(Refer Slide Time: 05:05)



```
root@libic-jp: ~# cat libic-sample/01-vm-specs
Main Options: VI Options: VI Facts
#
# "((.Hostname)) ((.Domain))"
#
# which obtains its values from the Specs, Hostname and
# Org Domain, respectively.
#-----
# Specs
#   Hostname: foo # implicitly "foo.org1.example.com"
#   CommonName: foo7.org5.example.com # overrides Hostname-based FQDN set above
#   Hostname: bar
#   Hostname: baz
#-----
# Template
#
# Allows for the definition of 1 or more hosts that are created sequentially
# from a template. By default, this looks like "peerS" from 0 to Count-1.
# You may override the number of nodes (Count), the starting index (Start)
# or the template used to construct the name (Hostname).
#
# Note: Template and Specs are not mutually exclusive. You may define both
# sections and the aggregate nodes will be created for you. Take care with
# name collisions.
#-----
Template:
# Count: 2
# Start: 5
# Hostname: ((.Prefix))((.Index)) # default
#-----
Users:
# Count: 1
#-----
Org:
# Org2: See "Org1" for full specification
#-----
Name: Org2
Domain: org2.example.com
Include: true
Template:
# Count: 2
# Start: 0
#-----
Users:
# Count: 1
```

The second one this is going to be an organization 2 and here again we are going to have 2 peers connected to organization 2. So, we are going to have 2 organizations each running 2 peers so, that is going to be your network here.

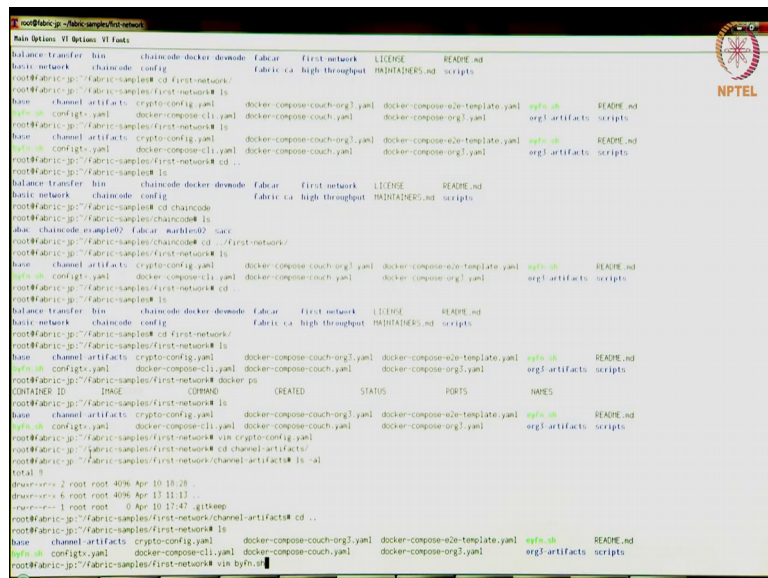
Now, it also has a user count. So, how many users are you are going be created for each organizations so, that is also part of this specification right. So, we will run through this example I will encourage you to also follow along in your virtual machine or your development environment whatever you are using. For go along these step let us set this up, but separately please do feel free to play around with this configurations. So, for instance you can change the number of peers, the number of users you can go play around with some of these things then you will see how the networks get setup.

So, let us get back so, this is the configuration file for the network. So, this needs to be filled out there is a template here that you can use directly that is what we will use right now. So, that is the configuration file and before we start I will also show you what crypto configuration exists. So, right now there is nothing in the network right, no containers, no organizations created. So, there is this folder called channel artifacts right, if you look at that right now it is empty it just has a dot gitkeep file. So, there is nothing

right now in the channel artifacts, there is no channel that is been created, nothing in the configuration for the channel.

There is also another directory that will get created here called crypto config, which will contain all the crypto materials for your network. Basically the orderers certificates, the MSPs, the peers, the CAs, all of those the crypto material for all of those will be in that folder will get created as we proceed. So, I will show you how that gets created and when it gets created.

(Refer Slide Time: 07:11)



```
root@fabric-jp7:/fabric-samples/first-network# ls -la
total 12
drwxr-xr-x 2 root root 4096 Apr 10 18:28 .
drwxr-xr-x 6 root root 4096 Apr 10 11:17 ..
-rw-r--r-- 1 root root  0 Apr 10 17:47 .gitkeep
root@fabric-jp7:/fabric-samples/first-network/channel-artifacts# cd ..
root@fabric-jp7:/fabric-samples/first-network# ls
base channel-artifacts crypto-config.yaml docker-compose-couch-org3.yaml docker-compose-e2e-template.yaml byfn.sh README.md
byfn.sh configtx.yaml docker-compose-cl1.yaml docker-compose-couch.yaml docker-compose-org3.yaml org3-artifacts scripts
root@fabric-jp7:/fabric-samples/first-network# cd ..
root@fabric-jp7:/fabric-samples# ls
balance-transfer bin chaincode docker devmode fabricca first-network LICENSE README.md
basic-network chaincode config fabricca high-throughput MAINTAINERS.md scripts
root@fabric-jp7:/fabric-samples# cd chaincode
root@fabric-jp7:/fabric-samples/chaincode# ls
fabric chaincode-example02 fabric-waiver02 sac
root@fabric-jp7:/fabric-samples/chaincode# cd ../first-network/
root@fabric-jp7:/fabric-samples/first-network# ls
base channel-artifacts crypto-config.yaml docker-compose-couch-org3.yaml docker-compose-e2e-template.yaml byfn.sh README.md
byfn.sh configtx.yaml docker-compose-cl1.yaml docker-compose-couch.yaml docker-compose-org3.yaml org3-artifacts scripts
root@fabric-jp7:/fabric-samples/first-network# cd ..
root@fabric-jp7:/fabric-samples# ls
balance-transfer bin chaincode docker devmode fabricca first-network LICENSE README.md
basic-network chaincode config fabricca high-throughput MAINTAINERS.md scripts
root@fabric-jp7:/fabric-samples/first-network# docker ps
CONTAINER ID        IMAGE               CREATED             STATUS             PORTS             NAMES
root@fabric-jp7:/fabric-samples/first-network# docker ps
base channel-artifacts crypto-config.yaml docker-compose-couch-org3.yaml docker-compose-e2e-template.yaml byfn.sh README.md
byfn.sh configtx.yaml docker-compose-cl1.yaml docker-compose-couch.yaml docker-compose-org3.yaml org3-artifacts scripts
root@fabric-jp7:/fabric-samples/first-network# vim crypto-config.yaml
root@fabric-jp7:/fabric-samples/first-network# cd channel-artifacts/
root@fabric-jp7:/fabric-samples/first-network/channel-artifacts# ls -la
total 3
```

So that is just to show that clean slate from which we are starting. So, there is a do it all script right. So, this is script called build your first network byfn dot sh. So, let us look at what that script really does right. So, this is the script that we are going to open.

(Refer Slide Time: 07:20)

```
root@fabric-jp: ~# cat fabric-sample-first-network
Main Options: VI Options: VI Facts
# /bin/bash
# Copyright IBM Corp. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
#
# This script will orchestrate a sample end-to-end execution of the Hyperledger
# Fabric network.
#
# The end-to-end verification provisions a sample fabric network consisting of
# two organizations, each maintaining two peers, and a "solo" ordering service.
#
# This verification makes use of two fundamental tools, which are necessary to
# create a functioning transactional network with digital signature validation
# and access control:
#
# * cryptogen - generates the .509 certificates used to identify and
#   authenticate the various components in the network.
# * configgen - generates the requisite configuration artifacts for ordering
#   bootstrap and channel creation.
#
# Each tool consumes a configuration yaml file, within which we specify the topology
# of our network (cryptogen) and the location of our certificates for various
# configuration operations (configgen). Once the tools have been successfully run,
# we are able to launch our network. More detail on the tools and the structure of
# the network will be provided later in this document. For now, let's get going...
#
# prepending $PWD/../bin to PATH to ensure we are picking up the correct binaries
# this may be commented out to resolve installed version of tools if desired
export PATH=$PWD/../bin:$PATH
export GOPATH=$PWD
export GOBIN=$PWD

# Print the usage message
function printHelp() {
  echo Usage
  echo byfn.sh up|down|restart|generate|upgrade [-c channel name] [-t timeout] [-d delay] [-f docker-compose file] [-s dbtype] [-i imagetag]
  echo byfn.sh -h|--help print this message
  echo mode = one of up | down | restart | generate
  echo - up - bring up the network with docker-compose up
  echo - down - clear the network with docker-compose down
  echo - restart - restart the network
  echo - generate - generate required certificates and genesis block
  echo - upgrade - upgrade the network from v1.0.x to v1.1
  echo -h|--help - print this message
}

# Typically, one would first generate the required certificates and
# genesis block, then bring up the network. e.g.:
#
# byfn.sh generate -c mychannel
# byfn.sh up -c mychannel -s couchdb -i 1.1.0-alpha
# byfn.sh down -c mychannel
# byfn.sh upgrade -c mychannel
#
# Taking all defaults:
# byfn.sh generate
# byfn.sh up
# byfn.sh down

# Ask user for confirmation to proceed
function askProceed() {
```

So, let us look at some of what some of the functions that it does right.

(Refer Slide Time: 07:27)

```
the network will be provided later in this document. For now, let's get going...
# prepending $PWD/../bin to PATH to ensure we are picking up the correct binaries
# this may be commented out to resolve installed version of tools if desired
export PATH=$PWD/../bin:$PATH
export GOPATH=$PWD
export GOBIN=$PWD

# Print the usage message
function printHelp() {
  echo Usage
  echo byfn.sh up|down|restart|generate|upgrade [-c channel name] [-t timeout] [-d delay] [-f docker-compose file] [-s dbtype] [-i imagetag]
  echo byfn.sh -h|--help print this message
  echo mode = one of up | down | restart | generate
  echo - up - bring up the network with docker-compose up
  echo - down - clear the network with docker-compose down
  echo - restart - restart the network
  echo - generate - generate required certificates and genesis block
  echo - upgrade - upgrade the network from v1.0.x to v1.1
  echo -c channel name - channel name to use (defaults to 'mychannel')
  echo -t timeout - (t) timeout duration in seconds (defaults to 30)
  echo -d delay - delay duration in seconds (defaults to 0)
  echo -f docker-compose file - specify which docker-compose file use (defaults to docker-compose.yml)
  echo -s dbtype - the database backend to use: postgres (default) or couchdb
  echo -i imagetag - the image tag to use (defaults to 'latest')
  echo
  echo Typically, one would first generate the required certificates and
  echo genesis block, then bring up the network. e.g.:
  echo
  echo byfn.sh generate -c mychannel
  echo byfn.sh up -c mychannel -s couchdb
  echo byfn.sh down -c mychannel
  echo byfn.sh upgrade -c mychannel
  echo
  echo Taking all defaults:
  echo byfn.sh generate
  echo byfn.sh up
  echo byfn.sh down

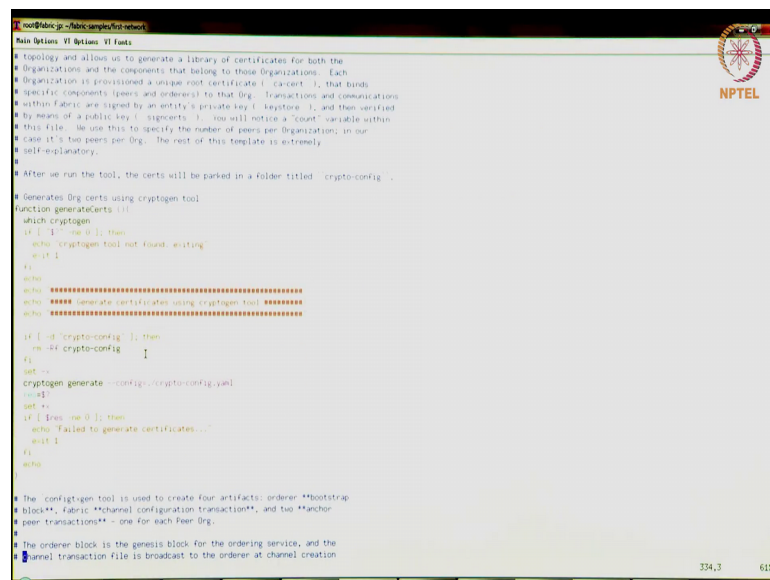
# Ask user for confirmation to proceed
function askProceed() {
```

So, this first print helps this gives you how to use this, it has many functions this going to this generate function is going to generate some of the crypto material needed to setup the network. The byfn dot sh up command is going to bring up the network it is going to create all the docker containers, for the peers, reorderers, the CA is going to bring that up.

