**Chaincode for Operators**

**What is Chaincode?**

Chaincode is a program, written in [Go](https://golang.org/), [node.js](https://nodejs.org/), or [Java](https://java.com/en/) that implements a prescribed interface. Chaincode runs in a secured Docker container isolated from the endorsing peer process. Chaincode initializes and manages ledger state through transactions submitted by applications.

A chaincode typically handles business logic agreed to by members of the network, so it may be considered as a “smart contract”. State created by a chaincode is scoped exclusively to that chaincode and can’t be accessed directly by another chaincode. However, within the same network, given the appropriate permission a chaincode may invoke another chaincode to access its state.

In the following sections, we will explore chaincode through the eyes of a blockchain network operator, Noah. For Noah’s interests, we will focus on chaincode lifecycle operations; the process of packaging, installing, instantiating and upgrading the chaincode as a function of the chaincode’s operational lifecycle within a blockchain network.

**Chaincode lifecycle**

The Hyperledger Fabric API enables interaction with the various nodes in a blockchain network - the peers, orderers and MSPs - and it also allows one to package, install, instantiate and upgrade chaincode on the endorsing peer nodes. The Hyperledger Fabric language-specific SDKs abstract the specifics of the Hyperledger Fabric API to facilitate application development, though it can be used to manage a chaincode’s lifecycle. Additionally, the Hyperledger Fabric API can be accessed directly via the CLI, which we will use in this document.

We provide four commands to manage a chaincode’s lifecycle: package, install, instantiate, and upgrade. In a future release, we are considering adding stop and start transactions to disable and re-enable a chaincode without having to actually uninstall it. After a chaincode has been successfully installed and instantiated, the chaincode is active (running) and can process transactions via the invoke transaction. A chaincode may be upgraded any time after it has been installed.

**Packaging**

The chaincode package consists of 3 parts:

* the chaincode, as defined by ChaincodeDeploymentSpec or CDS. The CDS defines the chaincode package in terms of the code and other properties such as name and version,
* an optional instantiation policy which can be syntactically described by the same policy used for endorsement and described in [Endorsement policies](https://hyperledger-fabric.readthedocs.io/en/release-1.4/endorsement-policies.html), and
* a set of signatures by the entities that “own” the chaincode.

The signatures serve the following purposes:

* to establish an ownership of the chaincode,
* to allow verification of the contents of the package, and
* to allow detection of package tampering.

The creator of the instantiation transaction of the chaincode on a channel is validated against the instantiation policy of the chaincode.

**Creating the package**

There are two approaches to packaging chaincode. One for when you want to have multiple owners of a chaincode, and hence need to have the chaincode package signed by multiple identities. This workflow requires that we initially create a signed chaincode package (a SignedCDS) which is subsequently passed serially to each of the other owners for signing.

The simpler workflow is for when you are deploying a SignedCDS that has only the signature of the identity of the node that is issuing the install transaction.

We will address the more complex case first. However, you may skip ahead to the [Installing chaincode](https://hyperledger-fabric.readthedocs.io/en/release-1.4/chaincode4noah.html#install) section below if you do not need to worry about multiple owners just yet.

To create a signed chaincode package, use the following command:

peer chaincode package **-**n mycc **-**p github**.**com**/**hyperledger**/**fabric**/**examples**/**chaincode**/**go**/**example02**/**cmd **-**v 0 **-**s **-**S **-**i "AND('OrgA.admin')" ccpack**.**out

The -s option creates a package that can be signed by multiple owners as opposed to simply creating a raw CDS. When -s is specified, the -S option must also be specified if other owners are going to need to sign. Otherwise, the process will create a SignedCDS that includes only the instantiation policy in addition to the CDS.

The -S option directs the process to sign the package using the MSP identified by the value of the localMspid property in core.yaml.

The -S option is optional. However if a package is created without a signature, it cannot be signed by any other owner using the signpackage command.

The optional -i option allows one to specify an instantiation policy for the chaincode. The instantiation policy has the same format as an endorsement policy and specifies which identities can instantiate the chaincode. In the example above, only the admin of OrgA is allowed to instantiate the chaincode. If no policy is provided, the default policy is used, which only allows the admin identity of the peer’s MSP to instantiate chaincode.

**Package signing**

A chaincode package that was signed at creation can be handed over to other owners for inspection and signing. The workflow supports out-of-band signing of chaincode package.

