SIMBA

INSTALL HYPERLEDGER FABRIC 2.1 EVM

USER GUIDE

BY ADAM BRINCKMAN

Senior Architect, SIMBA Chain and Senior Research Programmer/Product Owner,Center for Research Computing at the University of Notre Dame

How to install Hyperledger Fabric EVM chaincode onto the latest release of Hyperledger Fabric (v2.1).

This guide is written for developers who want to install Hyperledger Fabric EVM chaincode onto the latest release of Hyperledger Fabric (at the time of this writing is version 2.1). The reason I am writing this article is because the official documentation provides examples for installing onto version 1.4 which is different from the process of installing to a 2.1 network.

It is assumed the reader has plenty of experience writing and compiling Solidity smart contracts and knows how to get their hands on the compiled bytecode, plus any other artifacts normally required for deploying smart contracts onto the Ethereum network. The process of compiling Solidity will not be explained. It is also assume the reader has at least a basic understanding of IBM's Hyperledger Fabric and has already installed Fabric 2.1 (if you haven't then please *follow this documentation* to install all of the binaries and docker images).

WHAT IS FABRIC EVM CHAINCODE AND WHY WOULD YOU WANT TO INSTALL IT ONTO A FABRIC PEER?

Fabric EVM chaincode (evmcc) makes it possible to execute EVM based instructions on Hyperledger Fabric. This means that you can interact with Fabric in exactly the same way you would interact with Ethereum. If you'd like to know more about how this works, you can read all about it in *this article*. Without going into too much detail, the chaincode imports components from Hyperledger Burrow (a new proof-of-authority EVM ledger) which is able to interpret the Solidity code stored on the Fabric ledger. When a Solidity smart contract method is invoked, the code is fetched from the Fabric ledger and loaded into memory, then the Burrow EVM is called to interpret and execute the invoked method.

CHECK PREREQUISITES

Before we begin, do a quick sanity check to make sure Fabric 2.1 is installed correctly:

```bash
docker images | grep hyperledger
```

```bash
peer version
```

You should see your peer version is at 2.1 and all images tagged with 2.1 (It is okay to see fabric-ca 1.4.7 as this is the latest release of fabric-ca).

If you don't already have a network running, you can use the sample test network that comes packaged with github.com/hyperledger/fabric-samples. Simply navigate to test-network and execute the following commands to bring up the network and create a channel:

```bash
./network.sh up
./network.sh createChannel <channel_name>
```

Next, clone the Fabric EVM chaincode repository (if you're curious, the actual chaincode can be found in evmcc/evmcc.go):

```bash
git clone https://github.com/hyperledger/fabric-evm-chaincode.git
```

The first thing we are going to do is vendor all of the chaincode dependencies. This will resolve all code dependencies for the evmcc.go module. This will ensure that when we package the chaincode in the next step, we don't exclude any dependencies which would cause the installation to fail at a later step. To do this, run these commands:

```bash
cd fabric-evm-chaincode/evmcc
GO111MODULE=on go mod vendor
```

CONFIGURE THE CORE PEER

Next, we need to configure the peer CLI to operate on the core peer for your organization. To do this, select a peer that you want to act as the core peer (this is an arbitrary selection), then define the following environment variables:

(NOTE: This is just an example!! You'll need to set your peer accordingly.)

```bash
export FABRIC_CFG_PATH=<path to directory containing the core.yaml configuration>
export CORE_PEER_TLS_ENABLED=<true | false>
export CORE_PEER_LOCALMSPID=<your org's MSP ID>
export CORE_PEER_TLS_ROOTCERT_FILE=<path to core peer's TLS CA cert>
export CORE_PEER_MSPCONFIGPATH=<path to core peer admin user's MSP>
export CORE_PEER_ADDRESS=<peer address>
export ORDERER_CA=<path to orderer's /msp/tlscacerts/tlsca.pem file>
```

(HINT: If you are using the sample-network, then you will need to create two environment profiles! Each of the two organizations will have its own core peer. My suggestion would be to create a profile for each peer so you can easily switch between environments. This way you can run the peer cli command, switch profiles, and run the same command again to target the other peer.)

PACKAGE THE CHAINCODE

Navigate to fabric-evm-chaincode/evmcc directory (if not already there) and package the chaincode into a tar.gz file:

```bash
peer lifecycle chaincode package evmcc.tar.gz -p . -l golang --label evmcc_1
```

Please note the label name is `evmcc_1` with the number 1 at the end. In a later step, we will specify the sequence number for this chaincode definition. The sequence must start at 1, so to be clear, we will label our chaincode using the format <name_sequence>.

(If you're curious about what a Fabric lifecycle is and would like to know more, please read *this article*. This may be very useful to know if in the future you would like to upgrade the chaincode or install different versions.)

INSTALL THE CHAINCODE

```bash
peer lifecycle chaincode install evmcc.tar.gz
```

This will install the chaincode at version 1.