And is going to create a channel called mychannel right, we can change that to it is with the minus c parameter, you can bring down the network also byfn dot sh down and it is possible to upgrade the network from one fabric version to another fabric version. So, these are the commands that are that are available and certain set of optional parameters like the channel name, timeout delay, what is the docker compose file and so on right. So, if nothing a specified just default values will be used otherwise you can specify what parameters to use here.

So, let us look at a few of these interesting functions that show what happens right. So, let us go to one of them right. Let us first go to thing called generate let me just jump to that thing you can run through this file later, but I am going to use.

(Refer Slide Time: 08:45)



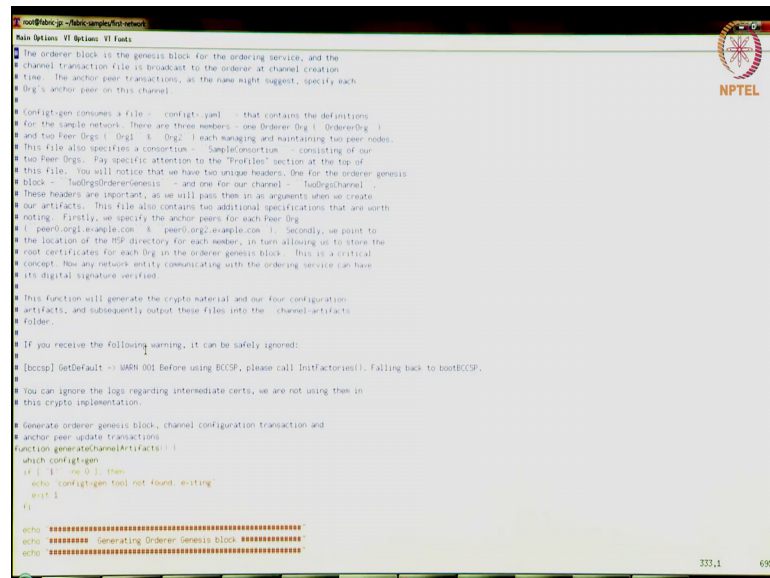
```
root@fabric3g:~/fabric-sample-first-network#
Main Options: VI Options: VI Facts
# topology and allow us to generate a library of certificates for both the
# Organizations and the components that belong to those Organizations. Each
# Organization is provisioned a unique root certificate ( ca-cert ), that binds
# specific components (peers and orderers) to that Org. Transactions and communications
# within Fabric are signed by an entity's private key ( keyfile ), and then verified
# by means of a public key ( sigcerts ). You will notice a "count" variable within
# this file. We use this to specify the number of peers per Organization; in our
# case it's two peers per Org. The rest of this template is extremely
# self-explanatory.
#
# After we run the tool, the certs will be parked in a folder titled " crypto-config ".
# Generates Org certs using cryptogen tool
function generateCerts ()
{
  which cryptogen
  if [ $? -ne 0 ]; then
    echo "cryptogen tool not found, exiting"
    exit 1
  fi
  echo
  echo "***** generate certs using cryptogen tool *****"
  echo "*****"
  if [ -d "crypto-config" ]; then
    rm -rf crypto-config
  fi
  set -x
  cryptogen generate --config ./crypto-config.yaml
  set +x
  if [ $? -ne 0 ]; then
    echo "Failed to generate certificates..."
    exit 1
  fi
  echo
}

# The configgen tool is used to create four artifacts: orderer **bootstrap
# block**, fabric **channel configuration transaction**, and two **anchor
# peer transactions** - one for each Peer Org.
#
# The orderer block is the genesis block for the ordering service, and the
# channel transaction file is broadcast to the orderer at channel creation
```

So, this is file called generatecerts. So, this sorry this function called generatecerts what it is going to do is, it is going to generate a certificate for a particular component or a user right. So, that is peer certificates, clone certificates so, user certificates or orderers certificates, generatecerts is going to do that.

And there is a cryptogen tool that I have talked about in of the lectures. So, this is a tool that is provided with fabric it is a convenience tool for you to generate some of these certificates these x 5 or 9 certificates for you right. You can also do this without cryptogen using of fabric CA using a fabric CA to generate these certificates, but in this example we are going to use this cryptogen tool.

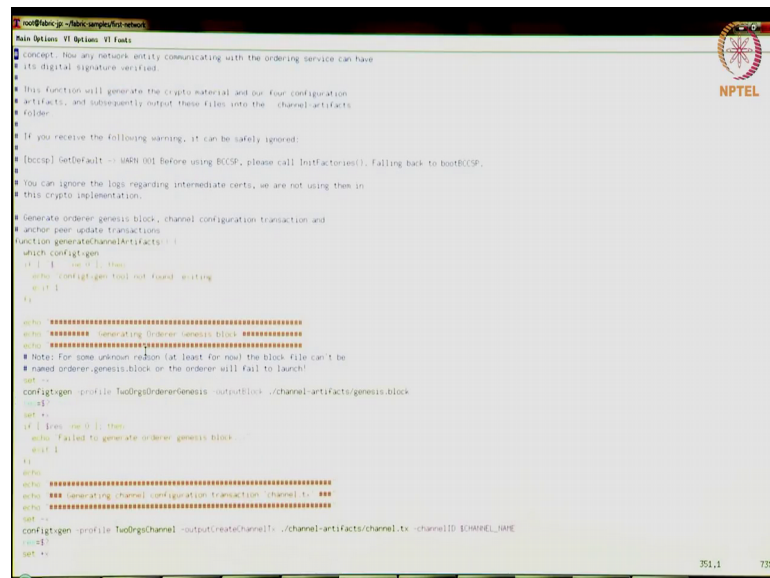
(Refer Slide Time: 09:32)



```
root@fabric-jp: ~# fabric-sample-first-network
Main Options VI Options VI Facts
# The orderer block is the genesis block for the ordering service, and the
# channel transaction file is broadcast to the orderer at channel creation
# time. The anchor peer transactions, as the name might suggest, specify each
# Org's anchor peer on this channel.
#
# Configgen consumes a file ( configgen.yaml ) that contains the definitions
# for the sample network. There are three members - one Orderer Org ( OrderOrg )
# and two Peer Orgs ( Org1 & Org2 ) each managing and maintaining two peer nodes.
# This file also specifies a consortium - SampleConsortium - consisting of our
# two Peer Orgs. Pay specific attention to the "Profiles" section at the top of
# this file. You will notice that we have two unique headers, one for the orderer genesis
# block - TwoOrgsOrdererGenesis - and one for our channel - TwoOrgsChannel .
# These headers are important, as we will pass them in as arguments when we create
# our artifacts. This file also contains two additional specifications that are worth
# noting. Firstly, we specify the anchor peers for each Peer Org
# ( peer0.org1.example.com & peer0.org2.example.com ). Secondly, we point to
# the location of the MSP directory for each member, in turn allowing us to store the
# root certificates for each Org in the orderer genesis block. This is a critical
# concept. Now any network entity communicating with the ordering service can have
# its digital signature verified.
#
# This function will generate the crypto material and our four configuration
# artifacts, and subsequently output these files into the channel-artifacts
# folder.
#
# If you receive the following warning, it can be safely ignored:
# [bccsp] GetDefault -> WARN 001 Before using BCSP, please call InitFactories(). Falling back to bootBCSP.
# You can ignore the logs regarding intermediate certs, we are not using them in
# this crypto implementation.
#
# Generate orderer genesis block, channel configuration transaction and
# anchor peer update transactions
function generateChannelArtifacts() {
  which configgen
  if [ $? -ne 0 ]; then
    echo "configgen tool not found - exiting"
    exit 1
  fi
  echo "=====
  echo "Generating Orderer Genesis block"
  echo "=====
  configgen -profile TwoOrgsOrdererGenesis -outputfiles ./channel-artifacts/genesis.block
  exit 1
  echo "=====
  echo "Generating Channel Configuration Transaction"
  echo "=====
  configgen -profile TwoOrgsChannel -outputcreateschannel ./channel-artifacts/channel.tx -channelID CHANNEL_NAME
  exit 1
}
```

Generates channel artifacts by the way there are lot of extensive comments here that you see it all explains what happening in each function, how it happens step by step. The generate channel artifacts is going to generate basically the genesis block and the channel configuration right.

(Refer Slide Time: 09:50)



```
root@fabric-jp: ~# fabric-sample-first-network
Main Options VI Options VI Facts
# concept. Now any network entity communicating with the ordering service can have
# its digital signature verified.
#
# This function will generate the crypto material and our four configuration
# artifacts, and subsequently output these files into the channel-artifacts
# folder.
#
# If you receive the following warning, it can be safely ignored:
# [bccsp] GetDefault -> WARN 001 Before using BCSP, please call InitFactories(). Falling back to bootBCSP.
# You can ignore the logs regarding intermediate certs, we are not using them in
# this crypto implementation.
#
# Generate orderer genesis block, channel configuration transaction and
# anchor peer update transactions
function generateChannelArtifacts() {
  which configgen
  if [ $? -ne 0 ]; then
    echo "configgen tool not found - exiting"
    exit 1
  fi
  echo "=====
  echo "Generating Orderer Genesis block"
  echo "=====
  # Note: For some unknown reason (at least for now) the block file can't be
  # named orderer-genesis.block or the orderer will fail to launch!
  configgen -profile TwoOrgsOrdererGenesis -outputfiles ./channel-artifacts/genesis.block
  exit 1
  echo "=====
  echo "Generating Channel Configuration Transaction"
  echo "=====
  configgen -profile TwoOrgsChannel -outputcreateschannel ./channel-artifacts/channel.tx -channelID CHANNEL_NAME
  exit 1
}
```

So, maybe we will come back to the explaining that step.

(Refer Slide Time: 10:00)

```
root@fabric-gp-fabric-sample-test-network:~# cat scripts/script.sh
#!/bin/bash
set -e

# Print help
printHelp() {
  echo "Usage: script.sh [options]"
  echo "Options:"
  echo "  -c CHANNEL_NAME: Channel name (required)"
  echo "  -t TIMEOUT: Timeout in seconds (default: 300)"
  echo "  -d DELAY: Delay in seconds (default: 10)"
  echo "  -f FABRIC_VERSION: Fabric version (default: 1.0.0)"
  echo "  -s SCRIPT_PATH: Script path (default: ./)"
  echo "  -i IMAGE_PATH: Image path (default: ./)"
  echo "  -h: Help"
}

# Announce what was requested
if [ "$1" = "-h" ]; then
  printHelp
else
  echo "IE:PRDEE with channel: '$CHANNEL_NAME' and CLI timeout of '$(($TIMEOUT))' seconds and CLI delay of '$(($DELAY))' seconds and using database '$(($COCHDB))'"
  echo "IE:PRDEE with channel: '$CHANNEL_NAME' and CLI timeout of '$(($TIMEOUT))' seconds and CLI delay of '$(($DELAY))' seconds"
fi

# Ask for confirmation to proceed
askProceed() {
  read -p "Press [Y] to proceed or [N] to abort: " response
  if [ "$response" != "Y" ]; then
    exit 1
  fi
}

# Create the network using docker compose
networkUp() {
  networkDown
  generateArtifacts
  replacePrivateKey
  generateChannelArtifacts
  restartNetwork
  networkDown
}

function main() {
  channelName=$(getChannelName)
  timeout=$(getTimeout)
  delay=$(getDelay)
  fabricVersion=$(getFabricVersion)
  scriptPath=$(getScriptPath)
  imagePath=$(getImagePath)

  askProceed

  networkUp

  echo "Network up successfully"
}

main
```

And then there is a function called networkup.

