# Commands Reference

* [peer](https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/peercommand.html)
* [peer chaincode](https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/peerchaincode.html)
* [peer channel](https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/peerchannel.html)
* [peer version](https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/peerversion.html)
* [peer logging](https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/peerlogging.html)
* [peer node](https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/peernode.html)
* [configtxgen](https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/configtxgen.html)
* [configtxlator](https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/configtxlator.html)
* [cryptogen](https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/cryptogen.html)
* [Service Discovery CLI](https://hyperledger-fabric.readthedocs.io/en/release-1.4/discovery-cli.html)
* [Fabric-CA Commands](https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/fabric-ca-commands.html)

# peer

## Description

The peer command has five different subcommands, each of which allows administrators to perform a specific set of tasks related to a peer. For example, you can use the peer channel subcommand to join a peer to a channel, or the peer chaincode command to deploy a smart contract chaincode to a peer.

## Syntax

The peer command has five different subcommands within it:

peer chaincode [option] [flags]

peer channel [option] [flags]

peer logging [option] [flags]

peer node [option] [flags]

peer version [option] [flags]

Each subcommand has different options available, and these are described in their own dedicated topic. For brevity, we often refer to a command (peer), a subcommand (channel), or subcommand option (fetch) simply as a **command**.

If a subcommand is specified without an option, then it will return some high level help text as described in the --help flag below.

## Flags

Each peer subcommand has a specific set of flags associated with it, many of which are designated global because they can be used in all subcommand options. These flags are described with the relevant peer subcommand.

The top level peer command has the following flag:

* --help

Use --help to get brief help text for any peer command. The --help flag is very useful – it can be used to get command help, subcommand help, and even option help.

For example

peer **--**help

peer channel **--**help

peer channel list **--**help

See individual peer subcommands for more detail.

## Usage

Here is an example using the available flag on the peer command.

* Using the --help flag on the peer channel join command.
* peer channel join **--**help
* Joins the peer to a channel**.**
* Usage:
* peer channel join [flags]
* Flags:
* **-**b, **--**blockpath string Path to file containing genesis block
* **-**h, **--**help help **for** join
* Global Flags:
* **--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint
* **--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint
* **--**clientauth Use mutual TLS when communicating **with** the orderer endpoint
* **--**connTimeout duration Timeout **for** client to [connect](https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/peercommand.html) (default 3s)
* **--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint
* **-**o, **--**orderer string Ordering service endpoint
* **--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**
* **--**tls Use TLS when communicating **with** the orderer endpoint

This shows brief help syntax for the peer channel join command.

# peer chaincode

The peer chaincode command allows administrators to perform chaincode related operations on a peer, such as installing, instantiating, invoking, packaging, querying, and upgrading chaincode.

## Syntax

The peer chaincode command has the following subcommands:

* install
* instantiate
* invoke
* list
* package
* query
* signpackage
* upgrade

The different subcommand options (install, instantiate…) relate to the different chaincode operations that are relevant to a peer. For example, use the peer chaincode install subcommand option to install a chaincode on a peer, or the peer chaincode query subcommand option to query a chaincode for the current value on a peer’s ledger.

Each peer chaincode subcommand is described together with its options in its own section in this topic.

## Flags

Each peer chaincode subcommand has both a set of flags specific to an individual subcommand, as well as a set of global flags that relate to all peer chaincode subcommands. Not all subcommands would use these flags. For instance, the query subcommand does not need the --orderer flag.

The individual flags are described with the relevant subcommand. The global flags are

* --cafile <string>

Path to file containing PEM-encoded trusted certificate(s) for the ordering endpoint

* --certfile <string>

Path to file containing PEM-encoded X509 public key to use for mutual TLS communication with the orderer endpoint

* --keyfile <string>

Path to file containing PEM-encoded private key to use for mutual TLS communication with the orderer endpoint

* -o or --orderer <string>

Ordering service endpoint specified as <hostname or IP address>:<port>

* --ordererTLSHostnameOverride <string>

The hostname override to use when validating the TLS connection to the orderer

* --tls

Use TLS when communicating with the orderer endpoint

* --transient <string>

Transient map of arguments in JSON encoding

## peer chaincode install

Package the specified chaincode into a deployment spec **and** save it on the peer's path.

Usage:

peer chaincode install [flags]

Flags:

**--**connectionProfile string Connection profile that provides the necessary connection information **for** the network**.** Note: currently only supported **for** providing peer connection information

**-**c, **--**ctor string Constructor message **for** the chaincode **in** JSON format (default "{}")

**-**h, **--**help help **for** install

**-**l, **--**lang string Language the chaincode **is** written **in** (default "golang")

**-**n, **--**name string Name of the chaincode

**-**p, **--**path string Path to chaincode

**--**peerAddresses stringArray The addresses of the peers to connect to

**--**tlsRootCertFiles stringArray If TLS **is** enabled, the paths to the TLS root cert files of the peers to connect to**.** The order **and** number of certs specified should match the **--**peerAddresses flag

**-**v, **--**version string Version of the chaincode specified **in** install**/**instantiate**/**upgrade commands

Global Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

**--**transient string Transient map of arguments **in** JSON encoding

## peer chaincode instantiate

Deploy the specified chaincode to the network**.**

Usage:

peer chaincode instantiate [flags]

Flags:

**-**C, **--**channelID string The channel on which this command should be executed

**--**collections**-**config string The fully qualified path to the collection JSON file including the file name

**--**connectionProfile string Connection profile that provides the necessary connection information **for** the network**.** Note: currently only supported **for** providing peer connection information

**-**c, **--**ctor string Constructor message **for** the chaincode **in** JSON format (default "{}")

**-**E, **--**escc string The name of the endorsement system chaincode to be used **for** this chaincode

**-**h, **--**help help **for** instantiate

**-**l, **--**lang string Language the chaincode **is** written **in** (default "golang")

**-**n, **--**name string Name of the chaincode

**--**peerAddresses stringArray The addresses of the peers to connect to

**-**P, **--**policy string The endorsement policy associated to this chaincode

**--**tlsRootCertFiles stringArray If TLS **is** enabled, the paths to the TLS root cert files of the peers to connect to**.** The order **and** number of certs specified should match the **--**peerAddresses flag

**-**v, **--**version string Version of the chaincode specified **in** install**/**instantiate**/**upgrade commands

**-**V, **--**vscc string The name of the verification system chaincode to be used **for** this chaincode

Global Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

**--**transient string Transient map of arguments **in** JSON encoding

## peer chaincode invoke

Invoke the specified chaincode**.** It will **try** to commit the endorsed transaction to the network**.**

Usage:

peer chaincode invoke [flags]

Flags:

**-**C, **--**channelID string The channel on which this command should be executed

**--**connectionProfile string Connection profile that provides the necessary connection information **for** the network**.** Note: currently only supported **for** providing peer connection information

**-**c, **--**ctor string Constructor message **for** the chaincode **in** JSON format (default "{}")

**-**h, **--**help help **for** invoke

**-**n, **--**name string Name of the chaincode

**--**peerAddresses stringArray The addresses of the peers to connect to

**--**tlsRootCertFiles stringArray If TLS **is** enabled, the paths to the TLS root cert files of the peers to connect to**.** The order **and** number of certs specified should match the **--**peerAddresses flag

**--**waitForEvent Whether to wait **for** the event **from** each peer's deliver filtered service signifying that the 'invoke' transaction has been committed successfully

**--**waitForEventTimeout duration Time to wait **for** the event **from** each peer's deliver filtered service signifying that the 'invoke' transaction has been committed successfully (default 30s)

Global Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

**--**transient string Transient map of arguments **in** JSON encoding

## peer chaincode list

Get the instantiated chaincodes **in** the channel **if** specify channel, **or** get installed chaincodes on the peer

Usage:

peer chaincode list [flags]

Flags:

**-**C, **--**channelID string The channel on which this command should be executed

**--**connectionProfile string Connection profile that provides the necessary connection information **for** the network**.** Note: currently only supported **for** providing peer connection information

**-**h, **--**help help **for** list

**--**installed Get the installed chaincodes on a peer

**--**instantiated Get the instantiated chaincodes on a channel

**--**peerAddresses stringArray The addresses of the peers to connect to

**--**tlsRootCertFiles stringArray If TLS **is** enabled, the paths to the TLS root cert files of the peers to connect to**.** The order **and** number of certs specified should match the **--**peerAddresses flag

Global Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

**--**transient string Transient map of arguments **in** JSON encoding

## peer chaincode package

Package the specified chaincode into a deployment spec**.**

Usage:

peer chaincode package [flags]

Flags:

**-**s, **--**cc**-**package create CC deployment spec **for** owner endorsements instead of raw CC deployment spec

**-**c, **--**ctor string Constructor message **for** the chaincode **in** JSON format (default "{}")

**-**h, **--**help help **for** package

**-**i, **--**instantiate**-**policy string instantiation policy **for** the chaincode

**-**l, **--**lang string Language the chaincode **is** written **in** (default "golang")

**-**n, **--**name string Name of the chaincode

**-**p, **--**path string Path to chaincode

**-**S, **--**sign **if** creating CC deployment spec package **for** owner endorsements, also sign it **with** local MSP

**-**v, **--**version string Version of the chaincode specified **in** install**/**instantiate**/**upgrade commands

Global Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

**--**transient string Transient map of arguments **in** JSON encoding

## peer chaincode query

Get endorsed result of chaincode function call **and** print it**.** It won't generate transaction.

Usage:

peer chaincode query [flags]

Flags:

**-**C, **--**channelID string The channel on which this command should be executed

**--**connectionProfile string Connection profile that provides the necessary connection information **for** the network**.** Note: currently only supported **for** providing peer connection information

**-**c, **--**ctor string Constructor message **for** the chaincode **in** JSON format (default "{}")

**-**h, **--**help help **for** query

**-**x, **--**hex If true, output the query value byte array **in** hexadecimal**.** Incompatible **with** **--**raw

**-**n, **--**name string Name of the chaincode

**--**peerAddresses stringArray The addresses of the peers to connect to

**-**r, **--**raw If true, output the query value **as** raw bytes, otherwise format **as** a printable string

**--**tlsRootCertFiles stringArray If TLS **is** enabled, the paths to the TLS root cert files of the peers to connect to**.** The order **and** number of certs specified should match the **--**peerAddresses flag

Global Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

**--**transient string Transient map of arguments **in** JSON encoding

## peer chaincode signpackage

Sign the specified chaincode package

Usage:

peer chaincode signpackage [flags]

Flags:

**-**h, **--**help help **for** signpackage

Global Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

**--**transient string Transient map of arguments **in** JSON encoding

## peer chaincode upgrade

Upgrade an existing chaincode **with** the specified one**.** The new chaincode will immediately replace the existing chaincode upon the transaction committed**.**

Usage:

peer chaincode upgrade [flags]

Flags:

**-**C, **--**channelID string The channel on which this command should be executed

**--**collections**-**config string The fully qualified path to the collection JSON file including the file name

**--**connectionProfile string Connection profile that provides the necessary connection information **for** the network**.** Note: currently only supported **for** providing peer connection information

**-**c, **--**ctor string Constructor message **for** the chaincode **in** JSON format (default "{}")

**-**E, **--**escc string The name of the endorsement system chaincode to be used **for** this chaincode

**-**h, **--**help help **for** upgrade

**-**l, **--**lang string Language the chaincode **is** written **in** (default "golang")

**-**n, **--**name string Name of the chaincode

**-**p, **--**path string Path to chaincode

**--**peerAddresses stringArray The addresses of the peers to connect to

**-**P, **--**policy string The endorsement policy associated to this chaincode

**--**tlsRootCertFiles stringArray If TLS **is** enabled, the paths to the TLS root cert files of the peers to connect to**.** The order **and** number of certs specified should match the **--**peerAddresses flag

**-**v, **--**version string Version of the chaincode specified **in** install**/**instantiate**/**upgrade commands

**-**V, **--**vscc string The name of the verification system chaincode to be used **for** this chaincode

Global Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

**--**transient string Transient map of arguments **in** JSON encoding

## Example Usage

### peer chaincode instantiate examples

Here are some examples of the peer chaincode instantiate command, which instantiates the chaincode named mycc at version 1.0 on channel mychannel:

* Using the --tls and --cafile global flags to instantiate the chaincode in a network with TLS enabled:
* export ORDERER\_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
* peer chaincode instantiate -o orderer.example.com:7050 --tls --cafile $ORDERER\_CA -C mychannel -n mycc -v 1.0 -c '{"Args":["init","a","100","b","200"]}' -P "AND ('Org1MSP.peer','Org2MSP.peer')"
* 2018-02-22 16:33:53.324 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
* 2018-02-22 16:33:53.324 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
* 2018-02-22 16:34:08.698 UTC [main] main -> INFO 003 Exiting.....
* Using only the command-specific options to instantiate the chaincode in a network with TLS disabled:
* peer chaincode instantiate **-**o orderer**.**example**.**com:7050 **-**C mychannel **-**n mycc **-**v 1.0 **-**c '{"Args":["init","a","100","b","200"]}' **-**P "AND ('Org1MSP.peer','Org2MSP.peer')"
* 2018**-**02**-**22 16:34:09.324 UTC [chaincodeCmd] checkChaincodeCmdParams **->** INFO 001 Using default escc
* 2018**-**02**-**22 16:34:09.324 UTC [chaincodeCmd] checkChaincodeCmdParams **->** INFO 002 Using default vscc
* 2018**-**02**-**22 16:34:24.698 UTC [main] main **->** INFO 003 Exiting**.....**

### peer chaincode invoke example

Here is an example of the peer chaincode invoke command:

* Invoke the chaincode named mycc at version 1.0 on channel mychannel on peer0.org1.example.com:7051 and peer0.org2.example.com:9051 (the peers defined by --peerAddresses), requesting to move 10 units from variable a to variable b:
* peer chaincode invoke **-**o orderer**.**example**.**com:7050 **-**C mychannel **-**n mycc **--**peerAddresses peer0**.**org1**.**example**.**com:7051 **--**peerAddresses peer0**.**org2**.**example**.**com:9051 **-**c '{"Args":["invoke","a","b","10"]}'
* 2018**-**02**-**22 16:34:27.069 UTC [chaincodeCmd] checkChaincodeCmdParams **->** INFO 001 Using default escc
* 2018**-**02**-**22 16:34:27.069 UTC [chaincodeCmd] checkChaincodeCmdParams **->** INFO 002 Using default vscc
* **.**
* **.**
* **.**
* 2018**-**02**-**22 16:34:27.106 UTC [chaincodeCmd] chaincodeInvokeOrQuery **->** DEBU 00a ESCC invoke result: version:1 response:**<**status:200 message:"OK" **>** payload:"\n \237mM\376? [\214\002 \332\204\035\275q\227\2132A\n\204&\2106\037W|\346#\3413\274\022Y\nE\022\024\n\004lscc\022\014\n\n\n\004mycc\022\002\010\003\022-\n\004mycc\022%\n\007\n\001a\022\002\010\003\n\007\n\001b\022\002\010\003\032\007\n\001a\032\00290\032\010\n\001b\032\003210\032\003\010\310\001\"\013\022\004mycc\032\0031.0" endorsement:**<**endorser:"\n\007Org1MSP\022\262\006-----BEGIN CERTIFICATE-----\nMIICLjCCAdWgAwIBAgIRAJYomxY2cqHA/fbRnH5a/bwwCgYIKoZIzj0EAwIwczEL\nMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcTDVNhbiBG\ncmFuY2lzY28xGTAXBgNVBAoTEG9yZzEuZXhhbXBsZS5jb20xHDAaBgNVBAMTE2Nh\nLm9yZzEuZXhhbXBsZS5jb20wHhcNMTgwMjIyMTYyODE0WhcNMjgwMjIwMTYyODE0\nWjBwMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMN\nU2FuIEZyYW5jaXNjbzETMBEGA1UECxMKRmFicmljUGVlcjEfMB0GA1UEAxMWcGVl\ncjAub3JnMS5leGFtcGxlLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABDEa\nWNNniN3qOCQL89BGWfY39f5V3o1pi//7JFDHATJXtLgJhkK5KosDdHuKLYbCqvge\n46u3AC16MZyJRvKBiw6jTTBLMA4GA1UdDwEB/wQEAwIHgDAMBgNVHRMBAf8EAjAA\nMCsGA1UdIwQkMCKAIN7dJR9dimkFtkus0R5pAOlRz5SA3FB5t8Eaxl9A7lkgMAoG\nCCqGSM49BAMCA0cAMEQCIC2DAsO9QZzQmKi8OOKwcCh9Gd01YmWIN3oVmaCRr8C7\nAiAlQffq2JFlbh6OWURGOko6RckizG8oVOldZG/Xj3C8lA==\n-----END CERTIFICATE-----\n" signature:"0D\002 \022\_\342\350\344\231G&\237\n\244\375\302J\220l\302\345\210\335D\250y\253P\0214:\221e\332@\002 \000\254\361\224\247\210\214L\277\370\222\213\217\301\r\341v\227\265\277\336\256^\217\336\005y\*\321\023\025\367" **>**
* 2018**-**02**-**22 16:34:27.107 UTC [chaincodeCmd] chaincodeInvokeOrQuery **->** INFO 00b Chaincode invoke successful**.** result: status:200
* 2018**-**02**-**22 16:34:27.107 UTC [main] main **->** INFO 00c Exiting**.....**

Here you can see that the invoke was submitted successfully based on the log message:

2018**-**02**-**22 16:34:27.107 UTC [chaincodeCmd] chaincodeInvokeOrQuery **->** INFO 00b Chaincode invoke successful**.** result: status:200

A successful response indicates that the transaction was submitted for ordering successfully. The transaction will then be added to a block and, finally, validated or invalidated by each peer on the channel.

### peer chaincode list example

Here are some examples of the peer chaincode list command:

* Using the --installed flag to list the chaincodes installed on a peer.
* peer chaincode list **--**installed
* Get installed chaincodes on peer:
* Name: mycc, Version: 1.0, Path: github**.**com**/**hyperledger**/**fabric**/**examples**/**chaincode**/**go**/**chaincode\_example02, Id: 8cc2730fdafd0b28ef734eac12b29df5fc98ad98bdb1b7e0ef96265c3d893d61
* 2018**-**02**-**22 17:07:13.476 UTC [main] main **->** INFO 001 Exiting**.....**

You can see that the peer has installed a chaincode called mycc which is at version 1.0.

* Using the --instantiated in combination with the -C (channel ID) flag to list the chaincodes instantiated on a channel.
* peer chaincode list **--**instantiated **-**C mychannel
* Get instantiated chaincodes on channel mychannel:
* Name: mycc, Version: 1.0, Path: github**.**com**/**hyperledger**/**fabric**/**examples**/**chaincode**/**go**/**chaincode\_example02, Escc: escc, Vscc: vscc
* 2018**-**02**-**22 17:07:42.969 UTC [main] main **->** INFO 001 Exiting**.....**

You can see that chaincode mycc at version 1.0 is instantiated on channel mychannel.

### peer chaincode package example

Here is an example of the peer chaincode package command, which packages the chaincode named mycc at version 1.1, creates the chaincode deployment spec, signs the package using the local MSP, and outputs it as ccpack.out:

peer chaincode package ccpack**.**out **-**n mycc **-**p github**.**com**/**hyperledger**/**fabric**/**examples**/**chaincode**/**go**/**chaincode\_example02 **-**v 1.1 **-**s **-**S

**.**

**.**

**.**

2018**-**02**-**22 17:27:01.404 UTC [chaincodeCmd] checkChaincodeCmdParams **->** INFO 003 Using default escc

2018**-**02**-**22 17:27:01.405 UTC [chaincodeCmd] checkChaincodeCmdParams **->** INFO 004 Using default vscc

**.**

**.**

**.**

2018**-**02**-**22 17:27:01.879 UTC [chaincodeCmd] chaincodePackage **->** DEBU 011 Packaged chaincode into deployment spec of size **<**3426**>**, **with** args **=** [ccpack**.**out]

2018**-**02**-**22 17:27:01.879 UTC [main] main **->** INFO 012 Exiting**.....**

### peer chaincode query example

Here is an example of the peer chaincode query command, which queries the peer ledger for the chaincode named mycc at version 1.0 for the value of variable a:

* You can see from the output that variable a had a value of 90 at the time of the query.
* peer chaincode query **-**C mychannel **-**n mycc **-**c '{"Args":["query","a"]}'
* 2018**-**02**-**22 16:34:30.816 UTC [chaincodeCmd] checkChaincodeCmdParams **->** INFO 001 Using default escc
* 2018**-**02**-**22 16:34:30.816 UTC [chaincodeCmd] checkChaincodeCmdParams **->** INFO 002 Using default vscc
* Query Result: 90

### peer chaincode signpackage example

Here is an example of the peer chaincode signpackage command, which accepts an existing signed package and creates a new one with signature of the local MSP appended to it.

peer chaincode signpackage ccwith1sig**.**pak ccwith2sig**.**pak

Wrote signed package to ccwith2sig**.**pak successfully

2018**-**02**-**24 19:32:47.189 EST [main] main **->** INFO 002 Exiting**.....**

### peer chaincode upgrade example

Here is an example of the peer chaincode upgrade command, which upgrades the chaincode named mycc at version 1.0 on channel mychannel to version 1.1, which contains a new variable c:

* Using the --tls and --cafile global flags to upgrade the chaincode in a network with TLS enabled:
* export ORDERER\_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
* peer chaincode upgrade -o orderer.example.com:7050 --tls --cafile $ORDERER\_CA -C mychannel -n mycc -v 1.2 -c '{"Args":["init","a","100","b","200"]}' -P "AND ('Org1MSP.peer','Org2MSP.peer')"
* .
* .
* .
* 2018-02-22 18:26:31.433 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 003 Using default escc
* 2018-02-22 18:26:31.434 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 004 Using default vscc
* 2018-02-22 18:26:31.435 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 005 java chaincode enabled
* 2018-02-22 18:26:31.435 UTC [chaincodeCmd] upgrade -> DEBU 006 Get upgrade proposal for chaincode <name:"mycc" version:"1.1" >
* .
* .
* .
* 2018-02-22 18:26:46.687 UTC [chaincodeCmd] upgrade -> DEBU 009 endorse upgrade proposal, get response <status:200 message:"OK" payload:"\n\004mycc\022\0031.1\032\004escc\"\004vscc\*,\022\014\022\n\010\001\022\002\010\000\022\002\010\001\032\r\022\013\n\007Org1MSP\020\003\032\r\022\013\n\007Org2MSP\020\0032f\n \261g(^v\021\220\240\332\251\014\204V\210P\310o\231\271\036\301\022\032\205fC[|=\215\372\223\022 \311b\025?\323N\343\325\032\005\365\236\001XKj\004E\351\007\247\265fu\305j\367\331\275\253\307R\032 \014H#\014\272!#\345\306s\323\371\350\364\006.\000\356\230\353\270\263\215\217\303\256\220i^\277\305\214: \375\200zY\275\203}\375\244\205\035\340\226]l!uE\334\273\214\214\020\303\3474\360\014\234-\006\315B\031\022\010\022\006\010\001\022\002\010\000\032\r\022\013\n\007Org1MSP\020\001" >
* .
* .
* .
* 2018-02-22 18:26:46.693 UTC [chaincodeCmd] upgrade -> DEBU 00c Get Signed envelope
* 2018-02-22 18:26:46.693 UTC [chaincodeCmd] chaincodeUpgrade -> DEBU 00d Send signed envelope to orderer
* 2018-02-22 18:26:46.908 UTC [main] main -> INFO 00e Exiting.....
* Using only the command-specific options to upgrade the chaincode in a network with TLS disabled:
* peer chaincode upgrade **-**o orderer**.**example**.**com:7050 **-**C mychannel **-**n mycc **-**v 1.2 **-**c '{"Args":["init","a","100","b","200"]}' **-**P "AND ('Org1MSP.peer','Org2MSP.peer')"
* **.**
* **.**
* **.**
* 2018**-**02**-**22 18:28:31.433 UTC [chaincodeCmd] checkChaincodeCmdParams **->** INFO 003 Using default escc
* 2018**-**02**-**22 18:28:31.434 UTC [chaincodeCmd] checkChaincodeCmdParams **->** INFO 004 Using default vscc
* 2018**-**02**-**22 18:28:31.435 UTC [chaincodeCmd] getChaincodeSpec **->** DEBU 005 java chaincode enabled
* 2018**-**02**-**22 18:28:31.435 UTC [chaincodeCmd] upgrade **->** DEBU 006 Get upgrade proposal **for** chaincode **<**name:"mycc" version:"1.1" **>**
* **.**
* **.**
* **.**
* 2018**-**02**-**22 18:28:46.687 UTC [chaincodeCmd] upgrade **->** DEBU 009 endorse upgrade proposal, get response **<**status:200 message:"OK" payload:"\n\004mycc\022\0031.1\032\004escc\"\004vscc\*,\022\014\022\n\010\001\022\002\010\000\022\002\010\001\032\r\022\013\n\007Org1MSP\020\003\032\r\022\013\n\007Org2MSP\020\0032f\n \261g(^v\021\220\240\332\251\014\204V\210P\310o\231\271\036\301\022\032\205fC[|=\215\372\223\022 \311b\025?\323N\343\325\032\005\365\236\001XKj\004E\351\007\247\265fu\305j\367\331\275\253\307R\032 \014H#\014\272!#\345\306s\323\371\350\364\006.\000\356\230\353\270\263\215\217\303\256\220i^\277\305\214: \375\200zY\275\203}\375\244\205\035\340\226]l!uE\334\273\214\214\020\303\3474\360\014\234-\006\315B\031\022\010\022\006\010\001\022\002\010\000\032\r\022\013\n\007Org1MSP\020\001" **>**
* **.**
* **.**
* **.**
* 2018**-**02**-**22 18:28:46.693 UTC [chaincodeCmd] upgrade **->** DEBU 00c Get Signed envelope
* 2018**-**02**-**22 18:28:46.693 UTC [chaincodeCmd] chaincodeUpgrade **->** DEBU 00d Send signed envelope to orderer
* 2018**-**02**-**22 18:28:46.908 UTC [main] main **->** INFO 00e Exiting**.....**