(HINT: Again, if you are on the sample-network, switch to the other profile for the other core peer and run the above command again! Be careful to switch profiles each time you do this!)

QUERY THE PEER FOR THE PACKAGE ID.

We will need this package ID in subsequent steps.

```bash
PACKAGE_ID=$(peer lifecycle chaincode queryinstalled | sed -n "/evmcc_1/{s/^Package ID: //; s/,
Label:.*$//; p;}")
```

APPROVE THE CHAINCODE FOR YOUR ORGANIZATION:

Your organization must first approve the chaincode before it can be installed on any one of its peers. This is true for all chaincode installations.

```bash
peer lifecycle chaincode approveformyorg -o <Orderer's Address> --ordererTLSHostnameOverride <Orderer's Hostname> --tls --cafile $ORDERER_CA --channelID <channel name> --name evmcc --version 1 --init-required --package-id ${PACKAGE_ID} --sequence 1
```

Switch profiles and run the above command again for the second organization.

CHECK TO SEE THAT THE CHAINCODE IS READY TO BE COMMITTED TO THE PEER:

```bash
peer lifecycle chaincode checkcommitreadiness --channelID <channel_name> --name evmcc --version 1 --sequence 1 --output json --init-required
```

If all is going well so far, you should see that your organization has approved the chaincode to be committed to the ledger.

CONFIGURE THE TARGET PEERS TO RECEIVE THE CHAINCODE:

```bash
export PEER_CONN_PARMS="--peerAddresses <peer Address> --tlsRootCertFiles <path to peer's tls/ca.crt>"
```

(NOTE: If using sample-network, append the address and root cert files of the second peer. You will have two --peerAddresses, and two --tlsRootCertFiles)

COMMIT THE CHAINCODE

After you have targeted the peers that will receive the chaincode, go ahead and commit the chaincode:

```bash
peer lifecycle chaincode commit -o <orderer Address> --ordererTLSHostnameOverride <orderer hostname> --tls --cafile $ORDERER_CA --channelID <channel_name> --name evmcc $PEER_CONN_PARMS --version 1 --sequence 1 --init-required
```

QUERY THE PEER(S) TO CHECK IF THE CHAINCODE HAS BEEN SUCCESSFULLY COMMITTED:

```bash
peer lifecycle chaincode querycommitted --channelID <channel_name> --name evmcc
```

INSTANTIATE THE CHAINCODE

Finally, invoke the chaincode's constructor to instantiate itself. After this step, the chaincode should be fully deployed and ready to go:

```bash
peer chaincode invoke -o <orderer Address> --ordererTLSHostnameOverride <orderer hostname> --tls --cafile $ORDERER_CA -C <channel_name> -n evmcc $PEER_CONN_PARMS --isInit -c '{"Args":[]}'
```

To test if all went well, you follow *this guide*. Skip ahead past installing the evmcc and pick up where it begins to create a contract and make smart method calls. Since the instructions were written for the 1.4 network, you will need to modify the commands slightly to work for 2.1. The main difference is that you'll need to target the endorsing peers. Here is an example of how to deploy a smart contract for 2.1:

(NOTE: This is just an example, you'll need to set your peer's cert file and address accordingly.)

```bash
peer chaincode invoke -o <orderer address> --ordererTLSHostnameOverride <orderer host> --tls --cafile
$ORDERER_CA -C mychannel -n evmcc  --peerAddresses localhost:7051 --tlsRootCertFiles $PEER0_ORG1_CA
--peerAddresses localhost:9051 --tlsRootCertFiles $PEER0_ORG2_CA -c '{"Args":["000000000000000000000000
0000000000000000","608060405234801561001057600080fd5b5060df8061001f6000396000f300608060405260043610604 9
5760003 57c0100000000000000000000000000000000000000000000000000000000000900463ffffffff16806360fe47b114604e
5780636d4ce63c146078575b600080fd5b348015605957600080fd5b506076600480360381019080803590602001909291905 05
05060a0565b005b34801560835760080fd5b50608a60aa565b6040518082815260200191505060405180910390f35b80600081
90555050565b600080549050905600a165627a7a723058203dbaed52da8059a841ed6d7b484bf6fa6f61a7e975a803fdedf076a
121a8c4010029"]}'
```

If you were able to create and deploy a smart contract using Fabric EVM, then congratulations! You've now successfully installed the chaincode onto version 2.1 of Hyperledger's Fabric.

WHERE SHOULD YOU GO FROM HERE?

If you want to be able to use web3 tools to interact with Fabric EVM then we would suggest building and configuring the Fab3 proxy and connecting this to the same anchor peer that has the installed chaincode. The Fab3 proxy accepts as input gRPC API calls exactly as you would send them to an ethereum peer, and translates these calls into peer cli commands. In this way you can treat the Fab3 proxy as an ethereum peer.

SIMBA Chain enables seamless utilization and integration of blockchain technology to bolster trust, security, and risk mitigation for enterprise and government

SIMBA

simbachain.com
574-914-4446
info@simbachain.com