(Refer Slide Time: 10:09)

```
root@fabric-gp-fabric-sample-test-network:~# cat scripts/script.sh
#!/bin/bash
set -e

# Print help
printHelp() {
  echo "Usage: script.sh [options]"
  echo "Options:"
  echo "  -c CHANNEL_NAME: Channel name (required)"
  echo "  -t TIMEOUT: Timeout in seconds (default: 300)"
  echo "  -d DELAY: Delay in seconds (default: 10)"
  echo "  -f FABRIC_VERSION: Fabric version (default: 1.0.0)"
  echo "  -s SCRIPT_PATH: Script path (default: ./)"
  echo "  -i IMAGE_PATH: Image path (default: ./)"
  echo "  -h: Help"
}

# Announce what was requested
if [ "$1" = "-h" ]; then
  printHelp
else
  echo "IE:PRDEE with channel: '$CHANNEL_NAME' and CLI timeout of '$(($TIMEOUT))' seconds and CLI delay of '$(($DELAY))' seconds and using database '$(($COCHDB))'"
  echo "IE:PRDEE with channel: '$CHANNEL_NAME' and CLI timeout of '$(($TIMEOUT))' seconds and CLI delay of '$(($DELAY))' seconds"
fi

# Ask for confirmation to proceed
askProceed() {
  read -p "Press [Y] to proceed or [N] to abort: " response
  if [ "$response" != "Y" ]; then
    exit 1
  fi
}

# Create the network using docker compose
networkUp() {
  networkDown
  generateArtifacts
  replacePrivateKey
  generateChannelArtifacts
  restartNetwork
  networkDown
}

function main() {
  channelName=$(getChannelName)
  timeout=$(getTimeout)
  delay=$(getDelay)
  fabricVersion=$(getFabricVersion)
  scriptPath=$(getScriptPath)
  imagePath=$(getImagePath)

  askProceed

  networkUp

  echo "Network up successfully"
}

main
```

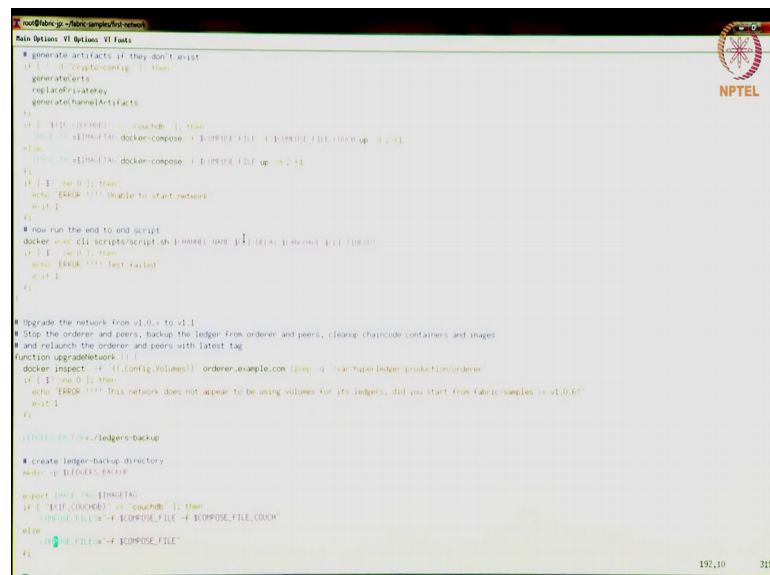
So, this is that function so, this is the function this main function this is going to bring up the network, once all the crypto material been generated all the identities been created for the peer orderer client and so on.

The network up function is going to bring up all the docker containers running all of these entities. So, by the way we are going to have just 1 vm right now running all of these components, but eventually think about the way blockchain would actually be set up is maybe each organization would probably run their own server running those

components may even be a distributed systems like the peer may be on a different server, from the certificate authority and so on.

And so, eventually this will be a completely distributed system, but just for demonstration purposes making it easy we just going to do it as docker containers within a single VM. There are also instructions on how you can set up a multi VM environment for like think of it as multiple VMs for each organization that is also possible the instructions for that and I would encourage you to look at that later, there are links from some of the ones pointers that I have given myself.

(Refer Slide Time: 11:19)



```
root@fabric-gp:~/fabric-sample/network#
Main Options: 11:19:05 AM
# generate artifacts if they don't exist
if [ -d artifacts ]; then
  generateArtifacts
  explainPostkey
  generateChannelArtifacts
fi

if [ "$1" == "start" ]; then
  # start the network
  docker-compose -f $COMPOSE_FILE -f $COMPOSE_FILE_COCH up -d
else
  docker-compose -f $COMPOSE_FILE -f $COMPOSE_FILE_COCH up -d
fi

if [ "$1" == "stop" ]; then
  echo "ERROR: unable to start network"
  exit 1
fi

# now run the end to end script
docker-compose -f scripts/script.sh --network $1 --scale $2 --scale $3 --scale $4
if [ "$1" == "0" ]; then
  echo "ERROR: test failed"
  exit 1
fi

# Upgrade the network from v1.0.3 to v1.1
# Stop the orderer and peers, backup the ledger from orderer and peers, cleanup chaincode containers and images
# and relaunch the orderer and peers with latest tag
function upgradeNetwork() {
  docker inspect -f '{{.Config.Volumes}}' orderer.example.com | grep -q 'hyperledger-production/orderer'
  if [ "$?" == "0" ]; then
    echo "ERROR: this network does not appear to be using volumes for its ledgers, did you start from fabric-sample v1.0.3?"
    exit 1
  fi
}

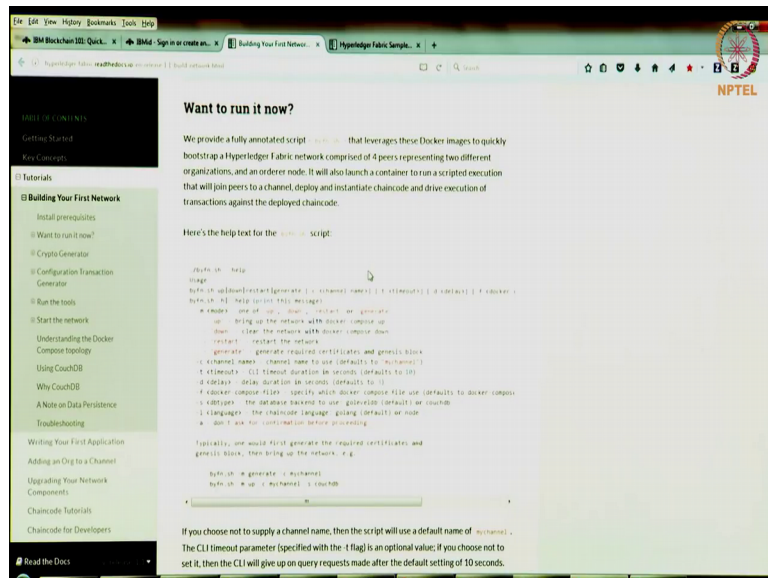
# create ledger-backup directory
mkdir -p $LEDGER_BACKUP

export COMPOSE_FILE=$COMPOSE_FILE
if [ "$1" == "start" ]; then
  COMPOSE_FILE="$COMPOSE_FILE" -f $COMPOSE_FILE_COCH
else
  COMPOSE_FILE="$COMPOSE_FILE"
fi
192.10 31%
```

So, this network up function is the one that is going to bring up the, it is going to call docker compose. So, there is docker compose file has all the components and the it has pointers to the docker images to be brought up. So, based on what you have configured you will be able to you will be bringing up that many peers, orderers and so on so, that is what is happening in the networkup function.

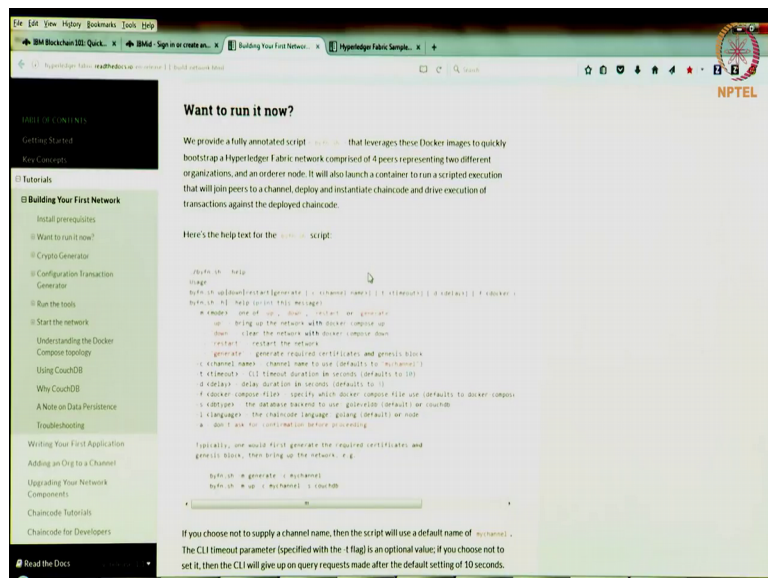
So, let us head back out of this thing, there are many other functions that you can look at, but let us head back, I will go back to the to the html file gives you the overview of how to build your network.

(Refer Slide Time: 12:01)



So, going back to some of these things right so, what are some of the things we are doing we are going to be generating certificates first.

(Refer Slide Time: 12:06)



Then we are going to generate the network artifacts the channel aspects where we are going to create the orderer genesis block and then the channel configuration and then we are going to add peers to that channel. So, this is how this is what we are going to be doing. So, let us do that.

(Refer Slide Time: 12:23)

```
root@fabric-jg:~/fabric-samples/first-network# ls
base channel artifacts crypto-config.yaml docker-compose-couch-org1.yaml docker-compose-c2e-template.yaml org1.example.com README.md
root@fabric-jg:~/fabric-samples/first-network# ./bin/sh generate
Generating certs and genesis block for with channel 'mychannel' and (LI timeout of '10' seconds and (LI delay of '3' seconds
continue) [Y/n] y
proceeding ...
root@fabric-samples/first-network# ./bin/cryptogen
#####
##### Generate certificates using cryptogen tool #####
#####
```

So, let me clear, now the first thing I am going to do is generate the identities for the different components. So, this is using the generate function generate command. So, I am going to run that it will ask you a prompt yes or no, you can hit yes.

(Refer Slide Time: 12:41)

```
root@fabric-jg:~/fabric-samples/first-network# ./bin/cryptogen
#####
##### Generate certificates using cryptogen tool #####
#####
+ cryptogen generate --config ./crypto-config.yaml
org1.example.com
+ resp0
+ set +x

root@fabric-samples/first-network# ./bin/configgen
#####
##### Generating channel configuration transaction 'channel.tx' #####
+ configgen -profile twoOrgsOrdererGenesis -outputBlock ./channel-artifacts/genesis.block
2018-04-13 11:20:14.823 UTC [common/tools/configgen] main -> INFO 001 Loading configuration
2018-04-13 11:20:14.841 UTC [msg] getMspConfig -> INFO 002 Loading HsB00s
2018-04-13 11:20:14.842 UTC [msg] getMspConfig -> INFO 003 Loading HsB00s
2018-04-13 11:20:14.842 UTC [common/tools/configgen] doOutputBlock -> INFO 004 Generating genesis block
2018-04-13 11:20:14.842 UTC [common/tools/configgen] doOutputBlock -> INFO 005 Writing genesis block
+ resp0
+ set +x

#####
##### Generating channel configuration transaction 'channel.tx' #####
+ configgen -profile twoOrgsChannel -outputChannelTx ./channel-artifacts/channel.tx -channelID mychannel
2018-04-13 11:20:14.861 UTC [common/tools/configgen] main -> INFO 001 Loading configuration
2018-04-13 11:20:14.885 UTC [common/tools/configgen] doOutputChannelCreate -> INFO 002 Generating new channel config
2018-04-13 11:20:14.885 UTC [msg] getMspConfig -> INFO 003 Loading HsB00s
2018-04-13 11:20:14.894 UTC [msg] getMspConfig -> INFO 004 Loading HsB00s
2018-04-13 11:20:14.911 UTC [common/tools/configgen] doOutputChannelCreate -> INFO 005 Writing new channel tx
+ resp0
+ set +x

#####
##### Generating anchor peer update for Org1MSP #####
```

And it has now generated all the identities. So, let us go over what it is done. So, we have used the cryptogen tool right, it is created an orderer genesis block. It is created a channel configuration transaction called channel dot tx and this is now initialize the channel is now created.