[Creative Commons License](http://creativecommons.org/licenses/by/4.0/)  
This work is licensed under a [Creative Commons Attribution 4.0 International License](http://creativecommons.org/licenses/by/4.0/).

# peer channel

The peer channel command allows administrators to perform channel related operations on a peer, such as joining a channel or listing the channels to which a peer is joined.

## Syntax

The peer channel command has the following subcommands:

* create
* fetch
* getinfo
* join
* list
* signconfigtx
* update

## peer channel

Operate a channel: create**|**fetch**|**join**|**list**|**update**|**signconfigtx**|**getinfo**.**

Usage:

peer channel [command]

Available Commands:

create Create a channel

fetch Fetch a block

getinfo get blockchain information of a specified channel**.**

join Joins the peer to a channel**.**

list List of channels peer has joined**.**

signconfigtx Signs a configtx update**.**

update Send a configtx update**.**

Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**-**h, **--**help help **for** channel

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

Use "peer channel [command] --help" **for** more information about a command**.**

## peer channel create

Create a channel **and** write the genesis block to a file**.**

Usage:

peer channel create [flags]

Flags:

**-**c, **--**channelID string In case of a newChain command, the channel ID to create**.** It must be all lower case, less than 250 characters long **and** match the regular expression: [a**-**z][a**-**z0**-**9.**-**]**\***

**-**f, **--**file string Configuration transaction file generated by a tool such **as** configtxgen **for** submitting to orderer

**-**h, **--**help help **for** create

**--**outputBlock string The path to write the genesis block **for** the channel**.** (default **./<**channelID**>.**block)

**-**t, **--**timeout duration Channel creation timeout (default 10s)

Global Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

## peer channel fetch

Fetch a specified block, writing it to a file**.**

Usage:

peer channel fetch **<**newest**|**oldest**|**config**|**(number)**>** [outputfile] [flags]

Flags:

**--**bestEffort Whether fetch requests should ignore errors **and** **return** blocks on a best effort basis

**-**c, **--**channelID string In case of a newChain command, the channel ID to create**.** It must be all lower case, less than 250 characters long **and** match the regular expression: [a**-**z][a**-**z0**-**9.**-**]**\***

**-**h, **--**help help **for** fetch

Global Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

## peer channel getinfo

get blockchain information of a specified channel**.** Requires '-c'**.**

Usage:

peer channel getinfo [flags]

Flags:

**-**c, **--**channelID string In case of a newChain command, the channel ID to create**.** It must be all lower case, less than 250 characters long **and** match the regular expression: [a**-**z][a**-**z0**-**9.**-**]**\***

**-**h, **--**help help **for** getinfo

Global Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

## peer channel join

Joins the peer to a channel**.**

Usage:

peer channel join [flags]

Flags:

**-**b, **--**blockpath string Path to file containing genesis block

**-**h, **--**help help **for** join

Global Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

## peer channel list

List of channels peer has joined**.**

Usage:

peer channel list [flags]

Flags:

**-**h, **--**help help **for** list

Global Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

## peer channel signconfigtx

Signs the supplied configtx update file **in** place on the filesystem**.** Requires '-f'**.**

Usage:

peer channel signconfigtx [flags]

Flags:

**-**f, **--**file string Configuration transaction file generated by a tool such **as** configtxgen **for** submitting to orderer

**-**h, **--**help help **for** signconfigtx

Global Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

## peer channel update

Signs **and** sends the supplied configtx update file to the channel**.** Requires '-f', '-o', '-c'**.**

Usage:

peer channel update [flags]

Flags:

**-**c, **--**channelID string In case of a newChain command, the channel ID to create**.** It must be all lower case, less than 250 characters long **and** match the regular expression: [a**-**z][a**-**z0**-**9.**-**]**\***

**-**f, **--**file string Configuration transaction file generated by a tool such **as** configtxgen **for** submitting to orderer

**-**h, **--**help help **for** update

Global Flags:

**--**cafile string Path to file containing PEM**-**encoded trusted certificate(s) **for** the ordering endpoint

**--**certfile string Path to file containing PEM**-**encoded X509 public key to use **for** mutual TLS communication **with** the orderer endpoint

**--**clientauth Use mutual TLS when communicating **with** the orderer endpoint

**--**connTimeout duration Timeout **for** client to connect (default 3s)

**--**keyfile string Path to file containing PEM**-**encoded private key to use **for** mutual TLS communication **with** the orderer endpoint

**-**o, **--**orderer string Ordering service endpoint

**--**ordererTLSHostnameOverride string The hostname override to use when validating the TLS connection to the orderer**.**

**--**tls Use TLS when communicating **with** the orderer endpoint

## Example Usage

### peer channel create examples

Here’s an example that uses the --orderer global flag on the peer channel create command.

* Create a sample channel mychannel defined by the configuration transaction contained in file ./createchannel.tx. Use the orderer at orderer.example.com:7050.
* peer channel create **-**c mychannel **-**f **./**createchannel**.**tx **--**orderer orderer**.**example**.**com:7050
* 2018**-**02**-**25 08:23:57.548 UTC [channelCmd] InitCmdFactory **->** INFO 003 Endorser **and** orderer connections initialized
* 2018**-**02**-**25 08:23:57.626 UTC [channelCmd] InitCmdFactory **->** INFO 019 Endorser **and** orderer connections initialized
* 2018**-**02**-**25 08:23:57.834 UTC [channelCmd] readBlock **->** INFO 020 Received block: 0
* 2018**-**02**-**25 08:23:57.835 UTC [main] main **->** INFO 021 Exiting**.....**

Block 0 is returned indicating that the channel has been successfully created.

Here’s an example of the peer channel create command option.

* Create a new channel mychannel for the network, using the orderer at ip address orderer.example.com:7050. The configuration update transaction required to create this channel is defined the file ./createchannel.tx. Wait 30 seconds for the channel to be created.
* peer channel create **-**c mychannel **--**orderer orderer**.**example**.**com:7050 **-**f **./**createchannel**.**tx **-**t 30s
* 2018**-**02**-**23 06:31:58.568 UTC [channelCmd] InitCmdFactory **->** INFO 003 Endorser **and** orderer connections initialized
* 2018**-**02**-**23 06:31:58.669 UTC [channelCmd] InitCmdFactory **->** INFO 019 Endorser **and** orderer connections initialized
* 2018**-**02**-**23 06:31:58.877 UTC [channelCmd] readBlock **->** INFO 020 Received block: 0
* 2018**-**02**-**23 06:31:58.878 UTC [main] main **->** INFO 021 Exiting**.....**
* ls **-**l
* **-**rw**-**r**--**r**--** 1 root root 11982 Feb 25 12:24 mychannel**.**block

You can see that channel mychannel has been successfully created, as indicated in the output where block 0 (zero) is added to the blockchain for this channel and returned to the peer, where it is stored in the local directory as mychannel.block.

Block zero is often called the genesis block as it provides the starting configuration for the channel. All subsequent updates to the channel will be captured as configuration blocks on the channel’s blockchain, each of which supersedes the previous configuration.

### peer channel fetch example

Here’s some examples of the peer channel fetch command.

* Using the newest option to retrieve the most recent channel block, and store it in the file mychannel.block.
* peer channel fetch newest mychannel**.**block **-**c mychannel **--**orderer orderer**.**example**.**com:7050
* 2018**-**02**-**25 13:10:16.137 UTC [channelCmd] InitCmdFactory **->** INFO 003 Endorser **and** orderer connections initialized
* 2018**-**02**-**25 13:10:16.144 UTC [channelCmd] readBlock **->** INFO 00a Received block: 32
* 2018**-**02**-**25 13:10:16.145 UTC [main] main **->** INFO 00b Exiting**.....**
* ls **-**l
* **-**rw**-**r**--**r**--** 1 root root 11982 Feb 25 13:10 mychannel**.**block

You can see that the retrieved block is number 32, and that the information has been written to the file mychannel.block.

* Using the (block number) option to retrieve a specific block – in this case, block number 16 – and store it in the default block file.
* peer channel fetch 16 **-**c mychannel **--**orderer orderer**.**example**.**com:7050
* 2018**-**02**-**25 13:46:50.296 UTC [channelCmd] InitCmdFactory **->** INFO 003 Endorser **and** orderer connections initialized
* 2018**-**02**-**25 13:46:50.302 UTC [channelCmd] readBlock **->** INFO 00a Received block: 16
* 2018**-**02**-**25 13:46:50.302 UTC [main] main **->** INFO 00b Exiting**.....**
* ls **-**l
* **-**rw**-**r**--**r**--** 1 root root 11982 Feb 25 13:10 mychannel**.**block
* **-**rw**-**r**--**r**--** 1 root root 4783 Feb 25 13:46 mychannel\_16**.**block

You can see that the retrieved block is number 16, and that the information has been written to the default file mychannel\_16.block.

For configuration blocks, the block file can be decoded using the [configtxlator command](https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/configtxlator.html). See this command for an example of decoded output. User transaction blocks can also be decoded, but a user program must be written to do this.

### peer channel getinfo example

Here’s an example of the peer channel getinfo command.

* Get information about the local peer for channel mychannel.
* peer channel getinfo **-**c mychannel
* 2018**-**02**-**25 15:15:44.135 UTC [channelCmd] InitCmdFactory **->** INFO 003 Endorser **and** orderer connections initialized
* Blockchain info: {"height":5,"currentBlockHash":"JgK9lcaPUNmFb5Mp1qe1SVMsx3o/22Ct4+n5tejcXCw=","previousBlockHash":"f8lZXoAn3gF86zrFq7L1DzW2aKuabH9Ow6SIE5Y04a4="}
* 2018**-**02**-**25 15:15:44.139 UTC [main] main **->** INFO 006 Exiting**.....**

You can see that the latest block for channel mychannel is block 5. You can also see the cryptographic hashes for the most recent blocks in the channel’s blockchain.

### peer channel join example

Here’s an example of the peer channel join command.

* Join a peer to the channel defined in the genesis block identified by the file ./mychannel.genesis.block. In this example, the channel block was previously retrieved by the peer channel fetch command.
* peer channel join **-**b **./**mychannel**.**genesis**.**block
* 2018**-**02**-**25 12:25:26.511 UTC [channelCmd] InitCmdFactory **->** INFO 003 Endorser **and** orderer connections initialized
* 2018**-**02**-**25 12:25:26.571 UTC [channelCmd] executeJoin **->** INFO 006 Successfully submitted proposal to join channel
* 2018**-**02**-**25 12:25:26.571 UTC [main] main **->** INFO 007 Exiting**.....**

You can see that the peer has successfully made a request to join the channel.