The [ChaincodeDeploymentSpec](https://github.com/hyperledger/fabric/blob/master/protos/peer/chaincode.proto" \l "L78) may be optionally be signed by the collective owners to create a [SignedChaincodeDeploymentSpec](https://github.com/hyperledger/fabric/blob/master/protos/peer/signed_cc_dep_spec.proto" \l "L26) (or SignedCDS). The SignedCDS contains 3 elements:

1. The CDS contains the source code, the name, and version of the chaincode.
2. An instantiation policy of the chaincode, expressed as endorsement policies.
3. The list of chaincode owners, defined by means of [Endorsement](https://github.com/hyperledger/fabric/blob/master/protos/peer/proposal_response.proto#L111).

**Note**

Note that this endorsement policy is determined out-of-band to provide proper MSP principals when the chaincode is instantiated on some channels. If the instantiation policy is not specified, the default policy is any MSP administrator of the channel.

Each owner endorses the ChaincodeDeploymentSpec by combining it with that owner’s identity (e.g. certificate) and signing the combined result.

A chaincode owner can sign a previously created signed package using the following command:

peer chaincode signpackage ccpack**.**out signedccpack**.**out

Where ccpack.out and signedccpack.out are the input and output packages, respectively. signedccpack.out contains an additional signature over the package signed using the Local MSP.

**Installing chaincode**

The install transaction packages a chaincode’s source code into a prescribed format called a ChaincodeDeploymentSpec (or CDS) and installs it on a peer node that will run that chaincode.

**Note**

You must install the chaincode on **each** endorsing peer node of a channel that will run your chaincode.

When the install API is given simply a ChaincodeDeploymentSpec, it will default the instantiation policy and include an empty owner list.

**Note**

Chaincode should only be installed on endorsing peer nodes of the owning members of the chaincode to protect the confidentiality of the chaincode logic from other members on the network. Those members without the chaincode, can’t be the endorsers of the chaincode’s transactions; that is, they can’t execute the chaincode. However, they can still validate and commit the transactions to the ledger.

To install a chaincode, send a [SignedProposal](https://github.com/hyperledger/fabric/blob/master/protos/peer/proposal.proto" \l "L104) to the lifecycle system chaincode (LSCC) described in the [System Chaincode](https://hyperledger-fabric.readthedocs.io/en/release-1.4/chaincode4noah.html#system-chaincode) section. For example, to install the **sacc** sample chaincode described in section [Simple Asset Chaincode](https://hyperledger-fabric.readthedocs.io/en/release-1.4/chaincode4ade.html#simple-asset-chaincode) using the CLI, the command would look like the following:

peer chaincode install **-**n asset\_mgmt **-**v 1.0 **-**p sacc

The CLI internally creates the SignedChaincodeDeploymentSpec for **sacc** and sends it to the local peer, which calls the Install method on the LSCC. The argument to the -p option specifies the path to the chaincode, which must be located within the source tree of the user’s GOPATH, e.g. $GOPATH/src/sacc. Note if using -l node or -l java for node chaincode or java chaincode, use -p with the absolute path of the chaincode location. See the [Commands Reference](https://hyperledger-fabric.readthedocs.io/en/release-1.4/command_ref.html) for a complete description of the command options.

Note that in order to install on a peer, the signature of the SignedProposal must be from 1 of the peer’s local MSP administrators.

**Instantiate**

The instantiate transaction invokes the lifecycle System Chaincode (LSCC) to create and initialize a chaincode on a channel. This is a chaincode-channel binding process: a chaincode may be bound to any number of channels and operate on each channel individually and independently. In other words, regardless of how many other channels on which a chaincode might be installed and instantiated, state is kept isolated to the channel to which a transaction is submitted.

The creator of an instantiate transaction must satisfy the instantiation policy of the chaincode included in SignedCDS and must also be a writer on the channel, which is configured as part of the channel creation. This is important for the security of the channel to prevent rogue entities from deploying chaincodes or tricking members to execute chaincodes on an unbound channel.

For example, recall that the default instantiation policy is any channel MSP administrator, so the creator of a chaincode instantiate transaction must be a member of the channel administrators. When the transaction proposal arrives at the endorser, it verifies the creator’s signature against the instantiation policy. This is done again during the transaction validation before committing it to the ledger.

The instantiate transaction also sets up the endorsement policy for that chaincode on the channel. The endorsement policy describes the attestation requirements for the transaction result to be accepted by members of the channel.