(Refer Slide Time: 13:05)

```
root@fabric-jp:~/fabric-samples/first-network# ./cryptogen
Main Options: -f /optima -t /facts

2018-04-13 11:20:14.823 UTC [common/tools/configtgen] main -- INFO 001 Loading configuration
2018-04-13 11:20:14.841 UTC [esp] getHqsConfig -- INFO 002 Loading Node0p
2018-04-13 11:20:14.842 UTC [esp] getHqsConfig -- INFO 003 Loading Node0s
2018-04-13 11:20:14.842 UTC [common/tools/configtgen] doOutputBlock -- INFO 004 Generating genesis block
2018-04-13 11:20:14.842 UTC [common/tools/configtgen] doOutputBlock -- INFO 005 Writing genesis block
+ set +*

##### Generating channel configuration transaction 'channel.tx' #####
+ configtgen -profile fudOrgsChannel -output=channel.tx -channel-artifacts/channel.tx -channelID mychannel
2018-04-13 11:20:14.867 UTC [common/tools/configtgen] main -- INFO 001 Loading configuration
2018-04-13 11:20:14.883 UTC [common/tools/configtgen] doOutputChannelCreate -- INFO 002 Generating new channel config
2018-04-13 11:20:14.883 UTC [esp] getHqsConfig -- INFO 003 Loading Node0p
2018-04-13 11:20:14.884 UTC [esp] getHqsConfig -- INFO 004 Loading Node0s
2018-04-13 11:20:14.911 UTC [common/tools/configtgen] doOutputChannelCreate -- INFO 005 Writing new channel tx
+ set +*

##### Generating anchor peer update for Org1MSP #####
+ configtgen -profile fudOrgsChannel -output=AnchorPeerUpdate -channel-artifacts/Org1MSPanchors.tx -channelID mychannel -adding Org1MSP
2018-04-13 11:20:14.935 UTC [common/tools/configtgen] main -- INFO 001 Loading configuration
2018-04-13 11:20:14.951 UTC [common/tools/configtgen] doOutputAnchorPeerUpdate -- INFO 002 Generating anchor peer update
2018-04-13 11:20:14.951 UTC [common/tools/configtgen] doOutputAnchorPeerUpdate -- INFO 003 Writing anchor peer update
+ set +*

##### Generating anchor peer update for Org2MSP #####
+ configtgen -profile fudOrgsChannel -output=AnchorPeerUpdate -channel-artifacts/Org2MSPanchors.tx -channelID mychannel -adding Org2MSP
2018-04-13 11:20:14.978 UTC [common/tools/configtgen] main -- INFO 001 Loading configuration
2018-04-13 11:20:14.993 UTC [common/tools/configtgen] doOutputAnchorPeerUpdate -- INFO 002 Generating anchor peer update
2018-04-13 11:20:14.994 UTC [common/tools/configtgen] doOutputAnchorPeerUpdate -- INFO 003 Writing anchor peer update
+ set +*

root@fabric-jp:~/fabric-samples/first-network# ls
base channel-artifacts crypto-config docker-compose-c11.yaml docker-compose-couch.yaml docker-compose-e2e.yaml myfa.sh README.md
myfa.sh configtx.yaml crypto-config.yaml docker-compose-couch-org3.yaml docker-compose-e2e-template.yaml docker-compose-org2.yaml org3-artifacts scripts
root@fabric-jp:~/fabric-samples/first-network# cd crypto-config/
root@fabric-jp:~/fabric-samples/first-network/crypto-config#
```

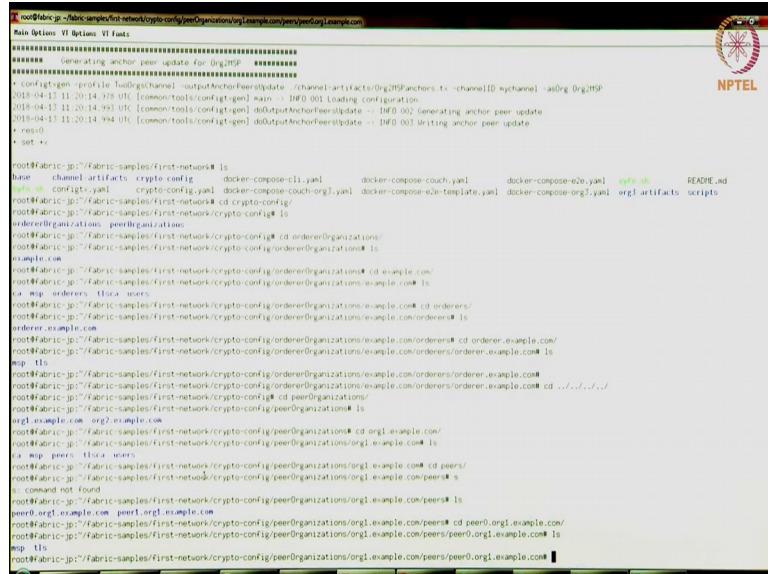
And then there is there are 2 anchor peers one for each organizations Org1MSP and Org2MSP all both have one anchor peer which has been added to the channel and the way it is structured is only the anchor peer is going to be talking to the ordering service. The second peer in each organization will just talk to the first peer and obtain that information and that is because it is both of those peers are within the same (Refer Time:13:32) boundary of the organization.

So, the second peer can trust the first peer, but the each peer in the 2 organizations will not trust each other they will both independently get the blocks from the orderer and will verify the orderer signature right. So, all the identities are set up the channel is set up the peers have been set up. So, that is what is happen so far.

So, while we added so, the other thing to notice here is now the crypto config folder has been created. So, this is where all the identities are currently stored right now and again a word of caution right now you stored all these identities on file system. It may not be the best idea to do this it is probably a good idea to put this inside let us say hardware security module. If you are drawing this in production should really be storing these identities inside a crypto module in hardware so, where it gets does not get stolen right.

You could also (Refer Time: 14:32) software for encrypted in a database. So, all those are options, but for now for the simple example just to make things easier of you to get started as a developer we just put it on the file system.

(Refer Slide Time: 14:52)

The image shows a terminal window with a dark background and a window title bar that reads "root@fabric-jp: ~ - fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com". The terminal output shows the execution of the 'crypto-config-generateanchors' command. It displays the generation of an anchor peer update for Org2SP and the output of 'configgen-profile' which lists various configuration files like 'crypto-config.yaml', 'docker-compose-couch.yaml', and 'README.md'. Subsequent commands show directory navigation: 'cd crypto-config/', 'cd ordererOrganizations/', and 'cd example.com/'. Finally, the 'ls' command is used to list the contents of several directories, including 'orderers', 'peerOrganizations', and 'example.com/orderers', demonstrating the hierarchical structure of the configuration files.

Let us look at the crypto config folder. So, it has orderer organizations and it has peerorganization. So, just for you to just to give you feel for what this looks like you just go look at the folder structure, within the orderer organization example dot com is the orderer that you created and within that it has the ca msp orderers, the tls certificates and users.

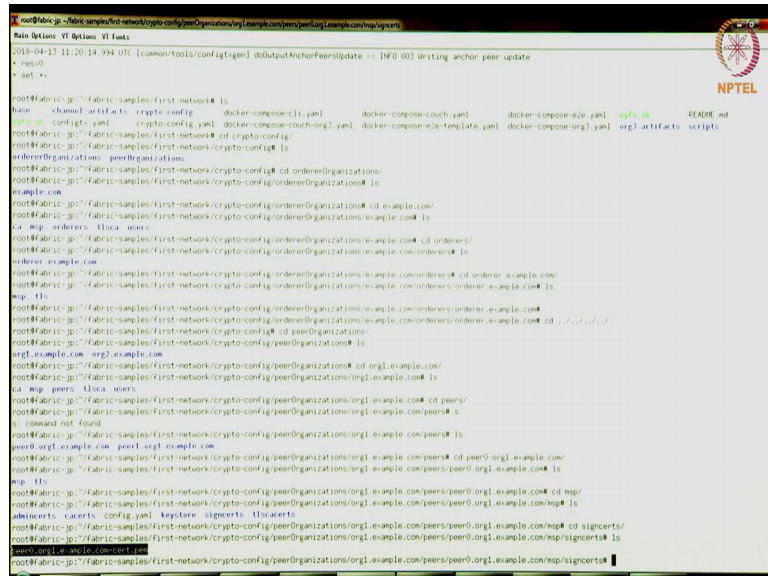
So, it has folders for each of these we could go inside let us say let us go into orderers, it has orderer dot example dot com and it will show you that msp and the tls certificates for the orderer right. So, likewise let us go back a few steps, you can go to peerorganizations and you will see there is org 1 dot example dot com and org 2 dot example dot com let us go into one of them.

So, org 1 dot example dot com again has the same folders right same folder structure this ca msp peers inside of the orderers you know have the peers folder and then again the tls certificates for network communication and the users for that organization. So, if you go into peers sorry into going to peers for this peer we are going to for this organization we are creating 2 peers, peer 0 dot org 1 dot example dot com and peer 1 dot org 1 dot example dot com.

So, again if you go into one of the peers, let us go into the msp of that peer this has for instance now the sign certificates. So, this will be where the private key for the peers. So, signcerts is there so, this if you see is the private key that the peer will use to sign this or

sign whatever messages it is sending out in the network. So, maybe we can also open and view what this looks like.

(Refer Slide Time: 17:12)



```
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com# ./generateCerts.sh
Main Options: VI Options: VI Fast
2018-04-13 11:20:14.994 UTC [common/tools/configgen] doOutputAnchorPeerUpdate --> INFO: 003 Writing anchor peer update
+ root
+ set +x
root@fabric-jp7:~/fabric-samples/first-network# ls
base          channel-artifacts  crypto-config      docker-compose-1.1.yml  docker-compose-couch.yml  docker-compose-e2e.yml  npx             README.md
byfn          config.yml         crypto-config.yml  docker-compose-couch-org1.yml  docker-compose-e2e-template.yml  docker-compose-org1.yml  org1-artifacts  scripts
root@fabric-jp7:~/fabric-samples/first-network# cd crypto-config/
root@fabric-jp7:~/fabric-samples/first-network/crypto-config# ls
ordererOrganizations  peersOrganizations
root@fabric-jp7:~/fabric-samples/first-network/crypto-config# cd ordererOrganizations/
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/ordererOrganizations# ls
example.com
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/ordererOrganizations# cd example.com/
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/ordererOrganizations/example.com# ls
ca  msp  orderers  tlsca  users
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/ordererOrganizations/example.com# cd orderers/
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/ordererOrganizations/example.com/orderers# ls
orderer.example.com
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/ordererOrganizations/example.com/orderers# cd orderer.example.com/
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com# ls
msp  tls
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com# cd ../../../../../../
root@fabric-jp7:~/fabric-samples/first-network/crypto-config# cd peerOrganizations/
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/peerOrganizations# ls
org1.example.com  org2.example.com
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/peerOrganizations# cd org1.example.com/
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com# ls
ca  msp  peers  tlsca  users
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com# cd peers/
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/peers# ls
command-not-found
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/peers# cd peer0.org1.example.com/
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com# ls
msp
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com# cd msp/
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp# ls
anchors  certs  config.yml  keysetke  signcerts  tlsanchors
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp# cd signcerts/
root@fabric-jp7:~/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp/signcerts# ls
peer0.org1.example.com/signcerts/
```

So, this is the certificate that it is using to sign right, you can also see some of the other things for instance you can go `tls ca certs` and `caerts`.

So, if you go to `caerts` this has the org one ca certificate right. So, all of these certificates have been generated and have been stored inside the crypto config folder. So, let us head back right. So, I have shown you where the certificates reside now I will do a `docker ps` again now there are no containers after this point, we only created the identities we have created the genesis block, but there is nothing else. So, genesis block is just the data structure right nothing else.

So, next up let us bring up the network. So, we have we are; what we are going to bring up is 2 peers, 1 orderer, 2 cas for each of the organizations and the that is what you are going to bring up now. So, let us do that so, we are going to do `byfn dot sh up`. So, this command is going to bring up the network is going to do a bunch of things again a prompt press yes.

(Refer Slide Time: 18:03)

```

root@fabric-j3:~/fabric-samples/first-network#
Main Options VI Options VI Facts
root@fabric-j3:~/fabric-samples/first-network# crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/
root@fabric-j3:~/fabric-samples/first-network# crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/ls
my:
  11:
root@fabric-j3:~/fabric-samples/first-network# crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com# cd /sp/
root@fabric-j3:~/fabric-samples/first-network# keytool -sigcerts -listcerts
root@fabric-j3:~/fabric-samples/first-network# crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp# cd /
root@fabric-j3:~/fabric-samples/first-network# crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp# ls
peer0.org1.example.com-cert.pem
root@fabric-j3:~/fabric-samples/first-network# crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp# cp cert.pem
root@fabric-j3:~/fabric-samples/first-network# crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp# cp cert.pem
root@fabric-j3:~/fabric-samples/first-network# keytool -sigcerts -listcerts
root@fabric-j3:~/fabric-samples/first-network# crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp# cd /ca/certs/
root@fabric-j3:~/fabric-samples/first-network# crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp# ls
ca.org1.example.com-cert.pem
root@fabric-j3:~/fabric-samples/first-network# crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp# cp ca-cert.pem
root@fabric-j3:~/fabric-samples/first-network# crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp# cd /..
root@fabric-j3:~/fabric-samples/first-network# crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp# cd /..
root@fabric-j3:~/fabric-samples/first-network# cd /..
root@fabric-j3:~/fabric-samples/first-network# cd /..
root@fabric-j3:~/fabric-samples/first-network# cd /..
root@fabric-j3:~/fabric-samples/first-network# ls
Name channel artifacts crypto-config docker-compose-1.yaml docker-compose-couch.yaml docker-compose-e2e.yaml kafka.sh README.md
root@fabric-j3:~/fabric-samples/first-network# docker ps
CONTAINER ID        IMAGE                COMMAND             CREATED            STATUS             PORTS              NAMES
root@fabric-j3:~/fabric-samples/first-network# ./byfn.sh up
Starting with channel 'mychannel' and CLI timeout of '10' seconds and CLI delay of '3' seconds
Continue? [Y/n] y
Proceeding ...
2018-04-13 11:25:41.525 UTC [main] main -> INFO 001 Exiting....
LOCAL_VERSION:1.0
DOCKER_IMAGE_VERSION:1.0
Creating network 'net_byfn' with the default driver
Creating volume 'net_peer0.org1.example.com' with default driver
Creating volume 'net_peer1.org1.example.com' with default driver
Creating volume 'net_peer2.org1.example.com' with default driver
Creating volume 'net_orderer.example.com' with default driver
Creating peer0.org1.example.com
Creating peer1.org1.example.com
Creating peer2.org1.example.com
Creating peer0.org1.example.com
Creating peer1.org1.example.com
Creating peer2.org1.example.com

```

So, now, this is going to create all of these entities. So, the peer 0 dot org 1 peer 0 dot org 2.