### peer channel list example

Here’s an example of the peer channel list command.

* List the channels to which a peer is joined.
* peer channel list
* 2018**-**02**-**25 14:21:20.361 UTC [channelCmd] InitCmdFactory **->** INFO 003 Endorser **and** orderer connections initialized
* Channels peers has joined:
* mychannel
* 2018**-**02**-**25 14:21:20.372 UTC [main] main **->** INFO 006 Exiting**.....**

You can see that the peer is joined to channel mychannel.

### peer channel signconfigtx example

Here’s an example of the peer channel signconfigtx command.

* Sign the channel update transaction defined in the file ./updatechannel.tx. The example lists the configuration transaction file before and after the command.
* ls **-**l
* **-**rw**-**r**--**r**--** 1 anthonyodowd staff 284 25 Feb 18:16 updatechannel**.**tx
* peer channel signconfigtx **-**f updatechannel**.**tx
* 2018**-**02**-**25 18:16:44.456 GMT [channelCmd] InitCmdFactory **->** INFO 001 Endorser **and** orderer connections initialized
* 2018**-**02**-**25 18:16:44.459 GMT [main] main **->** INFO 002 Exiting**.....**
* ls **-**l
* **-**rw**-**r**--**r**--** 1 anthonyodowd staff 2180 25 Feb 18:16 updatechannel**.**tx

You can see that the peer has successfully signed the configuration transaction by the increase in the size of the file updatechannel.tx from 284 bytes to 2180 bytes.

### peer channel update example

Here’s an example of the peer channel update command.

* Update the channel mychannel using the configuration transaction defined in the file ./updatechannel.tx. Use the orderer at ip address orderer.example.com:7050 to send the configuration transaction to all peers in the channel to update their copy of the channel configuration.
* peer channel update **-**c mychannel **-**f **./**updatechannel**.**tx **-**o orderer**.**example**.**com:7050
* 2018**-**02**-**23 06:32:11.569 UTC [channelCmd] InitCmdFactory **->** INFO 003 Endorser **and** orderer connections initialized
* 2018**-**02**-**23 06:32:11.626 UTC [main] main **->** INFO 010 Exiting**.....**

At this point, the channel mychannel has been successfully updated.

[Creative Commons License](http://creativecommons.org/licenses/by/4.0/)  
This work is licensed under a [Creative Commons Attribution 4.0 International License](http://creativecommons.org/licenses/by/4.0/).

# peer version

The peer version command displays the version information of the peer. It displays version, Commit SHA, Go version, OS/architecture, and chaincode information. For example:

peer:

Version: 1.4**.**0

Commit SHA: 0efc897

Go version: go1**.**12.12

OS**/**Arch: linux**/**amd64

Chaincode:

Base Image Version: 0.4**.**14

Base Docker Namespace: hyperledger

Base Docker Label: org**.**hyperledger**.**fabric

Docker Namespace: hyperledger

## Syntax

Print current version of the fabric peer server**.**

Usage:

peer version [flags]

Flags:

**-**h, **--**help help **for** version

[Creative Commons License](http://creativecommons.org/licenses/by/4.0/)  
This work is licensed under a [Creative Commons Attribution 4.0 International License](http://creativecommons.org/licenses/by/4.0/).

# peer logging

The peer logging subcommand allows administrators to dynamically view and configure the log levels of a peer.

## Syntax

The peer logging command has the following subcommands:

* getlogspec
* setlogspec

and the following deprecated subcommands, which will be removed in a future release:

* getlevel
* setlevel
* revertlevels

The different subcommand options (getlogspec, setlogspec, getlevel, setlevel, and revertlevels) relate to the different logging operations that are relevant to a peer.

Each peer logging subcommand is described together with its options in its own section in this topic.

## peer logging

Logging configuration: getlevel**|**setlevel**|**getlogspec**|**setlogspec**|**revertlevels**.**

Usage:

peer logging [command]

Available Commands:

getlevel Returns the logging level of the requested logger**.**

getlogspec Returns the active log spec**.**

revertlevels Reverts the logging spec to the peer's spec at startup.

setlevel Adds the logger **and** log level to the current logging spec**.**

setlogspec Sets the logging spec**.**

Flags:

**-**h, **--**help help **for** logging

Use "peer logging [command] --help" **for** more information about a command**.**

## peer logging getlevel

Returns the logging level of the requested logger**.** Note: the logger name should exactly match the name that **is** displayed **in** the logs**.**

Usage:

peer logging getlevel **<**logger**>** [flags]

Flags:

**-**h, **--**help help **for** getlevel

## peer logging revertlevels

Reverts the logging spec to the peer's spec at startup.

Usage:

peer logging revertlevels [flags]

Flags:

**-**h, **--**help help **for** revertlevels

## peer logging setlevel

Adds the logger **and** log level to the current logging specification**.**

Usage:

peer logging setlevel **<**logger**>** **<**log level**>** [flags]

Flags:

**-**h, **--**help help **for** setlevel

## Example Usage

### Get Level Usage

Here is an example of the peer logging getlevel command:

* To get the log level for logger peer:
* peer logging getlevel peer
* 2018**-**11**-**01 14:18:11.276 UTC [cli**.**logging] getLevel **->** INFO 001 Current log level **for** logger 'peer': INFO

### Get Log Spec Usage

Here is an example of the peer logging getlogspec command:

* To get the active logging spec for the peer:
* peer logging getlogspec
* 2018**-**11**-**01 14:21:03.591 UTC [cli**.**logging] getLogSpec **->** INFO 001 Current logging spec: info

### Set Level Usage

Here are some examples of the peer logging setlevel command:

* To set the log level for loggers matching logger name prefix gossip to log level WARNING:
* peer logging setlevel gossip warning
* 2018**-**11**-**01 14:21:55.509 UTC [cli**.**logging] setLevel **->** INFO 001 Log level set **for** logger name**/**prefix 'gossip': WARNING
* To set the log level to ERROR for only the logger that exactly matches the supplied name, append a period to the logger name:
* peer logging setlevel gossip**.** error
* 2018**-**11**-**01 14:27:33.080 UTC [cli**.**logging] setLevel **->** INFO 001 Log level set **for** logger name**/**prefix 'gossip.': ERROR

### Set Log Spec Usage

Here is an example of the peer logging setlogspec command:

* To set the active logging spec for the peer where loggers that begin with gossip and msp are set to log level WARNING and the default for all other loggers is log level INFO:
* peer logging setlogspec gossip**=**warning:msp**=**warning:info
* 2018**-**11**-**01 14:32:12.871 UTC [cli**.**logging] setLogSpec **->** INFO 001 Current logging spec set to: gossip**=**warning:msp**=**warning:info

Note: there is only one active logging spec. Any previous spec, including modules updated via ‘setlevel’, will no longer be applicable.

### Revert Levels Usage

Here is an example of the peer logging revertlevels command:

* To revert the logging spec to the start-up value:
* peer logging revertlevels
* 2018**-**11**-**01 14:37:12.402 UTC [cli**.**logging] revertLevels **->** INFO 001 Logging spec reverted to the peer's spec at startup.

[Creative Commons License](http://creativecommons.org/licenses/by/4.0/)  
This work is licensed under a [Creative Commons Attribution 4.0 International License](http://creativecommons.org/licenses/by/4.0/).

# peer node

The peer node command allows an administrator to start a peer node, check the status of a peer, reset all channels in a peer to the genesis block, or rollback a channel to a given block number.

## Syntax

The peer node command has the following subcommands:

* start
* status
* reset
* rollback

## peer node start

Starts a node that interacts **with** the network**.**

Usage:

peer node start [flags]

Flags:

**-**h, **--**help help **for** start

**--**peer**-**chaincodedev Whether peer **in** chaincode development mode

## peer node status

Returns the status of the running node**.**

Usage:

peer node status [flags]

Flags:

**-**h, **--**help help **for** status

## peer node reset

Resets all channels to the genesis block**.** When the command **is** executed, the peer must be offline**.** When the peer starts after the reset, it will receive blocks starting **with** block number one **from** an orderer **or** another peer to rebuild the block store **and** state database**.**

Usage:

peer node reset [flags]

Flags:

**-**h, **--**help help **for** reset

## peer node rollback

Rolls back a channel to a specified block number**.** When the command **is** executed, the peer must be offline**.** When the peer starts after the rollback, it will receive blocks, which got removed during the rollback, **from** an orderer **or** another peer to rebuild the block store **and** state database**.**

Usage:

peer node rollback [flags]

Flags:

**-**b, **--**blockNumber uint Block number to which the channel needs to be rolled back to**.**

**-**c, **--**channelID string Channel to rollback**.**

**-**h, **--**help help **for** rollback

## Example Usage

### peer node start example

The following command:

peer node start **--**peer**-**chaincodedev

starts a peer node in chaincode development mode. Normally chaincode containers are started and maintained by peer. However in chaincode development mode, chaincode is built and started by the user. This mode is useful during chaincode development phase for iterative development. See more information on development mode in the [chaincode tutorial](https://hyperledger-fabric.readthedocs.io/en/release-1.4/chaincode4ade.html).

### peer node reset example

peer node reset

resets all channels in the peer to the genesis block, i.e., the first block in the channel. The command also records the pre-reset height of each channel in the file system. Note that the peer process should be stopped while executing this command. If the peer process is running, this command detects that and returns an error instead of performing the reset. When the peer is started after performing the reset, the peer will fetch the blocks for each channel which were removed by the reset command (either from other peers or orderers) and commit the blocks up to the pre-reset height. Until all channels reach the pre-reset height, the peer will not endorse any transactions.

### peer node rollback example

The following command:

peer node rollback **-**c ch1 **-**b 150

rolls back the channel ch1 to block number 150. The command also records the pre-rolled back height of channel ch1 in the file system. Note that the peer should be stopped while executing this command. If the peer process is running, this command detects that and returns an error instead of performing the rollback. When the peer is started after performing the rollback, the peer will fetch the blocks for channel ch1 which were removed by the rollback command (either from other peers or orderers) and commit the blocks up to the pre-rolled back height. Until the channel ch1 reaches the pre-rolled back height, the peer will not endorse any transaction for any channel.

[Creative Commons License](http://creativecommons.org/licenses/by/4.0/)  
This work is licensed under a [Creative Commons Attribution 4.0 International License](http://creativecommons.org/licenses/by/4.0/).

# configtxgen

The configtxgen command allows users to create and inspect channel config related artifacts. The content of the generated artifacts is dictated by the contents of configtx.yaml.

## Syntax

The configtxgen tool has no sub-commands, but supports flags which can be set to accomplish a number of tasks.