For example, using the CLI to instantiate the **sacc** chaincode and initialize the state with john and 0, the command would look like the following:

peer chaincode instantiate **-**n sacc **-**v 1.0 **-**c '{"Args":["john","0"]}' **-**C mychannel **-**P "AND ('Org1.member','Org2.member')"

**Note**

Note the endorsement policy (CLI uses polish notation), which requires an endorsement from both a member of Org1 and Org2 for all transactions to **sacc**. That is, both Org1 and Org2 must sign the result of executing the *Invoke* on **sacc** for the transactions to be valid.

After being successfully instantiated, the chaincode enters the active state on the channel and is ready to process any transaction proposals of type [ENDORSER\_TRANSACTION](https://github.com/hyperledger/fabric/blob/master/protos/common/common.proto#L42). The transactions are processed concurrently as they arrive at the endorsing peer.

**Upgrade**

A chaincode may be upgraded any time by changing its version, which is part of the SignedCDS. Other parts, such as owners and instantiation policy are optional. However, the chaincode name must be the same; otherwise it would be considered as a totally different chaincode.

Prior to upgrade, the new version of the chaincode must be installed on the required endorsers. Upgrade is a transaction similar to the instantiate transaction, which binds the new version of the chaincode to the channel. Other channels bound to the old version of the chaincode still run with the old version. In other words, the upgrade transaction only affects one channel at a time, the channel to which the transaction is submitted.

**Note**

Note that since multiple versions of a chaincode may be active simultaneously, the upgrade process doesn’t automatically remove the old versions, so user must manage this for the time being.

There’s one subtle difference with the instantiate transaction: the upgrade transaction is checked against the current chaincode instantiation policy, not the new policy (if specified). This is to ensure that only existing members specified in the current instantiation policy may upgrade the chaincode.

**Note**

Note that during upgrade, the chaincode Init function is called to perform any data related updates or re-initialize it, so care must be taken to avoid resetting states when upgrading chaincode.

**Stop and Start**

Note that stop and start lifecycle transactions have not yet been implemented. However, you may stop a chaincode manually by removing the chaincode container and the SignedCDS package from each of the endorsers. This is done by deleting the chaincode’s container on each of the hosts or virtual machines on which the endorsing peer nodes are running, and then deleting the SignedCDS from each of the endorsing peer nodes:

**Note**

TODO - in order to delete the CDS from the peer node, you would need to enter the peer node’s container, first. We really need to provide a utility script that can do this.

docker rm **-**f **<**container id**>**

rm **/**var**/**hyperledger**/**production**/**chaincodes**/<**ccname**>**:**<**ccversion**>**

Stop would be useful in the workflow for doing upgrade in controlled manner, where a chaincode can be stopped on a channel on all peers before issuing an upgrade.

**System chaincode**

System chaincode has the same programming model except that it runs within the peer process rather than in an isolated container like normal chaincode. Therefore, system chaincode is built into the peer executable and doesn’t follow the same lifecycle described above. In particular, **install**, **instantiate** and **upgrade** do not apply to system chaincodes.

The purpose of system chaincode is to shortcut gRPC communication cost between peer and chaincode, and tradeoff the flexibility in management. For example, a system chaincode can only be upgraded with the peer binary. It must also register with a [fixed set of parameters](https://github.com/hyperledger/fabric/blob/master/core/scc/importsysccs.go) compiled in and doesn’t have endorsement policies or endorsement policy functionality.

System chaincode is used in Hyperledger Fabric to implement a number of system behaviors so that they can be replaced or modified as appropriate by a system integrator.

The current list of system chaincodes:

1. [LSCC](https://github.com/hyperledger/fabric/tree/master/core/scc/lscc) Lifecycle system chaincode handles lifecycle requests described above.
2. [CSCC](https://github.com/hyperledger/fabric/tree/master/core/scc/cscc) Configuration system chaincode handles channel configuration on the peer side.
3. [QSCC](https://github.com/hyperledger/fabric/tree/master/core/scc/qscc) Query system chaincode provides ledger query APIs such as getting blocks and transactions.

The former system chaincodes for endorsement and validation have been replaced by the pluggable endorsement and validation function as described by the [Pluggable transaction endorsement and validation](https://hyperledger-fabric.readthedocs.io/en/release-1.4/pluggable_endorsement_and_validation.html) documentation.

Extreme care must be taken when modifying or replacing these system chaincodes, especially LSCC.