(Refer Slide Time: 18:09)

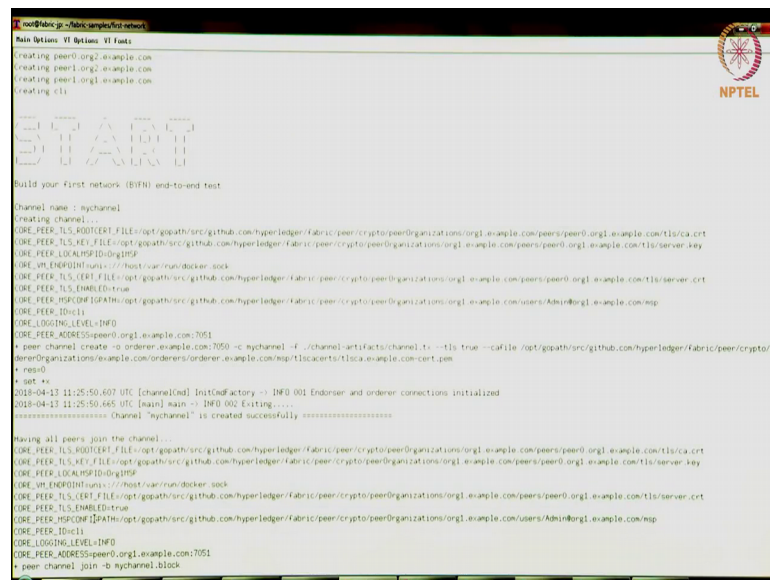
```

root@fabric-j3:~/fabric-samples/first-network#
Main Options VI Options VI Facts
Build your first network (BYFN) and-to-end test
[channel name] mychannel
Creating channel
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=org1sp
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_PROFILE_PATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
CORE_PEER_ID=cl
CORE_LOGGING_LEVEL=INFO
CORE_PEER_ADDRESS=peer0.org1.example.com:7051
+ peer channel create -o orderer.example.com:7050 -c mychannel -f /channel-artifacts/channel-1 --tls --no-file --file /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/orderer.example.com/msp/tlscaerts/tlsca.org1.example.com-cert.pem
+ ret0
2018-04-13 11:25:50.607 UTC [channelCmd] InitOfFactory -> INFO 001 Endorse and orderer connections initialized
2018-04-13 11:25:50.665 UTC [main] main -> INFO 002 Exiting....
===== Channel 'mychannel' is created successfully =====
Having all peers join the channel...
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=org1sp
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_PROFILE_PATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
CORE_PEER_ID=cl
CORE_LOGGING_LEVEL=INFO
CORE_PEER_ADDRESS=peer0.org1.example.com:7051
+ peer channel join -b mychannel block
+ ret0
+ ret +
2018-04-13 11:25:51.746 UTC [channelCmd] InitOfFactory -> INFO 001 Endorse and orderer connections initialized
2018-04-13 11:25:50.794 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
2018-04-13 11:25:50.794 UTC [main] main -> INFO 003 Exiting....
===== peer0.org1 joined on the channel 'mychannel' =====

```

Let us let it complete and then we will go over the output. So, it is going to run through a bit few things and I will walk you through what is actually doing in the backend and actually you can go back to the code and also figure out each of these steps and what it is doing in the backend. So, while it is running may be it can just it is take a couple of minutes we will start going through what it does right.

(Refer Slide Time: 18:33)



```
root@fabric-gp:~/fabric/sample-first-network#
Main Options: VI Options: VI Facts
creating peer0.org1.example.com
creating peer1.org1.example.com
creating peer2.org1.example.com
creating cli

Build your first network (BFTN) end-to-end test

Channel name : mychannel
Creating channel...
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=Org1MSP
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
CORE_PEER_ID=cli
CORE_PEER_ADDRESS=peer0.org1.example.com:7051
+ peer channel create -o orderer.example.com:7050 -c mychannel -f /channel-artifacts/channel.tx --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
+ exit 0
+ get +s
2018-04-13 11:25:50.607 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorse and orderer connections initialized
2018-04-13 11:25:50.662 UTC [main] main -> INFO 002 Exiting ----
===== Channel "mychannel" is created successfully =====
Having all peers join the channel...
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=Org1MSP
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
CORE_PEER_ID=cli
CORE_PEER_ADDRESS=peer0.org1.example.com:7051
+ peer channel join -b mychannel:610c
```

So, first it created the channel. So, using the channel configuration and the genesis block it went ahead and created this channel called “mychannel” it created that successfully. So, it is all clear then we are going to add the peers to the channel. So, peer 0 dot org 1 has joined the channel mychannel, peer 1 dot org 1 join the joined the channel, then peer 0 of org 2 I think it is skipped let us go back a bit peer 0 of org 2 and peer 1 of org 2 both joined the channel.

Then anchor peers for org 1 msp org 2 msp are updated so, we say that org 1. So, the peer 0 is going to be the anchor peer. So, we set the anchor peers for both the organizations org 2 and org 1. So, anchor peers are set these are the peers that are going to communicating with the orderer then we went ahead and installed a chaincode. So, this part I did not tell you about. So, I kept you (Refer Time: 19:43) about what it is going to do.

There is a chaincode that we have picked and installed. So, next chap we will go look at what chaincode we used right. So, there is a chaincode we setup on this channel, we installed that chaincode on peer 1 peer 1 dot org 2 and it is instantiated on the channel mychannel right. So, let us go back in the output. So, chaincode is installed on peer 0 org 1 chaincode is installed on peer 0 org 2. So, these are 2 anchor peers we installed that on them.

(Refer Slide Time: 20:27)

```

root@fabric-jp:~/fabric-samples/first-network#
Main Options: VI Options: VI Facts
CORE_PEER MSPIDM [PATH]:opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerorganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_ID=cl1
CORE_PEER_LOGGING_LEVEL=INFO
CORE_PEER_LOCALMSPID=peer1.org.example.com:7051
# peer chaincode install -n mycc -l 1.0 -l golang -p github.com/chaincode/chaincode.example02/go
+ res0
+ set +x
2018-04-13 11:27:34.887 UTC [chaincodeCmd] checkChaincodeParams -> INFO 001 Using default escc
2018-04-13 11:27:34.887 UTC [chaincodeCmd] checkChaincodeParams -> INFO 002 Using default vsc
2018-04-13 11:27:35.336 UTC [main] main -> INFO 003 Exiting....
***** Chaincode is installed on peer1.org *****

Querying chaincode on peer1.org...
CORE_PEER_TLS_ROOTCERT_FILES=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerorganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILES=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerorganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=golapp
CORE_PEER_ENDPOINTS=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerorganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPIDM [PATH]:opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerorganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_ID=cl1
CORE_PEER_LOGGING_LEVEL=INFO
CORE_PEER_ADDRESS=peer1.org.example.com:7051
***** Querying on peer1.org on channel 'mychannel' *****
Attempting to query peer1.org ... 3 sec
# peer chaincode query -C mychannel -n mycc -l '[*]go:[*]query,[*]'
+ res0
+ set +x
2018-04-13 11:27:38.575 UTC [chaincodeCmd] checkChaincodeParams -> INFO 001 Using default escc
2018-04-13 11:27:38.575 UTC [chaincodeCmd] checkChaincodeParams -> INFO 002 Using default vsc
Query Result: 90
2018-04-13 11:28:18.143 UTC [main] main -> INFO 003 Exiting....
***** Query on peer1.org on channel 'mychannel' is successful *****

***** All GOOD - BFN execution completed *****

         ____
        |  _ \ |
        | |_) | |
        |  _ < | |
        | |_) | |
        |  __/ | |
        |_____|_|

root@fabric-jp:~/fabric-samples/first-network#

```

Then after that it is rolled over a bit script is finished now. So, that makes it easier for us to go over things so, the chaincode has been added. So, then the chaincode was instantiated on the channel “mychannel” right using peer 0, this is already joined the channel. So, once the chaincode is all set up then you can query and invoke the chaincode you can perform the different transactions. So, we will do that. But before that let us first look at what containers have started.

(Refer Slide Time: 20:58)

```

root@fabric-jp:~/fabric-samples/chaincode/chaincode.example02#
Main Options: VI Options: VI Facts
*****
| | | | | | | | | | | | |
|_|_|_|_|_|_|_|_|_|_|_|
| |_|_|_|_|_|_|_|_|_|_|_|
| |_|_|_|_|_|_|_|_|_|_|_|
| |_|_|_|_|_|_|_|_|_|_|_|

root@fabric-jp:~/fabric-samples/first-network# docker ps
CONTAINER ID        IMAGE                COMMAND              CREATED           STATUS
92125a8a3db        dev-peer1.org2.example.com:mycc-1.0-262af32838554aac47a8410ac9a95ae79593a12e68d7443d01f3346    "chaincode-peer.a..."   30 seconds ago   Up 29 s
54555fa952c4        dev-peer0.org1.example.com:mycc-1.0-384f1f48497028f00453200c625174305fce9f53f4e94465ee28ccab0e9    "chaincode-peer.a..."   About a minute ago   Up About a minute
808b2eb9423        dev-peer0.org2.example.com:mycc-1.0-15657f0c9496601bc144939a6764e0f1365a7f3951674285e07e42b    "chaincode-peer.a..."   About a minute ago   Up About a minute
60e5686cf5d        hyperledger/fabric-tools:latest      cli                   "fabricsh"           2 minutes ago       Up 2 min
13799755e0         hyperledger/fabric-peer:latest       cli                   "peer node start"     3 minutes ago       Up 2 min
f01fa30e47         hyperledger/fabric-peer:latest       peer0.org1.example.com
ites               0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp      peer0.org2.example.com
peer node start"  3 minutes ago       Up 2 min
49bc47721681     hyperledger/fabric-orderer:latest    orderer.example.com
ites               0.0.0.0:7050->7050/tcp                        orderer.example.com
peer node start"  3 minutes ago       Up 3 min
60ee9423e9d         hyperledger/fabric-peer:latest       peer1.org1.example.com
ites               0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp      peer1.org1.example.com
peer node start"  3 minutes ago       Up 2 min
362e9d9e04c         hyperledger/fabric-peer:latest       peer1.org1.example.com
ites               0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp      peer0.org1.example.com
peer node start"  3 minutes ago       Up 2 min

root@fabric-jp:~/fabric-samples/first-network#
root@fabric-jp:~/fabric-samples/first-network# cd ../
root@fabric-jp:~/fabric-samples# ls
balance-transfer  bin  chaincode  docker  devnode  fabric  first-network  LICENSE  README.md
base-network     chaincode  config     fabric  ca  high-throughput  JARFILES.md  scripts
root@fabric-jp:~/fabric-samples# cd chaincode
root@fabric-jp:~/fabric-samples/chaincode# ls
dev  chaincode.example02  fabric  wallet=02  sax
root@fabric-jp:~/fabric-samples/chaincode# cd chaincode.example02/
root@fabric-jp:~/fabric-samples/chaincode/chaincode.example02# ls
go  node
root@fabric-jp:~/fabric-samples/chaincode/chaincode.example02# cd go/
root@fabric-jp:~/fabric-samples/chaincode/chaincode.example02/go# ls
1
chaincode.example02.go
root@fabric-jp:~/fabric-samples/chaincode/chaincode.example02/go# vi chaincode.example02.go

```

So, I am going to do docker ps. So, these are the containers that have started. So, we have peer 0 dot org 1 we have peer 0 dot org 2 peer 1 dot org 1 and peer 1 dot org 2. So, these are the 4 peers 2 peers for each organization that is come up

and note that we have not brought up ca in this example we use the cryptogen tool to generate the certificates we did not use a fabric ca. So, that is also the power of an msp, we do not really the fabric itself does not care at where the certificates come from as long as the msp knows the certificates right, it knows the public keys for verification.