## configtxgen

Usage of configtxgen:

**-**asOrg string

Performs the config generation **as** a particular organization (by name), only including values **in** the write set that org (likely) has privilege to set

**-**channelCreateTxBaseProfile string

Specifies a profile to consider **as** the orderer system channel current state to allow modification of non**-**[application](https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/configtxgen.html) parameters during channel create tx generation**.** Only valid **in** conjuction **with** 'outputCreateChannelTx'**.**

**-**channelID string

The channel ID to use **in** the configtx

**-**configPath string

The path containing the configuration to use (**if** set)

**-**inspectBlock string

Prints the configuration contained **in** the block at the specified path

**-**inspectChannelCreateTx string

Prints the configuration contained **in** the transaction at the specified path

**-**outputAnchorPeersUpdate string

Creates an config update to update an anchor peer (works only **with** the default channel creation, **and** only **for** the first update)

**-**outputBlock string

The path to write the genesis block to (**if** set)

**-**outputCreateChannelTx string

The path to write a channel creation configtx to (**if** set)

**-**printOrg string

Prints the definition of an organization **as** JSON**.** (useful **for** adding an org to a channel manually)

**-**profile string

The profile **from** configtx.yaml to use **for** generation**.** (default "SampleInsecureSolo")

**-**version

Show version information

## Usage

### Output a genesis block

Write a genesis block to genesis\_block.pb for channel orderer-system-channel for profile SampleSingleMSPSoloV1\_1.

configtxgen **-**outputBlock genesis\_block**.**pb **-**profile SampleSingleMSPSoloV1\_1 **-**channelID orderer**-**system**-**channel

### Output a channel creation tx

Write a channel creation transaction to create\_chan\_tx.pb for profile SampleSingleMSPChannelV1\_1.

configtxgen **-**outputCreateChannelTx create\_chan\_tx**.**pb **-**profile SampleSingleMSPChannelV1\_1 **-**channelID application**-**channel**-**1

### Inspect a genesis block

Print the contents of a genesis block named genesis\_block.pb to the screen as JSON.

configtxgen **-**inspectBlock genesis\_block**.**pb

### Inspect a channel creation tx

Print the contents of a channel creation tx named create\_chan\_tx.pb to the screen as JSON.

configtxgen **-**inspectChannelCreateTx create\_chan\_tx**.**pb

### Print an organization definition

Construct an organization definition based on the parameters such as MSPDir from configtx.yaml and print it as JSON to the screen. (This output is useful for channel reconfiguration workflows, such as adding a member).

configtxgen **-**printOrg Org1

### Output anchor peer tx

Output a configuration update transaction to anchor\_peer\_tx.pb which sets the anchor peers for organization Org1 as defined in profile SampleSingleMSPChannelV1\_1 based on configtx.yaml.

configtxgen **-**outputAnchorPeersUpdate anchor\_peer\_tx**.**pb **-**profile SampleSingleMSPChannelV1\_1 **-**asOrg Org1

## Configuration

The configtxgen tool’s output is largely controlled by the content of configtx.yaml. This file is searched for at FABRIC\_CFG\_PATH and must be present for configtxgen to operate.

This configuration file may be edited, or, individual properties may be overridden by setting environment variables, such as CONFIGTX\_ORDERER\_ORDERERTYPE=kafka.

For many configtxgen operations, a profile name must be supplied. Profiles are a way to express multiple similar configurations in a single file. For instance, one profile might define a channel with 3 orgs, and another might define one with 4 orgs. To accomplish this without the length of the file becoming burdensome, configtx.yaml depends on the standard YAML feature of anchors and references. Base parts of the configuration are tagged with an anchor like &OrdererDefaults and then merged into a profile with a reference like <<: \*OrdererDefaults. Note, when configtxgen is operating under a profile, environment variable overrides do not need to include the profile prefix and may be referenced relative to the root element of the profile. For instance, do not specify CONFIGTX\_PROFILE\_SAMPLEINSECURESOLO\_ORDERER\_ORDERERTYPE, instead simply omit the profile specifics and use the CONFIGTX prefix followed by the elements relative to the profile name such as CONFIGTX\_ORDERER\_ORDERERTYPE.

Refer to the sample configtx.yaml shipped with Fabric for all possible configuration options. You may find this file in the config directory of the release artifacts tar, or you may find it under the sampleconfig folder if you are building from source.

[Creative Commons License](http://creativecommons.org/licenses/by/4.0/)  
This work is licensed under a [Creative Commons Attribution 4.0 International License](http://creativecommons.org/licenses/by/4.0/).

# configtxlator

The configtxlator command allows users to translate between protobuf and JSON versions of fabric data structures and create config updates. The command may either start a REST server to expose its functions over HTTP or may be utilized directly as a command line tool.

## Syntax

The configtxlator tool has five sub-commands, as follows:

* start
* proto\_encode
* proto\_decode
* compute\_update
* version

## configtxlator start

usage: configtxlator start [**<**flags**>**]

Start the configtxlator REST server

Flags:

**--**help Show context**-**sensitive help (also **try** **--**help**-**long **and**

**--**help**-**man)**.**

**--**hostname**=**"0.0.0.0" The hostname **or** IP on which the REST server will listen

**--**port**=**7059 The port on which the REST server will listen

**--**CORS**=**CORS **...** Allowable CORS domains, e**.**g**.** '\*' **or** 'www.example.com'

(may be repeated)**.**

## configtxlator proto\_encode

usage: configtxlator proto\_encode **--**type**=**TYPE [**<**flags**>**]

Converts a JSON document to protobuf**.**

Flags:

**--**help Show context**-**sensitive help (also **try** **--**help**-**long **and**

**--**help**-**man)**.**

**--**type**=**TYPE The type of protobuf structure to encode to**.** For

example, 'common.Config'**.**

**--**input**=/**dev**/**stdin A file containing the JSON document**.**

**--**output**=/**dev**/**stdout A file to write the output to**.**

## configtxlator proto\_decode

usage: configtxlator proto\_decode **--**type**=**TYPE [**<**flags**>**]

Converts a proto message to JSON**.**

Flags:

**--**help Show context**-**sensitive help (also **try** **--**help**-**long **and**

**--**help**-**man)**.**

**--**type**=**TYPE The type of protobuf structure to decode from**.** For

example, 'common.Config'**.**

**--**input**=/**dev**/**stdin A file containing the proto message**.**

**--**output**=/**dev**/**stdout A file to write the JSON [document](https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/configtxlator.html) to**.**

## configtxlator compute\_update

usage: configtxlator compute\_update **--**channel\_id**=**CHANNEL\_ID [**<**flags**>**]

Takes two marshaled common**.**Config messages **and** computes the config update which

transitions between the two**.**

Flags:

**--**help Show context**-**sensitive help (also **try** **--**help**-**long **and**

**--**help**-**man)**.**

**--**original**=**ORIGINAL The original config message**.**

**--**updated**=**UPDATED The updated config message**.**

**--**channel\_id**=**CHANNEL\_ID The name of the channel **for** this update**.**

**--**output**=/**dev**/**stdout A file to write the JSON document to**.**

## configtxlator version

usage: configtxlator version

Show version information

Flags:

**--**help Show context**-**sensitive help (also **try** **--**help**-**long **and** **--**help**-**man)**.**

## Examples

### Decoding

Decode a block named fabric\_block.pb to JSON and print to stdout.

configtxlator proto\_decode **--**input fabric\_block**.**pb **--**type common**.**Block

Alternatively, after starting the REST server, the following curl command performs the same operation through the REST API.

curl **-**X POST **--**data**-**binary @fabric\_block**.**pb "${CONFIGTXLATOR\_URL}/protolator/decode/common.Block"

### Encoding

Convert a JSON document for a policy from stdin to a file named policy.pb.

configtxlator proto\_encode **--**type common**.**Policy **--**output policy**.**pb

Alternatively, after starting the REST server, the following curl command performs the same operation through the REST API.

curl **-**X POST **--**data**-**binary **/**dev**/**stdin "${CONFIGTXLATOR\_URL}/protolator/encode/common.Policy" **>** policy**.**pb

### Pipelines

Compute a config update from original\_config.pb and modified\_config.pb and decode it to JSON to stdout.

configtxlator compute\_update **--**channel\_id testchan **--**original original\_config**.**pb **--**updated modified\_config**.**pb **|** configtxlator proto\_decode **--**type common**.**ConfigUpdate

Alternatively, after starting the REST server, the following curl commands perform the same operations through the REST API.

curl **-**X POST **-**F channel**=**testchan **-**F "original=@original\_config.pb" **-**F "updated=@modified\_config.pb" "${CONFIGTXLATOR\_URL}/configtxlator/compute/update-from-configs" **|** curl **-**X POST **--**data**-**binary **/**dev**/**stdin "${CONFIGTXLATOR\_URL}/protolator/encode/common.ConfigUpdate"

## Additional Notes

The tool name is a portmanteau of configtx and translator and is intended to convey that the tool simply converts between different equivalent data representations. It does not generate configuration. It does not submit or retrieve configuration. It does not modify configuration itself, it simply provides some bijective operations between different views of the configtx format.

There is no configuration file configtxlator nor any authentication or authorization facilities included for the REST server. Because configtxlator does not have any access to data, key material, or other information which might be considered sensitive, there is no risk to the owner of the server in exposing it to other clients. However, because the data sent by a user to the REST server might be confidential, the user should either trust the administrator of the server, run a local instance, or operate via the CLI.

[Creative Commons License](http://creativecommons.org/licenses/by/4.0/)  
This work is licensed under a [Creative Commons Attribution 4.0 International License](http://creativecommons.org/licenses/by/4.0/).

# cryptogen

cryptogen is an utility for generating Hyperledger Fabric key material. It is provided as a means of preconfiguring a network for testing purposes. It would normally not be used in the operation of a production network.

## Syntax

The cryptogen command has five subcommands, as follows:

* help
* generate
* showtemplate
* extend
* version

## cryptogen help

usage: cryptogen [**<**flags**>**] **<**command**>** [**<**args**>** **...**]

Utility **for** generating Hyperledger Fabric key material

Flags:

**--**help Show context**-**sensitive help (also **try** **--**help**-**long **and** **--**help**-**man)**.**

Commands:

help [**<**command**>...**]

Show help**.**

generate [**<**flags**>**]

Generate key material

showtemplate

Show the default configuration template

version

Show version information

extend [**<**flags**>**]

Extend existing network

## cryptogen generate

usage: cryptogen generate [**<**flags**>**]

Generate key material

Flags:

**--**help Show context**-**sensitive help (also **try** **--**help**-**long

**and** **--**help**-**man)**.**

**--**output**=**"crypto-config" The output directory **in** which to place artifacts

**--**config**=**CONFIG The configuration template to use

## cryptogen showtemplate

usage: cryptogen showtemplate

Show the default configuration template

Flags:

**--**help Show [context](https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/cryptogen.html)**-**sensitive help (also **try** **--**help**-**long **and** **--**help**-**man)**.**

## cryptogen extend

usage: cryptogen extend [**<**flags**>**]

Extend existing network

Flags:

**--**help Show context**-**sensitive help (also **try** **--**help**-**long **and**

**--**help**-**man)**.**

**--**input**=**"crypto-config" The input directory **in** which existing network place

**--**config**=**CONFIG The configuration template to use

## cryptogen version

usage: cryptogen version

Show version information

Flags:

**--**help Show context**-**sensitive help (also **try** **--**help**-**long **and** **--**help**-**man)**.**

## Usage

Here’s an example using the different available flags on the cryptogen extend command.

cryptogen extend **--**input**=**"crypto-config" **--**config**=**config**.**yaml

org3**.**example**.**com

Where config.yaml adds a new peer organization called org3.example.com

[Creative Commons License](http://creativecommons.org/licenses/by/4.0/)  
This work is licensed under a [Creative Commons Attribution 4.0 International License](http://creativecommons.org/licenses/by/4.0/).

# Service Discovery CLI

The discovery service has its own Command Line Interface (CLI) which uses a YAML configuration file to persist properties such as certificate and private key paths, as well as MSP ID.

The discover command has the following subcommands:

* saveConfig
* peers
* config
* endorsers

And the usage of the command is shown below:

usage: discover [<flags>] <command> [<args> ...]

Command line client for fabric discovery service

Flags:

--help Show context-sensitive help (also try --help-long and --help-man).

--configFile=CONFIGFILE Specifies the config file to load the configuration from

--peerTLSCA=PEERTLSCA Sets the TLS CA certificate file path that verifies the TLS peer's certificate

--tlsCert=TLSCERT (Optional) Sets the client TLS certificate file path that is used when the peer enforces client authentication

--tlsKey=TLSKEY (Optional) Sets the client TLS key file path that is used when the peer enforces client authentication

--userKey=USERKEY Sets the user's key file path that is used to sign messages sent to the peer

--userCert=USERCERT Sets the user's certificate file path that is used to authenticate the messages sent to the peer

--MSP=MSP Sets the MSP ID of the user, which represents the CA(s) that issued its user certificate

Commands:

help [<command>...]

Show help.

peers [<flags>]

Discover peers

config [<flags>]

Discover channel config

endorsers [<flags>]

Discover chaincode endorsers

saveConfig

Save the config passed by flags into the file specified by --configFile

## Configuring external endpoints

Currently, to see peers in service discovery they need to have EXTERNAL\_ENDPOINT to be configured for them. Otherwise, Fabric assumes the peer should not be disclosed.

To define these endpoints, you need to specify them in the core.yaml of the peer, replacing the sample endpoint below with the ones of your peer.