So, you can pre generate the certificates bring them on board into fabric and plug them and that is what we are doing now the other option of course, is to have a fabric ca generate the certificates and give it to the network. So, apart from that there is chaincodes that have been deployed. So, chaincodes have been deployed on org 1 org 2 and actually 2 peers on org 2 right. So, that is on both peer 0 and peer 1 chaincodes have been deployed and they have brought they those containers are up.

And another cool feature of fabric is that it automatically handles the default (Refer Time: 22:21) with chaincodes, let us say these are all docker containers right the chaincode might crash what if this docker container this process crashes. The next time you perform a query or a transaction on that node, the chaincode will automatically come up and then the transaction will run. So, fabric will takes care of bring up the net[work] the chaincode whenever necessary right.

And all you have to do is when you instantiate you are bringing up the chaincode on that node only if you do a transaction or a query on another peer we will that think about, the reason we got these 3 up was because we performed transactions as part of this script you will see that there were queries and there were queries performed on those peers and when you perform those queries those chaincodes came up automatically.

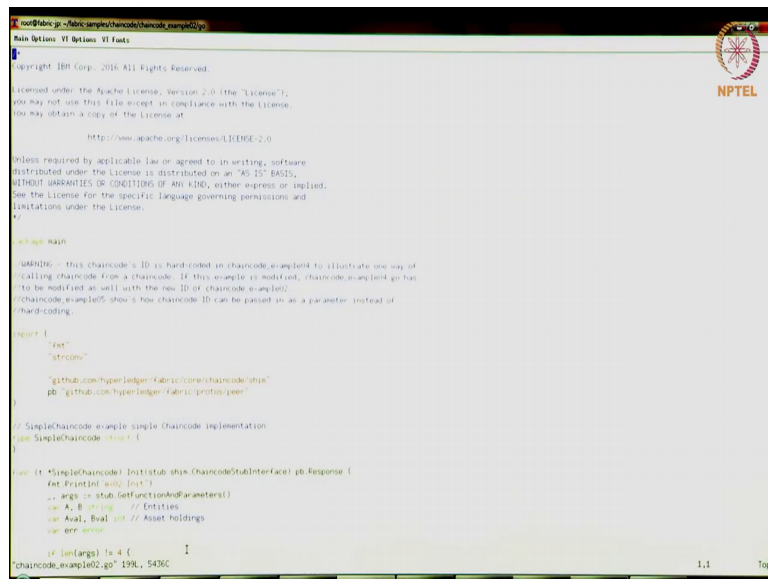
So, when talking about chaincodes let us go see what that chaincode is we actually deploy a very simple chaincode. We will go up one directory we will go to the parent directory fabric samples there is a folder called chaincode right. So, this actually has multiple chaincode examples, but the one we have used now is chaincode example 02 right. I will walk through the chaincode example 02 for you and note that there is also marbles chaincode. So, this is precisely the chaincode that we used then and deployed in the IBM cloud right in the marbles example.

So, this chaincode is also available here, you can use this and deploy the marbles chaincode also on top of this network. So, I would not be showing that to you, but I encourage you to do that as an exercise take this marbles chaincode install it on multiple

peers instantiate it on the mychannel channel that you have created right. So, try it as tried that try that as an exercise.

So, now, let us look at chaincode example 02 if you go in there is a go code as well as the node code node js code which is the application code, but let us now only focus on the smart contract or the golang code. So, there is a chaincode example 0 2 dot go.

(Refer Slide Time: 24:30)



```
root@ibrc-go:~/fabric-simple/chaincode/chaincode_example02.go
Main Options: VI Options: VI Facts
Copyright IBM Corp. 2016 All Rights Reserved.
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
    http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
//
// Package main
//
// WARNING: this chaincode's ID is hard-coded in chaincode example02 to illustrate one way of
// calling chaincode from a chaincode. If this example is modified, chaincode example02.go has
// to be modified as well with the new ID of chaincode example02.
// chaincode example02 shows how chaincode ID can be passed in as a parameter instead of
// hard-coding.
import (
    "fmt"
    "strconv"
    "github.com/hyperledger/fabric/core/chaincode/shim"
    pb "github.com/hyperledger/fabric/protos/peer"
)

// SimpleChaincode example simple Chaincode implementation
type SimpleChaincode struct {

}

func (t *SimpleChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {
    fmt.Println("init")
    // args is stub.GetFunctionParameters()
    var A, B string // Entities
    var Aval, Bval int // Asset holdings
    var err error

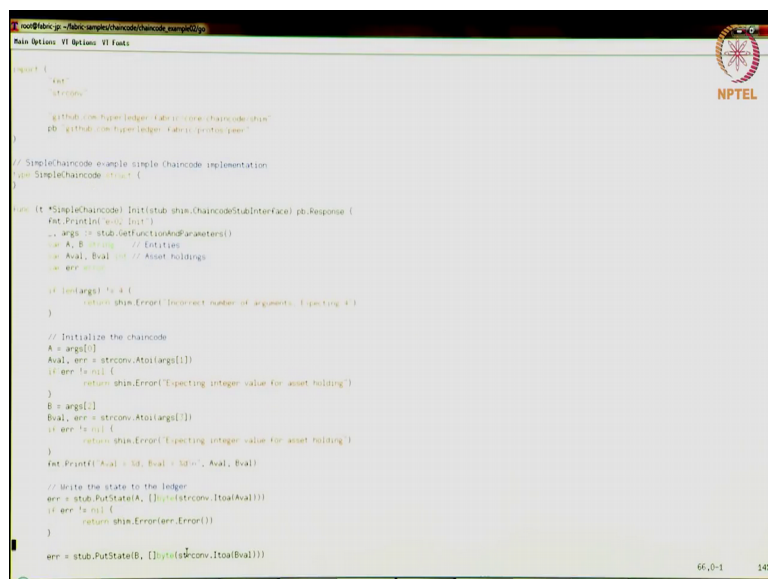
    if len(args) != 4 {
        return shim.Error("Incorrect number of arguments: expecting 4")
    }

    // Initialize the chaincode
    A = args[0]
    Aval, err = strconv.Atoi(args[1])
    if err != nil {
        return shim.Error("Expecting integer value for asset holding")
    }
    B = args[2]
    Bval, err = strconv.Atoi(args[3])
    if err != nil {
        return shim.Error("Expecting integer value for asset holding")
    }
    fmt.Printf("Aval = %d, Bval = %d\n", Aval, Bval)

    // Write the state to the ledger
    err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
    if err != nil {
        return shim.Error(err.Error())
    }
    err = stub.PutState(B, []byte(strconv.Itoa(Bval)))
}
```

So, let us look at this function, as would any smart contract any chaincode look there are 2 main functions, one is the init function and the other is the invoke function right.

(Refer Slide Time: 24:43)



```
root@ibrc-go:~/fabric-simple/chaincode/chaincode_example02.go
Main Options: VI Options: VI Facts
Copyright IBM Corp. 2016 All Rights Reserved.
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
    http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
//
// Package main
//
// WARNING: this chaincode's ID is hard-coded in chaincode example02 to illustrate one way of
// calling chaincode from a chaincode. If this example is modified, chaincode example02.go has
// to be modified as well with the new ID of chaincode example02.
// chaincode example02 shows how chaincode ID can be passed in as a parameter instead of
// hard-coding.
import (
    "fmt"
    "strconv"
    "github.com/hyperledger/fabric/core/chaincode/shim"
    pb "github.com/hyperledger/fabric/protos/peer"
)

// SimpleChaincode example simple Chaincode implementation
type SimpleChaincode struct {

}

func (t *SimpleChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response {
    fmt.Println("init")
    // args is stub.GetFunctionParameters()
    var A, B string // Entities
    var Aval, Bval int // Asset holdings
    var err error

    if len(args) != 4 {
        return shim.Error("Incorrect number of arguments: expecting 4")
    }

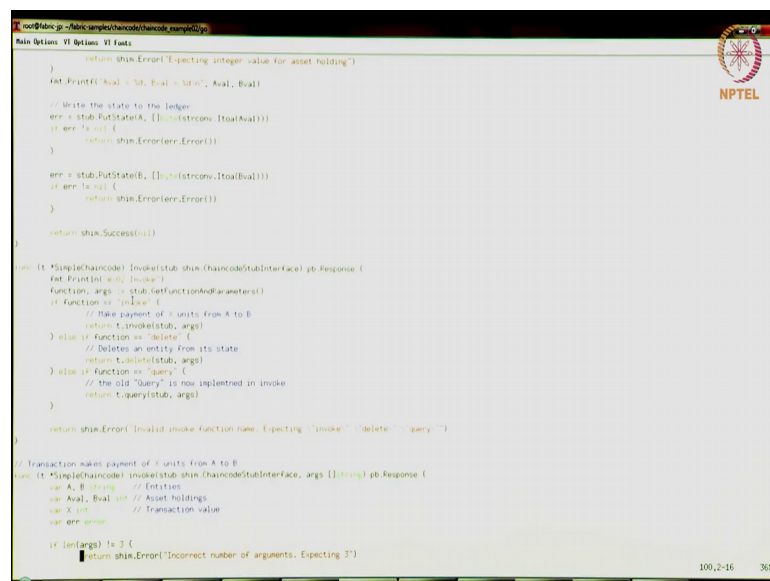
    // Initialize the chaincode
    A = args[0]
    Aval, err = strconv.Atoi(args[1])
    if err != nil {
        return shim.Error("Expecting integer value for asset holding")
    }
    B = args[2]
    Bval, err = strconv.Atoi(args[3])
    if err != nil {
        return shim.Error("Expecting integer value for asset holding")
    }
    fmt.Printf("Aval = %d, Bval = %d\n", Aval, Bval)

    // Write the state to the ledger
    err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
    if err != nil {
        return shim.Error(err.Error())
    }
    err = stub.PutState(B, []byte(strconv.Itoa(Bval)))
}
```

The init is going to be called as I mentioned when the chaincode is instantiated. So, this is the first time something is going to some transaction is happening on this chaincode, it is been instantiated we will call the init function to initialize any state information. So, what this chaincode does, it is very simple chaincode it has 2 state information it has 2 accounts account A and account B. They both initialized with a certain balance we can think of them as bank accounts and they have a certain balance.

So, in this case there is some function error checking that is first done, we are first going to initialize A with a certain balance and B with a certain balance and the balance is really a argument provided to the initialization function to the init function right. So, in this example we are actually going to set A's balance to be 100 and B's balance to be 200. So, this is what is going to get means initialized as this state information for this chaincode.

(Refer Slide Time: 25:44)



```
return shim.Error("Expecting integer value for asset holding")
}
fmt.Printf("Aval = %d, Bval = %d\n", Aval, Bval)

// Write the state to the ledger
err = stub.PutStateA, ([]byte)(strconv.Itoa(Aval)))
if err != nil {
    return shim.Error(err.Error())
}

err = stub.PutStateB, ([]byte)(strconv.Itoa(Bval)))
if err != nil {
    return shim.Error(err.Error())
}

return shim.Success(nil)

func (t *SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
    fmt.Println("invoke")
    function, args := stub.GetFunctionAndParameters()
    if function == "invoke" {
        // Make payment of X units from A to B
        return t.invoke(stub, args)
    } else if function == "delete" {
        // Deletes an entity from its state
        return t.delete(stub, args)
    } else if function == "query" {
        // the old "query" is now implemented in invoke
        return t.query(stub, args)
    }

    return shim.Error("Invalid invoke function name. Expecting 'invoke', 'delete', 'query'")
}

// Transaction makes payment of X units from A to B
func (t *SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    var A, B string // Entities
    var Aval, Bval int // Asset holdings
    var X int // Transaction value
    var err error

    if len(args) != 3 {
        return shim.Error("Incorrect number of arguments. Expecting 3")
    }
}
```

And what is the function it supports, it supports invoke delete and query. So, this the invoke function it supports arguments for invoke, delete and query. So, what I can do is that, I can invoke what does invoke do it actually makes a payment of X units from account A to account B. So, it is going to decrement A's balance by X and it is going to increment B's balance by X. So, that invoke function is out here right.