CORE\_PEER\_GOSSIP\_EXTERNALENDPOINT**=**peer1**.**org1**.**example**.**com:8051

## Persisting configuration

To persist the configuration, a config file name should be supplied via the flag --configFile, along with the command saveConfig:

discover --configFile conf.yaml --peerTLSCA tls/ca.crt --userKey msp/keystore/ea4f6a38ac7057b6fa9502c2f5f39f182e320f71f667749100fe7dd94c23ce43\_sk --userCert msp/signcerts/User1\@org1.example.com-cert.pem --MSP Org1MSP saveConfig

By executing the above command, configuration file would be created:

$ cat conf.yaml

version: 0

tlsconfig:

certpath: ""

keypath: ""

peercacertpath: /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/User1@org1.example.com/tls/ca.crt

timeout: 0s

signerconfig:

mspid: Org1MSP

identitypath: /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp/signcerts/User1@org1.example.com-cert.pem

keypath: /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp/keystore/ea4f6a38ac7057b6fa9502c2f5f39f182e320f71f667749100fe7dd94c23ce43\_sk

When the peer runs with TLS enabled, the discovery service on the peer requires the client to connect to it with mutual TLS, which means it needs to supply a TLS certificate. The peer is configured by default to request (but not to verify) client TLS certificates, so supplying a TLS certificate isn’t needed (unless the peer’s tls.clientAuthRequired is set to true).

When the discovery CLI’s config file has a certificate path for peercacertpath, but the certpath and keypath aren’t configured as in the above - the discovery CLI generates a self-signed TLS certificate and uses this to connect to the peer.

When the peercacertpath isn’t configured, the discovery CLI connects without TLS , and this is highly not recommended, as the information is sent over plaintext, un-encrypted.

## Querying the discovery service

The discoveryCLI acts as a discovery client, and it needs to be executed against a peer. This is done via specifying the --server flag. In addition, the queries are channel-scoped, so the --channel flag must be used.

The only query that doesn’t require a channel is the local membership peer query, which by default can only be used by administrators of the peer being queried.

The discover CLI supports all server-side queries:

* Peer membership query
* Configuration query
* Endorsers query

Let’s go over them and see how they should be invoked and parsed:

## Peer membership query:

$ discover --configFile conf.yaml peers --channel mychannel --server peer0.org1.example.com:7051

[

{

"MSPID": "Org2MSP",

"LedgerHeight": 5,

"Endpoint": "peer0.org2.example.com:9051",

"Identity": "-----BEGIN CERTIFICATE-----\nMIICKTCCAc+gAwIBAgIRANK4WBck5gKuzTxVQIwhYMUwCgYIKoZIzj0EAwIwczEL\nMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcTDVNhbiBG\ncmFuY2lzY28xGTAXBgNVBAoTEG9yZzIuZXhhbXBsZS5jb20xHDAaBgNVBAMTE2Nh\nLm9yZzIuZXhhbXBsZS5jb20wHhcNMTgwNjE3MTM0NTIxWhcNMjgwNjE0MTM0NTIx\nWjBqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMN\nU2FuIEZyYW5jaXNjbzENMAsGA1UECxMEcGVlcjEfMB0GA1UEAxMWcGVlcjAub3Jn\nMi5leGFtcGxlLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABJa0gkMRqJCi\nzmx+L9xy/ecJNvdAV2zmSx5Sf2qospVAH1MYCHyudDEvkiRuBPgmCdOdwJsE0g+h\nz0nZdKq6/X+jTTBLMA4GA1UdDwEB/wQEAwIHgDAMBgNVHRMBAf8EAjAAMCsGA1Ud\nIwQkMCKAIFZMuZfUtY6n2iyxaVr3rl+x5lU0CdG9x7KAeYydQGTMMAoGCCqGSM49\nBAMCA0gAMEUCIQC0M9/LJ7j3I9NEPQ/B1BpnJP+UNPnGO2peVrM/mJ1nVgIgS1ZA\nA1tsxuDyllaQuHx2P+P9NDFdjXx5T08lZhxuWYM=\n-----END CERTIFICATE-----\n",

"Chaincodes": [

"mycc"

]

},

{

"MSPID": "Org2MSP",

"LedgerHeight": 5,

"Endpoint": "peer1.org2.example.com:10051",

"Identity": "-----BEGIN CERTIFICATE-----\nMIICKDCCAc+gAwIBAgIRALnNJzplCrYy4Y8CjZtqL7AwCgYIKoZIzj0EAwIwczEL\nMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcTDVNhbiBG\ncmFuY2lzY28xGTAXBgNVBAoTEG9yZzIuZXhhbXBsZS5jb20xHDAaBgNVBAMTE2Nh\nLm9yZzIuZXhhbXBsZS5jb20wHhcNMTgwNjE3MTM0NTIxWhcNMjgwNjE0MTM0NTIx\nWjBqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMN\nU2FuIEZyYW5jaXNjbzENMAsGA1UECxMEcGVlcjEfMB0GA1UEAxMWcGVlcjEub3Jn\nMi5leGFtcGxlLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABNDopAkHlDdu\nq10HEkdxvdpkbs7EJyqv1clvCt/YMn1hS6sM+bFDgkJKalG7s9Hg3URF0aGpy51R\nU+4F9Muo+XajTTBLMA4GA1UdDwEB/wQEAwIHgDAMBgNVHRMBAf8EAjAAMCsGA1Ud\nIwQkMCKAIFZMuZfUtY6n2iyxaVr3rl+x5lU0CdG9x7KAeYydQGTMMAoGCCqGSM49\nBAMCA0cAMEQCIAR4fBmIBKW2jp0HbbabVepNtl1c7+6++riIrEBnoyIVAiBBvWmI\nyG02c5hu4wPAuVQMB7AU6tGSeYaWSAAo/ExunQ==\n-----END CERTIFICATE-----\n",

"Chaincodes": [

"mycc"

]

},

{

"MSPID": "Org1MSP",

"LedgerHeight": 5,

"Endpoint": "peer0.org1.example.com:7051",

"Identity": "-----BEGIN CERTIFICATE-----\nMIICKDCCAc6gAwIBAgIQP18LeXtEXGoN8pTqzXTHZTAKBggqhkjOPQQDAjBzMQsw\nCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMNU2FuIEZy\nYW5jaXNjbzEZMBcGA1UEChMQb3JnMS5leGFtcGxlLmNvbTEcMBoGA1UEAxMTY2Eu\nb3JnMS5leGFtcGxlLmNvbTAeFw0xODA2MTcxMzQ1MjFaFw0yODA2MTQxMzQ1MjFa\nMGoxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQHEw1T\nYW4gRnJhbmNpc2NvMQ0wCwYDVQQLEwRwZWVyMR8wHQYDVQQDExZwZWVyMC5vcmcx\nLmV4YW1wbGUuY29tMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEKeC/1Rg/ynSk\nNNItaMlaCDZOaQvxJEl6o3fqx1PVFlfXE4NarY3OO1N3YZI41hWWoXksSwJu/35S\nM7wMEzw+3KNNMEswDgYDVR0PAQH/BAQDAgeAMAwGA1UdEwEB/wQCMAAwKwYDVR0j\nBCQwIoAgcecTOxTes6rfgyxHH6KIW7hsRAw2bhP9ikCHkvtv/RcwCgYIKoZIzj0E\nAwIDSAAwRQIhAKiJEv79XBmr8gGY6kHrGL0L3sq95E7IsCYzYdAQHj+DAiBPcBTg\nRuA0//Kq+3aHJ2T0KpKHqD3FfhZZolKDkcrkwQ==\n-----END CERTIFICATE-----\n",

"Chaincodes": [

"mycc"

]

},

{

"MSPID": "Org1MSP",

"LedgerHeight": 5,

"Endpoint": "peer1.org1.example.com:8051",

"Identity": "-----BEGIN CERTIFICATE-----\nMIICJzCCAc6gAwIBAgIQO7zMEHlMfRhnP6Xt65jwtDAKBggqhkjOPQQDAjBzMQsw\nCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMNU2FuIEZy\nYW5jaXNjbzEZMBcGA1UEChMQb3JnMS5leGFtcGxlLmNvbTEcMBoGA1UEAxMTY2Eu\nb3JnMS5leGFtcGxlLmNvbTAeFw0xODA2MTcxMzQ1MjFaFw0yODA2MTQxMzQ1MjFa\nMGoxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQHEw1T\nYW4gRnJhbmNpc2NvMQ0wCwYDVQQLEwRwZWVyMR8wHQYDVQQDExZwZWVyMS5vcmcx\nLmV4YW1wbGUuY29tMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEoII9k8db/Q2g\nRHw5rk3SYw+OMFw9jNbsJJyC5ttJRvc12Dn7lQ8ZR9hW1vLQ3NtqO/couccDJcHg\nt47iHBNadaNNMEswDgYDVR0PAQH/BAQDAgeAMAwGA1UdEwEB/wQCMAAwKwYDVR0j\nBCQwIoAgcecTOxTes6rfgyxHH6KIW7hsRAw2bhP9ikCHkvtv/RcwCgYIKoZIzj0E\nAwIDRwAwRAIgGHGtRVxcFVeMQr9yRlebs23OXEECNo6hNqd/4ChLwwoCIBFKFd6t\nlL5BVzVMGQyXWcZGrjFgl4+fDrwjmMe+jAfa\n-----END CERTIFICATE-----\n",

"Chaincodes": null

}

]

As seen, this command outputs a JSON containing membership information about all the peers in the channel that the peer queried possesses.

The Identity that is returned is the enrollment certificate of the peer, and it can be parsed with a combination of jq and openssl:

$ discover --configFile conf.yaml peers --channel mychannel --server peer0.org1.example.com:7051 | jq .[0].Identity | sed "s/\\\n/\n/g" | sed "s/\"//g" | openssl x509 -text -noout

Certificate:

Data:

Version: 3 (0x2)

Serial Number:

55:e9:3f:97:94:d5:74:db:e2:d6:99:3c:01:24:be:bf

Signature Algorithm: ecdsa-with-SHA256

Issuer: C=US, ST=California, L=San Francisco, O=org2.example.com, CN=ca.org2.example.com

Validity

Not Before: Jun 9 11:58:28 2018 GMT

Not After : Jun 6 11:58:28 2028 GMT

Subject: C=US, ST=California, L=San Francisco, OU=peer, CN=peer0.org2.example.com

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

pub:

04:f5:69:7a:11:65:d9:85:96:65:b7:b7:1b:08:77:

43:de:cb:ad:3a:79:ec:cc:2a:bc:d7:93:68:ae:92:

1c:4b:d8:32:47:d6:3d:72:32:f1:f1:fb:26:e4:69:

c2:eb:c9:45:69:99:78:d7:68:a9:77:09:88:c6:53:

01:2a:c1:f8:c0

ASN1 OID: prime256v1

NIST CURVE: P-256

X509v3 extensions:

X509v3 Key Usage: critical

Digital Signature

X509v3 Basic Constraints: critical

CA:FALSE

X509v3 Authority Key Identifier:

keyid:8E:58:82:C9:0A:11:10:A9:0B:93:03:EE:A0:54:42:F4:A3:EF:11:4C:82:B6:F9:CE:10:A2:1E:24:AB:13:82:A0

Signature Algorithm: ecdsa-with-SHA256

30:44:02:20:29:3f:55:2b:9f:7b:99:b2:cb:06:ca:15:3f:93:

a1:3d:65:5c:7b:79:a1:7a:d1:94:50:f0:cd:db:ea:61:81:7a:

02:20:3b:40:5b:60:51:3c:f8:0f:9b:fc:ae:fc:21:fd:c8:36:

a3:18:39:58:20:72:3d:1a:43:74:30:f3:56:01:aa:26

## Configuration query:

The configuration query returns a mapping from MSP IDs to orderer endpoints, as well as the FabricMSPConfig which can be used to verify all peer and orderer nodes by the SDK:

$ discover --configFile conf.yaml config --channel mychannel --server peer0.org1.example.com:7051

{

"msps": {

"OrdererOrg": {

"name": "OrdererMSP",

"root\_certs": [

""

],

"admins": [

""

],

"crypto\_config": {

"signature\_hash\_family": "SHA2",

"identity\_identifier\_hash\_function": "SHA256"

},

"tls\_root\_certs": [

""

]

},

"Org1MSP": {

"name": "Org1MSP",

"root\_certs": [

""

],

"admins": [

""

],

"crypto\_config": {

"signature\_hash\_family": "SHA2",

"identity\_identifier\_hash\_function": "SHA256"

},

"tls\_root\_certs": [

""

],

"fabric\_node\_ous": {

"enable": true,

"client\_ou\_identifier": {

"certificate": "",

"organizational\_unit\_identifier": "client"

},

"peer\_ou\_identifier": {

"certificate": "",

"organizational\_unit\_identifier": "peer"

}

}

},

"Org2MSP": {

"name": "Org2MSP",

"root\_certs": [

""

],

"admins": [

""

],

"crypto\_config": {

"signature\_hash\_family": "SHA2",

"identity\_identifier\_hash\_function": "SHA256"

},

"tls\_root\_certs": [

""

],

"fabric\_node\_ous": {

"enable": true,

"client\_ou\_identifier": {

"certificate": "",

"organizational\_unit\_identifier": "client"

},

"peer\_ou\_identifier": {

"certificate": "",

"organizational\_unit\_identifier": "peer"

}

}

},

"Org3MSP": {

"name": "Org3MSP",

"root\_certs": [

""

],

"intermediate\_certs": [

""

],

"admins": [

"LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUhZd0VBWUhLb1pJemowQ0FRWUZLNEVFQUNJRFlnQUVUYk13SEZteEpEMWR3SjE2K0hnVnRDZkpVRzdKK2FTYgorbkVvVmVkREVHYmtTc1owa1lraEpyYkx5SHlYZm15ZWV0ejFIUk1rWjRvMjdxRlMzTlVFb1J2QlM3RHJPWDJjCnZLaDRnbWhHTmlPbzRiWjFOVG9ZL2o3QnpqMFlMSXNlCi0tLS0tRU5EIFBVQkxJQyBLRVktLS0tLQo="

]

}

},

"orderers": {

"OrdererOrg": {

"endpoint": [

{

"host": "orderer.example.com",

"port": 7050

}

]

}

}

}

It’s important to note that the certificates here are base64 encoded, and thus should decoded in a manner similar to the following:

$ discover --configFile conf.yaml config --channel mychannel --server peer0.org1.example.com:7051 | jq .msps.OrdererOrg.root\_certs[0] | sed "s/\"//g" | base64 --decode | openssl x509 -text -noout

Certificate:

Data:

Version: 3 (0x2)

Serial Number:

c8:99:2d:3a:2d:7f:4b:73:53:8b:39:18:7b:c3:e1:1e

Signature Algorithm: ecdsa-with-SHA256

Issuer: C=US, ST=California, L=San Francisco, O=example.com, CN=ca.example.com

Validity

Not Before: Jun 9 11:58:28 2018 GMT

Not After : Jun 6 11:58:28 2028 GMT

Subject: C=US, ST=California, L=San Francisco, O=example.com, CN=ca.example.com

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

pub:

04:28:ac:9e:51:8d:a4:80:15:0a:ff:ae:c9:61:d6:

08:67:b0:15:c3:c7:99:46:61:63:0a:10:a6:42:6a:

b0:af:14:0c:c0:e2:5b:b4:a1:c3:f0:07:7e:5b:7c:

c4:b2:95:13:95:81:4b:6a:b9:e3:87:a4:f3:2c:7c:

ae:00:91:9e:32

ASN1 OID: prime256v1

NIST CURVE: P-256

X509v3 extensions:

X509v3 Key Usage: critical

Digital Signature, Key Encipherment, Certificate Sign, CRL Sign

X509v3 Extended Key Usage:

Any Extended Key Usage

X509v3 Basic Constraints: critical

CA:TRUE

X509v3 Subject Key Identifier:

60:9D:F2:30:26:CE:8F:65:81:41:AD:96:15:0E:24:8D:A0:9D:C5:79:C1:17:BF:FE:E5:1B:FB:75:50:10:A6:4C

Signature Algorithm: ecdsa-with-SHA256

30:44:02:20:3d:e1:a7:6c:99:3f:87:2a:36:44:51:98:37:11:

d8:a0:47:7a:33:ff:30:c1:09:a6:05:ec:b0:53:53:39:c1:0e:

02:20:6b:f4:1d:48:e0:72:e4:c2:ef:b0:84:79:d4:2e:c2:c5:

1b:6f:e4:2f:56:35:51:18:7d:93:51:86:05:84:ce:1f

## Endorsers query:

To query for the endorsers of a chaincode call, additional flags need to be supplied:

* The --chaincode flag is mandatory and it provides the chaincode name(s). To query for a chaincode-to-chaincode invocation, one needs to repeat the --chaincode flag with all the chaincodes.
* The --collection is used to specify private data collections that are expected to used by the chaincode(s). To map from thechaincodes passed via --chaincode to the collections, the following syntax should be used: collection=CC:Collection1,Collection2,....

For example, to query for a chaincode invocation that results in both cc1 and cc2 to be invoked, as well as writes to private data collection col1 by cc2, one needs to specify: --chaincode=cc1 --chaincode=cc2 --collection=cc2:col1

Below is the output of an endorsers query for chaincode **mycc** when the endorsement policy is AND('Org1.peer', 'Org2.peer'):

$ discover --configFile conf.yaml endorsers --channel mychannel --server peer0.org1.example.com:7051 --chaincode mycc

[

{

"Chaincode": "mycc",

"EndorsersByGroups": {

"G0": [

{

"MSPID": "Org1MSP",

"LedgerHeight": 5,

"Endpoint": "peer0.org1.example.com:7051",

"Identity": "-----BEGIN CERTIFICATE-----\nMIICKDCCAc+gAwIBAgIRANTiKfUVHVGnrYVzEy1ZSKIwCgYIKoZIzj0EAwIwczEL\nMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcTDVNhbiBG\ncmFuY2lzY28xGTAXBgNVBAoTEG9yZzEuZXhhbXBsZS5jb20xHDAaBgNVBAMTE2Nh\nLm9yZzEuZXhhbXBsZS5jb20wHhcNMTgwNjA5MTE1ODI4WhcNMjgwNjA2MTE1ODI4\nWjBqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMN\nU2FuIEZyYW5jaXNjbzENMAsGA1UECxMEcGVlcjEfMB0GA1UEAxMWcGVlcjAub3Jn\nMS5leGFtcGxlLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABD8jGz1l5Rrw\n5UWqAYnc4JrR46mCYwHhHFgwydccuytb00ouD4rECiBsCaeZFr5tODAK70jFOP/k\n/CtORCDPQ02jTTBLMA4GA1UdDwEB/wQEAwIHgDAMBgNVHRMBAf8EAjAAMCsGA1Ud\nIwQkMCKAIOBdQLF+cMWa6e1p2CpOEx7SHUinzVvd55hLm7w6v72oMAoGCCqGSM49\nBAMCA0cAMEQCIC3bacbDYphXfHrNULxpV/zwD08t7hJxNe8MwgP8/48fAiBiC0cr\nu99oLsRNCFB7R3egyKg1YYao0KWTrr1T+rK9Bg==\n-----END CERTIFICATE-----\n"

}

],

"G1": [

{

"MSPID": "Org2MSP",

"LedgerHeight": 5,

"Endpoint": "peer1.org2.example.com:10051",

"Identity": "-----BEGIN CERTIFICATE-----\nMIICKDCCAc+gAwIBAgIRAIs6fFxk4Y5cJxSwTjyJ9A8wCgYIKoZIzj0EAwIwczEL\nMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcTDVNhbiBG\ncmFuY2lzY28xGTAXBgNVBAoTEG9yZzIuZXhhbXBsZS5jb20xHDAaBgNVBAMTE2Nh\nLm9yZzIuZXhhbXBsZS5jb20wHhcNMTgwNjA5MTE1ODI4WhcNMjgwNjA2MTE1ODI4\nWjBqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMN\nU2FuIEZyYW5jaXNjbzENMAsGA1UECxMEcGVlcjEfMB0GA1UEAxMWcGVlcjEub3Jn\nMi5leGFtcGxlLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABOVFyWVmKZ25\nxDYV3xZBDX4gKQ7rAZfYgOu1djD9EHccZhJVPsdwSjbRsvrfs9Z8mMuwEeSWq/cq\n0cGrMKR93vKjTTBLMA4GA1UdDwEB/wQEAwIHgDAMBgNVHRMBAf8EAjAAMCsGA1Ud\nIwQkMCKAII5YgskKERCpC5MD7qBUQvSj7xFMgrb5zhCiHiSrE4KgMAoGCCqGSM49\nBAMCA0cAMEQCIDJmxseFul1GZ26djKa6jZ6zYYf6hchNF5xxMRWXpCnuAiBMf6JZ\njZjVM9F/OidQ2SBR7OZyMAzgXc5nAabWZpdkuQ==\n-----END CERTIFICATE-----\n"

},

{

"MSPID": "Org2MSP",

"LedgerHeight": 5,

"Endpoint": "peer0.org2.example.com:9051",

"Identity": "-----BEGIN CERTIFICATE-----\nMIICJzCCAc6gAwIBAgIQVek/l5TVdNvi1pk8ASS+vzAKBggqhkjOPQQDAjBzMQsw\nCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMNU2FuIEZy\nYW5jaXNjbzEZMBcGA1UEChMQb3JnMi5leGFtcGxlLmNvbTEcMBoGA1UEAxMTY2Eu\nb3JnMi5leGFtcGxlLmNvbTAeFw0xODA2MDkxMTU4MjhaFw0yODA2MDYxMTU4Mjha\nMGoxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQHEw1T\nYW4gRnJhbmNpc2NvMQ0wCwYDVQQLEwRwZWVyMR8wHQYDVQQDExZwZWVyMC5vcmcy\nLmV4YW1wbGUuY29tMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE9Wl6EWXZhZZl\nt7cbCHdD3sutOnnszCq815NorpIcS9gyR9Y9cjLx8fsm5GnC68lFaZl412ipdwmI\nxlMBKsH4wKNNMEswDgYDVR0PAQH/BAQDAgeAMAwGA1UdEwEB/wQCMAAwKwYDVR0j\nBCQwIoAgjliCyQoREKkLkwPuoFRC9KPvEUyCtvnOEKIeJKsTgqAwCgYIKoZIzj0E\nAwIDRwAwRAIgKT9VK597mbLLBsoVP5OhPWVce3mhetGUUPDN2+phgXoCIDtAW2BR\nPPgPm/yu/CH9yDajGDlYIHI9GkN0MPNWAaom\n-----END CERTIFICATE-----\n"

}

]

},

"Layouts": [

{

"quantities\_by\_group": {

"G0": 1,

"G1": 1

}

}

]

}

]

## Not using a configuration file

It is possible to execute the discovery CLI without having a configuration file, and just passing all needed configuration as commandline flags. The following is an example of a local peer membership query which loads administrator credentials:

$ discover --peerTLSCA tls/ca.crt --userKey msp/keystore/cf31339d09e8311ac9ca5ed4e27a104a7f82f1e5904b3296a170ba4725ffde0d\_sk --userCert msp/signcerts/Admin\@org1.example.com-cert.pem --MSP Org1MSP --tlsCert tls/client.crt --tlsKey tls/client.key peers --server peer0.org1.example.com:7051

[

{

"MSPID": "Org1MSP",

"Endpoint": "peer1.org1.example.com:8051",

"Identity": "-----BEGIN CERTIFICATE-----\nMIICJzCCAc6gAwIBAgIQO7zMEHlMfRhnP6Xt65jwtDAKBggqhkjOPQQDAjBzMQsw\nCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMNU2FuIEZy\nYW5jaXNjbzEZMBcGA1UEChMQb3JnMS5leGFtcGxlLmNvbTEcMBoGA1UEAxMTY2Eu\nb3JnMS5leGFtcGxlLmNvbTAeFw0xODA2MTcxMzQ1MjFaFw0yODA2MTQxMzQ1MjFa\nMGoxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQHEw1T\nYW4gRnJhbmNpc2NvMQ0wCwYDVQQLEwRwZWVyMR8wHQYDVQQDExZwZWVyMS5vcmcx\nLmV4YW1wbGUuY29tMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEoII9k8db/Q2g\nRHw5rk3SYw+OMFw9jNbsJJyC5ttJRvc12Dn7lQ8ZR9hW1vLQ3NtqO/couccDJcHg\nt47iHBNadaNNMEswDgYDVR0PAQH/BAQDAgeAMAwGA1UdEwEB/wQCMAAwKwYDVR0j\nBCQwIoAgcecTOxTes6rfgyxHH6KIW7hsRAw2bhP9ikCHkvtv/RcwCgYIKoZIzj0E\nAwIDRwAwRAIgGHGtRVxcFVeMQr9yRlebs23OXEECNo6hNqd/4ChLwwoCIBFKFd6t\nlL5BVzVMGQyXWcZGrjFgl4+fDrwjmMe+jAfa\n-----END CERTIFICATE-----\n",