(Refer Slide Time: 26:16)

```
node@lib:~$ fabric-complexchaincode/chaincode_main.go
Main Options: 01 Options: 01 Facts

// A, B = string // Entities
// Aval, Bval int // Asset Holdings
// X int // Transaction value
// err error

if (len(args) != 3) {
    return shim.Error("Incorrect number of arguments. Expecting 3")
}

A = args[0]
B = args[1]

// Get the state from the ledger
// TODO: will be nice to have a GetAllState call to ledger
AvalBytes, err := stub.GetState(A)
if err != nil {
    return shim.Error("Failed to get state")
}
if AvalBytes == nil {
    return shim.Error("Entity not found")
}
Aval := strconv.Atoi(string(AvalBytes))

BvalBytes, err := stub.GetState(B)
if err != nil {
    return shim.Error("Failed to get state")
}
if BvalBytes == nil {
    return shim.Error("Entity not found")
}
Bval := strconv.Atoi(string(BvalBytes))

// Perform the execution
X, err := strconv.Atoi(args[2])
if err != nil {
    return shim.Error("Invalid transaction amount, expecting a integer value")
}

Aval = Aval - X
Bval = Bval + X

fmt.Println(Aval + " ", Bval + " sum: ", Aval, Bval)

// Write the state back to the ledger
err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
if err != nil {
```

Here what so, here we are going to take those parameters convert from string to integer, fabric always takes all arguments as strings and you will have to then morph them into any data type of your choice. So, here is the logic where A's balance is decremented by X, B's balance is incremented by X, but first for before doing that you will have to read the current state of the balance of these accounts. So, that is done in this GetState function. So, the getstate will read as balance getstate of A reads A's balance from the ledger, getstate of B reads B's balance from the ledger, they are both updated.

And then we have to write them back into the ledger. So, we are going to do a putstate of a saying this is the new balance for A and likewise for B, you are going to do A putstate for B right. So, that is really what this chaincode is about right.

(Refer Slide Time: 27:12)


```
root@fabric-gp:~/fabric-sample/chaincode/chaincode_example02#
Main Options: VI Options: VI Facts

)
if BvalBytes == nil {
    return shim.Error("Entity not found")
}
Bval := strconv.Atoi(string(BvalBytes))

// Perform the execution
err := strconv.Atoi(args[1])
if err != nil {
    return shim.Error("Invalid transaction amount, expecting a integer value")
}
Aval = Aval - X
Bval = Bval + X
fmt.Printf("Aval = %d, Bval = %d\n", Aval, Bval)

// Write the state back to the ledger
err = stub.PutStateA, []byte(strconv.Itoa(Aval))
if err != nil {
    return shim.Error(err.Error())
}

err = stub.PutStateB, []byte(strconv.Itoa(Bval))
if err != nil {
    return shim.Error(err.Error())
}

return shim.Success(nil)

// Deleted an entity from state
func (t *SimpleChaincode) delete(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    if len(args) != 1 {
        return shim.Error("Incorrect number of arguments. Expecting 1")
    }
    A := args[0]

    // Delete the key from the state in ledger
    err := stub.DeleteStateA()
    if err != nil {
        return shim.Error("Failed to delete state")
    }

    return shim.Success(nil)
}
163,2-9 76%
```

Separately there is also a query function where you can just ask for the account balance, what is the account balance for A or what is the Account balance for B.

(Refer Slide Time: 27:12)

```
root@fabric-gp:~/fabric-sample/chaincode/chaincode_example02#
Main Options: VI Options: VI Facts

)
// Query callback representing the query of a chaincode
func (t *SimpleChaincode) query(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    A := string(args[0])
    err := stub

    if len(args) != 1 {
        return shim.Error("Incorrect number of arguments. Expecting name of the person to query")
    }
    A := args[0]

    // Get the state from the ledger
    AvalBytes, err := stub.GetStateA()
    if err != nil {
        jsonResp := "{\"Error\":\"Failed to get state for " + A + "\"}"
        return shim.Error(jsonResp)
    }

    if AvalBytes == nil {
        jsonResp := "{\"Error\":\"No amount for " + A + "\"}"
        return shim.Error(jsonResp)
    }

    jsonResp := "{\"Name\":\"" + A + "\",\"Amount\":\"" + strconv.Itoa(int(*AvalBytes)) + "\"}"
    fmt.Printf("Query Response: %s\n", jsonResp)
    return shim.Success(AvalBytes)
}

func main() {
    err := shim.StartWith(SimpleChaincode)
    if err != nil {
        fmt.Printf("Error starting Simple chaincode: %s", err)
    }
}
165,0-1 86%
```

It will return the balance it will query the ledger this is the get state function and it will return that back return that to the user.

(Refer Slide Time: 27:37)

```

root@fabric-jp:~/fabric-samples/chaincode/chaincode_example02# docker ps
CONTAINER ID        IMAGE                                     COMMAND                                CREATED            STATUS
11315a849db        dev-peer1.org2.example.com-mycs-1.0-262e1283855aac3f7a8f10acaf65e7959c3a12e66764c8d1f834ab  "chaincode-peer.a..." 30 seconds ago    Up 29 s
700b                dev-peer1.org1.example.com-mycs-1.0                dev-peer1.org1.example.com-mycs-1.0  "chaincode-peer.a..." About a minute ago  Up About a minute
94585fa52c4        dev-peer0.org1.example.com-mycs-1.0-39411f484930d96c4520c8b25174305c6af5f4e49485ee3bcabc09 "chaincode-peer.a..." About a minute ago    Up About a minute
682bc26c9423        dev-peer0.org2.example.com-mycs-1.0-15671b3c049467e74497a7b7674d011465a9f79510429c076c42b "chaincode-peer.a..." About a minute ago    Up About a minute
60c5686c45d        hyperledger/fabric-tools:latest             /bin/bash                              2 minutes ago     Up 2 minutes
177887555e0        hyperledger/fabric-peer:latest              cli                                     "peer node start"     3 minutes ago     Up 2 minutes
0.0.0.0:9051->7051/tcp, 0.0.0.0:9051->7053/tcp
f61fa30c647        hyperledger/fabric-peer:latest              "peer node start"     3 minutes ago     Up 2 minutes
0.0.0.0:10051->7051/tcp, 0.0.0.0:10051->7053/tcp
919c4771683        hyperledger/fabric-orderer:latest           orderer.example.com    "orderer"            3 minutes ago     Up 3 minutes
0.0.0.0:7050->7050/tcp
f80ef423d9d        hyperledger/fabric-peer:latest              "peer node start"     3 minutes ago     Up 2 minutes
0.0.0.0:8051->7051/tcp, 0.0.0.0:8051->7053/tcp
362a9d6b04c        hyperledger/fabric-peer:latest              "peer node start"     3 minutes ago     Up 2 minutes
0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp
peer0.org1.example.com
root@fabric-jp:~/fabric-samples/first-network#
root@fabric-jp:~/fabric-samples/first-network# cd ../
root@fabric-jp:~/fabric-samples# ls
balance-transfer  bin                chaincode-docker-devmode  fabric  first-network  LICENSE  README.md
basic-network    chaincode  config  fabric-ca  high-throughput  MAINTAINERS.md  scripts
root@fabric-jp:~/fabric-samples# cd chaincode
root@fabric-jp:~/fabric-samples/chaincode# ls
chaincode_example02  fabric  makechain01  src
root@fabric-jp:~/fabric-samples/chaincode# cd chaincode_example02/
root@fabric-jp:~/fabric-samples/chaincode/chaincode_example02# ls
go.mod
root@fabric-jp:~/fabric-samples/chaincode/chaincode_example02# cd go/
root@fabric-jp:~/fabric-samples/chaincode/chaincode_example02/go# ls
chaincode_example02.go
root@fabric-jp:~/fabric-samples/chaincode/chaincode_example02/go# vi chaincode_example02.go
root@fabric-jp:~/fabric-samples/chaincode/chaincode_example02/go#

```

So, that is the, that is all this chaincode is about reads account balances and it allows you to transfer X units from one account to another. So, let us head back and let us go back to the first network that we had.

(Refer Slide Time: 27:50)

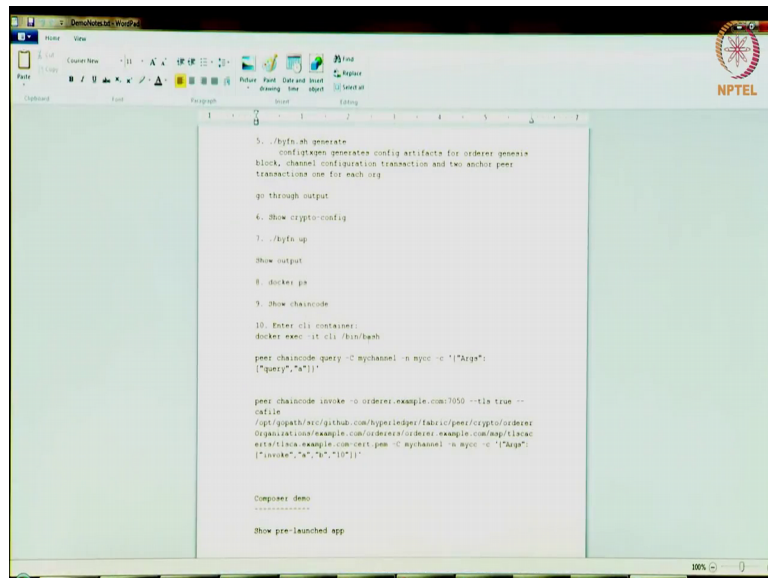
```

root@00c5686b5d:~/opt/gopath/src/github.com/hyperledger/fabric#
chaincode_example02.go
root@fabric-jp:~/fabric-samples/chaincode/chaincode_example02/go# vi chaincode_example02.go
root@fabric-jp:~/fabric-samples/chaincode/chaincode_example02/go# cd ../
root@fabric-jp:~/fabric-samples/chaincode# ls
balance-transfer  bin                chaincode-docker-devmode  fabric  first-network  LICENSE  README.md
basic-network    chaincode  config  fabric-ca  high-throughput  MAINTAINERS.md  scripts
root@fabric-jp:~/fabric-samples/chaincode# cd first-network/
root@fabric-jp:~/fabric-samples/chaincode/first-network# ls
base  channel-artifacts  crypto  config  docker-compose-cli.yaml  docker-compose-couch.yaml  docker-compose-e2e.yaml  go.mod  README.md
base-config.yaml  crypto-config.yaml  docker-compose-couch-org2.yaml  docker-compose-e2e-template.yaml  docker-compose-org1.yaml  org1-artifacts  scripts
root@fabric-jp:~/fabric-samples/chaincode/first-network# docker ps
CONTAINER ID        IMAGE                                     COMMAND                                CREATED            STATUS
11315a849db        dev-peer1.org2.example.com-mycs-1.0-262e1283855aac3f7a8f10acaf65e7959c3a12e66764c8d1f834ab  "chaincode-peer.a..." 7 minutes ago     Up 7 minutes
94585fa52c4        dev-peer0.org1.example.com-mycs-1.0-39411f484930d96c4520c8b25174305c6af5f4e49485ee3bcabc09 "chaincode-peer.a..." 8 minutes ago     Up 8 minutes
682bc26c9423        dev-peer0.org2.example.com-mycs-1.0-15671b3c049467e74497a7b7674d011465a9f79510429c076c42b "chaincode-peer.a..." 8 minutes ago     Up 8 minutes
60c5686c45d        hyperledger/fabric-tools:latest             /bin/bash                              9 minutes ago     Up 9 minutes
177887555e0        hyperledger/fabric-peer:latest              cli                                     "peer node start"     9 minutes ago     Up 9 minutes
0.0.0.0:9051->7051/tcp, 0.0.0.0:9051->7053/tcp
peer0.org2.example.com
f61fa30c647        hyperledger/fabric-peer:latest              "peer node start"     9 minutes ago     Up 9 minutes
0.0.0.0:10051->7051/tcp, 0.0.0.0:10051->7053/tcp
peer1.org2.example.com
919c4771683        hyperledger/fabric-orderer:latest           orderer.example.com    "orderer"            9 minutes ago     Up 9 minutes
0.0.0.0:7050->7050/tcp
f80ef423d9d        hyperledger/fabric-peer:latest              "peer node start"     9 minutes ago     Up 9 minutes
0.0.0.0:8051->7051/tcp, 0.0.0.0:8051->7053/tcp
peer0.org1.example.com
362a9d6b04c        hyperledger/fabric-peer:latest              "peer node start"     9 minutes ago     Up 9 minutes
0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp
peer0.org1.example.com
root@fabric-jp:~/fabric-samples/first-network# docker exec -it cli /bin/bash
root@00c5686b5d:~/opt/gopath/src/github.com/hyperledger/fabric/peer# ls
channel-artifacts  crypto  log.txt  mychannel  blocks  scripts
root@00c5686b5d:~/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode query -c mychannel -n myc -c '{"Args":["query","a"]}'
2018-04-13 11:37:36.426 UTC [chaincodeCmd] checkChaincodeParams -> INFO 002 Using default vsc
2018-04-13 11:37:36.426 UTC [chaincodeCmd] checkChaincodeParams -> INFO 002 Using default vsc
Query Result: 90
2018-04-13 11:37:36.448 UTC [main] main -> INFO 003 Exiting.....
root@00c5686b5d:~/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode query -c mychannel -n myc -c '{"Args":["query","b"]}'
2018-04-13 11:37:36.484 UTC [chaincodeCmd] checkChaincodeParams -> INFO 001 Using default vsc
2018-04-13 11:37:36.484 UTC [chaincodeCmd] checkChaincodeParams -> INFO 001 Using default vsc
Query Result: 210
2018-04-13 11:37:36.493 UTC [main] main -> INFO 002 Exiting.....
root@00c5686b5d:~/opt/gopath/src/github.com/hyperledger/fabric/peer#

```

I will again show you the containers one other container that we skipped over is the cli container this is the command line interface that you can use to connect to the network right. So, what I am going to do is, I am going to now enter this container this container and run command line invocations and queries on the smart contract that we have deployed.

(Refer Slide Time: 28:14)



```
5. ./bin/ah generate
   configtxgen generates config artifacts for orderer genesis
   block, channel configuration transaction and two anchor peer
   transactions one for each org

   go through output

6. show crypto-config

7. ./bin/ah up

   show output

8. docker ps

9. show chaincode

10. Enter cli container:
    docker exec -it cli bin/bash

    peer chaincode query -c mychannel -n mycc -c '{"Args":
    ["query","a"]}'

    peer chaincode invoke -o orderer.example.com:7050 --tls true --
    cafile
    /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/orderer
    orgs/organizations/organizations/orderer.example.com/assets/peers
    orgs/clients/example-contract/peer -c mychannel -n mycc -c '{"Args":
    ["invoke","a","b","10"]}'

Completed Demo
-----
Show pre-launched app
```

So, I am going to use cheat sheet here where I have these readily available. So, what I am going to do is, I have just entered I have used a docker exec command and I have now entered the container the cli container that I have here. So, this is really the cli container and I am in the peer folder inside the container and from here from this cli container this is really think of this as the client right; I am now going to query and invoke the chaincode.

And note that as part of the initial script itself there were a few transactions that happened. So, there was actually one transaction that move 10 units from a to b. So, a balances was initially 100, b's balance was 200 we transferred 10 units from a to b as part of this thing. So, let us first start with querying the network, I have a cli command here. So, I am going to use that so, let us just look at the structure of this cli command.

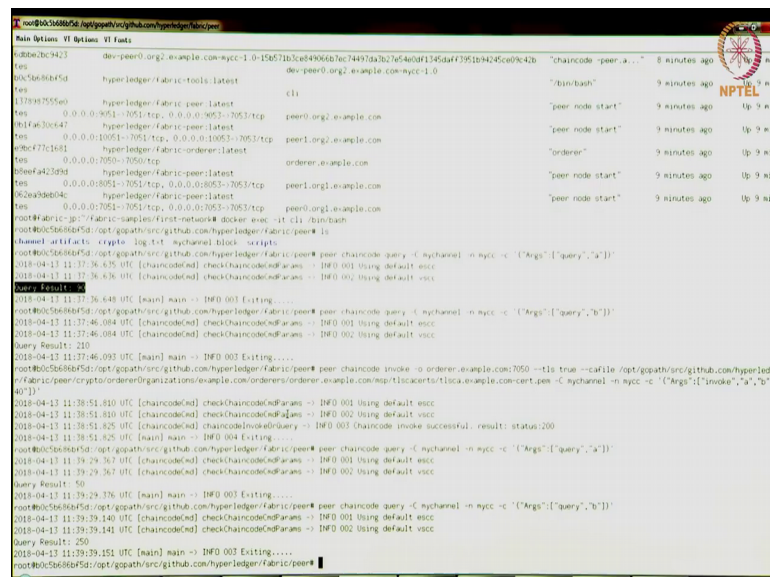
So, it is just peer chaincode query it as a reference to the channel so, minus c mychannel. So, that is a channel we are going to operate on mycc is the chaincode I have deployed. So, I have installed on the channel. So, it is minus n minus mycc and minus c as the args provided. So, the args are I am going to call the query function and I am going to query for the account balance of a right. So, that is what this cli command is going to do.

So, if you run it you will see the query result is 90 as expected. So, this is the balance for A, likewise let us also query for the balance of B and it says the balance for the B is 210 ok. So, now, let us do our blockchain transaction and again I have a cheat sheet with full

commands. So, that I do not have to type that all out right, now let us go over the structure of this invoke command.

So, it is a peer chaincode invoke minus orderer. So, we have we need to know who the orderer is minus minus tls saying I am going to use tls certificates to all my network communications is going to be encrypted, tls is true. What is the ca file? The ca is here is as the then full path to this tlscert, the tlscert for this here, minus c for mychannel minus n mycc this is the chaincode reference and minus c arguments. So, in the arguments we are calling the invoke function we are going to say transfer from a to b so many units.

(Refer Slide Time: 30:53)



```
root@05c5686f5d:~/opt/gopath/src/github.com/hyperledger/fabric#
Main Options: VI Options: VI Facts
8088c7bc9423  dev-peer0.org2.example.com:mycc-1.0:1565716c484960b7ec74477a3b276540af1345df3951b9426c0e9c42b  "chaincode-peer.a..." 8 minutes ago Up 9 min
80c5686b4f5d  hyperledger/fabric-tools:latest  dev-peer0.org1.example.com:mycc-1.0  "bin/bash" 9 minutes ago Up 9 min
1179497555e0  hyperledger/fabric-peer:latest  cli  "peer node start" 9 minutes ago Up 9 min
80174630c447  hyperledger/fabric-peer:latest  peer0.org2.example.com  "peer node start" 9 minutes ago Up 9 min
f0e  0.0.0.0:9051->7051/tcp, 0.0.0.0:9053->7053/tcp  peer1.org1.example.com  "orderer" 9 minutes ago Up 9 min
49bc4771681  hyperledger/fabric-orderer:latest  orderer.example.com  "peer node start" 9 minutes ago Up 9 min
85ee7a22094  hyperledger/fabric-peer:latest  peer1.org1.example.com  "peer node start" 9 minutes ago Up 9 min
362a9d0e04c  hyperledger/fabric-peer:latest  peer1.org1.example.com  "peer node start" 9 minutes ago Up 9 min
root@fabric:~/opt/fabric-first-network# docker exec -it cli /bin/bash
root@05c5686f5d:~/opt/gopath/src/github.com/hyperledger/fabric/peer# ls
channel artifacts crypto log.txt mychannel block scripts
root@05c5686f5d:~/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}'
2018-04-13 11:37:36.436 UTC [chaincodeCmd] checkChaincodeDefParams -> INFO 001 Using default vssc
2018-04-13 11:37:36.436 UTC [chaincodeCmd] checkChaincodeDefParams -> INFO 002 Using default vssc
New Result:
2018-04-13 11:37:36.448 UTC [main] main -> INFO 001 Exiting....
root@05c5686f5d:~/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode query -C mychannel -n mycc -c '{"Args":["query","b"]}'
2018-04-13 11:37:46.084 UTC [chaincodeCmd] checkChaincodeDefParams -> INFO 001 Using default vssc
2018-04-13 11:37:46.084 UTC [chaincodeCmd] checkChaincodeDefParams -> INFO 002 Using default vssc
Query Result: 200
2018-04-13 11:37:46.093 UTC [main] main -> INFO 003 Exiting....
root@05c5686f5d:~/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/tlscerts/tlscc.example.com-cert.pem -C mychannel -n mycc -c '{"Args":["invoke","a","b","40"]}'
2018-04-13 11:38:51.810 UTC [chaincodeCmd] checkChaincodeDefParams -> INFO 001 Using default vssc
2018-04-13 11:38:51.810 UTC [chaincodeCmd] checkChaincodeDefParams -> INFO 002 Using default vssc
2018-04-13 11:38:51.825 UTC [chaincodeCmd] chaincodeInvokeAndQuery -> INFO 003 Chaincode invoke successful. result: status:200
2018-04-13 11:38:51.825 UTC [main] main -> INFO 004 Exiting....
root@05c5686f5d:~/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}'
2018-04-13 11:39:29.367 UTC [chaincodeCmd] checkChaincodeDefParams -> INFO 001 Using default vssc
2018-04-13 11:39:29.367 UTC [chaincodeCmd] checkChaincodeDefParams -> INFO 002 Using default vssc
Query Result: 50
2018-04-13 11:39:29.376 UTC [main] main -> INFO 007 Exiting....
root@05c5686f5d:~/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode query -C mychannel -n mycc -c '{"Args":["query","b"]}'
2018-04-13 11:39:39.140 UTC [chaincodeCmd] checkChaincodeDefParams -> INFO 001 Using default vssc
2018-04-13 11:39:39.141 UTC [chaincodeCmd] checkChaincodeDefParams -> INFO 002 Using default vssc
Query Result: 250
2018-04-13 11:39:39.151 UTC [main] main -> INFO 007 Exiting....
root@05c5686f5d:~/opt/gopath/src/github.com/hyperledger/fabric/peer#
```

So, we going to say in this orderer says 10 units, but let us say we transfer 40 units from a to b ok.

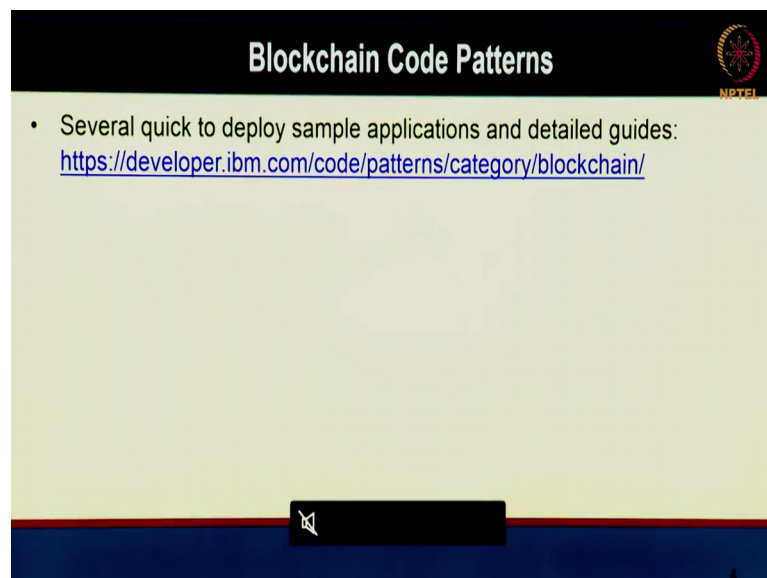
So, what we do is, this is going to be a asynchronous transaction. So, we have submitted this transaction on to the blockchain network and blockchain has told us that yes this is been accepted we get a 200 back, but this is not yet committed on to the ledger right. It might fail for some reason because may be the account balances are not there may be somewhere else tried to also spend at the same time. So, there is a conflict and the state is not state validation fails. So, many reasons why it could fail, but in this case I hope it is it is going to succeed.

So, let us query now for the account balance for a you see that it is now read (Refer Time: 31:38) 40 we transferred forty from a to b. So, from 90 it came down to 50 and we can see that B has gone from 210 to 250. So, that is the atomicity so, either both accounts will be updated or neither will be updated. So, if it fails, then the account balances will remain what was it before it could have remained 90 to 10 if this transaction is failed right.

So, we have gone through how you setup a network by yourself from scratch, you brought up the, you created the identities. You created the certificates you brought up the network as docker containers. So, in this network we have 2 orderers sorry 1 orderer, 2 peers for 2 organizations. So, 4 peers total and we installed instantiated a chaincode we performed certain transactions on that chaincode, we also perform some queries to find out account balances and so, that is that is one simple application deployed on your own blockchain network.

So, as I as an exercise as I mentioned please do try and install the marbles application and try and try and execute some transactions on that marbles application. So, with that let us get back here.

(Refer Slide Time: 33:01)



Apart from this IBM is also put out many code patterns example applications, again the code is there the examples are there. So, you can go quickly try those out as well. So, there are many different code patterns provided they are there in this link here.

You can go look that up in their code patterns. So, with that I hope you have a good feel for how to get started using hyperledger fabric develop your own applications set up your network deploy the applications layer and hopefully you can get started in building your own blockchain applications and do some projects.

Thanks a lot with that we end this demo session we will see you at the next lecture, bye.