},

{

"MSPID": "Org1MSP",

"Endpoint": "peer0.org1.example.com:7051",

"Identity": "-----BEGIN CERTIFICATE-----\nMIICKDCCAc6gAwIBAgIQP18LeXtEXGoN8pTqzXTHZTAKBggqhkjOPQQDAjBzMQsw\nCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMNU2FuIEZy\nYW5jaXNjbzEZMBcGA1UEChMQb3JnMS5leGFtcGxlLmNvbTEcMBoGA1UEAxMTY2Eu\nb3JnMS5leGFtcGxlLmNvbTAeFw0xODA2MTcxMzQ1MjFaFw0yODA2MTQxMzQ1MjFa\nMGoxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQHEw1T\nYW4gRnJhbmNpc2NvMQ0wCwYDVQQLEwRwZWVyMR8wHQYDVQQDExZwZWVyMC5vcmcx\nLmV4YW1wbGUuY29tMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEKeC/1Rg/ynSk\nNNItaMlaCDZOaQvxJEl6o3fqx1PVFlfXE4NarY3OO1N3YZI41hWWoXksSwJu/35S\nM7wMEzw+3KNNMEswDgYDVR0PAQH/BAQDAgeAMAwGA1UdEwEB/wQCMAAwKwYDVR0j\nBCQwIoAgcecTOxTes6rfgyxHH6KIW7hsRAw2bhP9ikCHkvtv/RcwCgYIKoZIzj0E\nAwIDSAAwRQIhAKiJEv79XBmr8gGY6kHrGL0L3sq95E7IsCYzYdAQHj+DAiBPcBTg\nRuA0//Kq+3aHJ2T0KpKHqD3FfhZZolKDkcrkwQ==\n-----END CERTIFICATE-----\n",

},

{

"MSPID": "Org2MSP",

"Endpoint": "peer0.org2.example.com:9051",

"Identity": "-----BEGIN CERTIFICATE-----\nMIICKTCCAc+gAwIBAgIRANK4WBck5gKuzTxVQIwhYMUwCgYIKoZIzj0EAwIwczEL\nMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcTDVNhbiBG\ncmFuY2lzY28xGTAXBgNVBAoTEG9yZzIuZXhhbXBsZS5jb20xHDAaBgNVBAMTE2Nh\nLm9yZzIuZXhhbXBsZS5jb20wHhcNMTgwNjE3MTM0NTIxWhcNMjgwNjE0MTM0NTIx\nWjBqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMN\nU2FuIEZyYW5jaXNjbzENMAsGA1UECxMEcGVlcjEfMB0GA1UEAxMWcGVlcjAub3Jn\nMi5leGFtcGxlLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABJa0gkMRqJCi\nzmx+L9xy/ecJNvdAV2zmSx5Sf2qospVAH1MYCHyudDEvkiRuBPgmCdOdwJsE0g+h\nz0nZdKq6/X+jTTBLMA4GA1UdDwEB/wQEAwIHgDAMBgNVHRMBAf8EAjAAMCsGA1Ud\nIwQkMCKAIFZMuZfUtY6n2iyxaVr3rl+x5lU0CdG9x7KAeYydQGTMMAoGCCqGSM49\nBAMCA0gAMEUCIQC0M9/LJ7j3I9NEPQ/B1BpnJP+UNPnGO2peVrM/mJ1nVgIgS1ZA\nA1tsxuDyllaQuHx2P+P9NDFdjXx5T08lZhxuWYM=\n-----END CERTIFICATE-----\n",

},

{

"MSPID": "Org2MSP",

"Endpoint": "peer1.org2.example.com:10051",

"Identity": "-----BEGIN CERTIFICATE-----\nMIICKDCCAc+gAwIBAgIRALnNJzplCrYy4Y8CjZtqL7AwCgYIKoZIzj0EAwIwczEL\nMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcTDVNhbiBG\ncmFuY2lzY28xGTAXBgNVBAoTEG9yZzIuZXhhbXBsZS5jb20xHDAaBgNVBAMTE2Nh\nLm9yZzIuZXhhbXBsZS5jb20wHhcNMTgwNjE3MTM0NTIxWhcNMjgwNjE0MTM0NTIx\nWjBqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEWMBQGA1UEBxMN\nU2FuIEZyYW5jaXNjbzENMAsGA1UECxMEcGVlcjEfMB0GA1UEAxMWcGVlcjEub3Jn\nMi5leGFtcGxlLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABNDopAkHlDdu\nq10HEkdxvdpkbs7EJyqv1clvCt/YMn1hS6sM+bFDgkJKalG7s9Hg3URF0aGpy51R\nU+4F9Muo+XajTTBLMA4GA1UdDwEB/wQEAwIHgDAMBgNVHRMBAf8EAjAAMCsGA1Ud\nIwQkMCKAIFZMuZfUtY6n2iyxaVr3rl+x5lU0CdG9x7KAeYydQGTMMAoGCCqGSM49\nBAMCA0cAMEQCIAR4fBmIBKW2jp0HbbabVepNtl1c7+6++riIrEBnoyIVAiBBvWmI\nyG02c5hu4wPAuVQMB7AU6tGSeYaWSAAo/ExunQ==\n-----END CERTIFICATE-----\n",

}

]

# Fabric-CA Commands

The Hyperledger Fabric CA is a Certificate Authority (CA) for Hyperledger Fabric. The commands available for the fabric-ca client and fabric-ca server are described in the links below.

## Fabric-CA Client

The fabric-ca-client command allows you to manage identities (including attribute management) and certificates (including renewal and revocation).

More information on fabric-ca-client commands can be found [here](https://hyperledger-fabric-ca.readthedocs.io/en/release-1.1/clientcli.html#fabric-ca-client-s-cli).

## Fabric-CA Server

The fabric-ca-server command allows you to initialize and start a server process which may host one or more certificate authorities.

More information on fabric-ca-server commands can be found [here](https://hyperledger-fabric-ca.readthedocs.io/en/release-1.1/servercli.html#fabric-ca-server-s-cli).

# Fabric-CA Server’s CLI

Hyperledger Fabric Certificate Authority Server

Usage:

fabric**-**ca**-**server [command]

Available Commands:

init Initialize the fabric**-**ca server

[start](https://hyperledger-fabric-ca.readthedocs.io/en/release-1.1/servercli.html) Start the fabric**-**ca server

version Prints Fabric CA Server version

Flags:

**--**address string Listening address of fabric**-**ca**-**server (default "0.0.0.0")

**-**b, **--**boot string The [user](https://hyperledger-fabric-ca.readthedocs.io/en/release-1.1/servercli.html):**pass** **for** bootstrap admin which **is** [required](https://hyperledger-fabric-ca.readthedocs.io/en/release-1.1/servercli.html) to build default config file

**--**ca**.**certfile string PEM**-**encoded CA certificate file (default "ca-cert.pem")

**--**ca**.**chainfile string PEM**-**encoded CA chain file (default "ca-chain.pem")

**--**ca**.**keyfile string PEM**-**encoded CA key file

**-**n, **--**ca**.**name string Certificate Authority name

**--**cacount int Number of non**-**default CA instances

**--**cafiles stringSlice A list of comma**-**separated CA configuration files

**--**cfg**.**affiliations**.**allowremove Enables removal of affiliations dynamically

**--**cfg**.**identities**.**allowremove Enables removal of identities dynamically

**--**crl**.**expiry duration Expiration **for** the CRL generated by the gencrl request (default 24h0m0s)

**--**crlsizelimit int Size limit of an acceptable CRL **in** bytes (default 512000)

**--**csr**.**cn string The common name field of the certificate signing request to a parent fabric**-**ca**-**server

**--**csr**.**hosts stringSlice A list of space**-**separated host names **in** a certificate signing request to a parent fabric**-**ca**-**server

**--**csr**.**serialnumber string The serial number **in** a certificate signing request to a parent fabric**-**ca**-**server

**--**db**.**datasource string Data source which **is** database specific (default "fabric-ca-server.db")

**--**db**.**tls**.**certfiles stringSlice A list of comma**-**separated PEM**-**encoded trusted certificate files (e**.**g**.** root1**.**pem,root2**.**pem)

**--**db**.**tls**.**client**.**certfile string PEM**-**encoded certificate file when mutual authenticate **is** enabled

**--**db**.**tls**.**client**.**keyfile string PEM**-**encoded key file when mutual authentication **is** enabled

**--**db**.**type string Type of database; one of: sqlite3, postgres, mysql (default "sqlite3")

**-**d, **--**debug Enable debug level logging

**-**H, **--**home string Server's home directory (default "/etc/hyperledger/fabric-ca")

**--**intermediate**.**enrollment**.**label string Label to use **in** HSM operations

**--**intermediate**.**enrollment**.**profile string Name of the signing profile to use **in** issuing the certificate

**--**intermediate**.**parentserver**.**caname string Name of the CA to connect to on fabric**-**ca**-**server

**-**u, **--**intermediate**.**parentserver**.**url string URL of the parent fabric**-**ca**-**server (e**.**g**.** http:**//<**username**>**:**<**password**>@<**address**>**:**<**port)

**--**intermediate**.**tls**.**certfiles stringSlice A list of comma**-**separated PEM**-**encoded trusted certificate files (e**.**g**.** root1**.**pem,root2**.**pem)

**--**intermediate**.**tls**.**client**.**certfile string PEM**-**encoded certificate file when mutual authenticate **is** enabled

**--**intermediate**.**tls**.**client**.**keyfile string PEM**-**encoded key file when mutual authentication **is** enabled

**--**ldap**.**attribute**.**names stringSlice The names of LDAP attributes to request on an LDAP search

**--**ldap**.**enabled Enable the LDAP client **for** authentication **and** attributes

**--**ldap**.**groupfilter string The LDAP group filter **for** a single affiliation group (default "(memberUid=%s)")

**--**ldap**.**tls**.**certfiles stringSlice A list of comma**-**separated PEM**-**encoded trusted certificate files (e**.**g**.** root1**.**pem,root2**.**pem)

**--**ldap**.**tls**.**client**.**certfile string PEM**-**encoded certificate file when mutual authenticate **is** enabled

**--**ldap**.**tls**.**client**.**keyfile string PEM**-**encoded key file when mutual authentication **is** enabled

**--**ldap**.**url string LDAP client URL of form ldap:**//**adminDN:adminPassword@host[:port]**/**base

**--**ldap**.**userfilter string The LDAP user filter to use when searching **for** users (default "(uid=%s)")

**-**p, **--**port int Listening port of fabric**-**ca**-**server (default 7054)

**--**registry**.**maxenrollments int Maximum number of enrollments; valid **if** LDAP **not** enabled (default **-**1)

**--**tls**.**certfile string PEM**-**encoded TLS certificate file **for** server's listening port (default "tls-cert.pem")

**--**tls**.**clientauth**.**certfiles stringSlice A list of comma**-**separated PEM**-**encoded trusted certificate files (e**.**g**.** root1**.**pem,root2**.**pem)

**--**tls**.**clientauth**.**type string Policy the server will follow **for** TLS Client Authentication**.** (default "noclientcert")

**--**tls**.**enabled Enable TLS on the listening port

**--**tls**.**keyfile string PEM**-**encoded TLS key **for** server's listening port

Use "fabric-ca-server [command] --help" **for** more information about a command**.**

© Copyright 2017, hyperledger. Revision e6568899.

Built with [Sphinx](http://sphinx-doc.org/) using a [theme](https://github.com/snide/sphinx_rtd_theme) provided by [Read the Docs](https://readthedocs.org/).