Upgrading your components

Audience: network administrators, node administrators

For information about special considerations for the latest release of Fabric, check out Upgrading to the latest release of Fabric.

This topic will only cover the process for upgrading components. For information about how to edit a channel to change the capability level of your channels, check out Updating a channel capability.

Note: when we use the term "upgrade" in Hyperledger Fabric, we're referring to changing the version of a component (for example, going from one version of a binary to the next version). The term "update," on the other hand, refers not to versions but to configuration changes, such as updating a channel configuration or a deployment script. As there is no data migration, technically speaking, in Fabric, we will not use the term "migration" or "migrate" here.

Overview

At a high level, upgrading the binary level of your nodes is a two step process:

1. Backup the ledger and MSPs.
2. Upgrade binaries to the latest version.

If you own both ordering nodes and peers, it is a best practice to upgrade the ordering nodes first. If a peer falls behind or is temporarily unable to process certain transactions, it can always catch up. If enough ordering nodes go down, by comparison, a network can effectively cease to function.

This topic presumes that these steps will be performed using Docker CLI commands. If you are utilizing a different deployment method (Rancher, Kubernetes, OpenShift, etc) consult their documentation on how to use their CLI.

For native deployments, note that you will also need to update the YAML configuration file for the nodes (for example, the `orderer.yaml` file) with the one from the release artifacts.

To do this, backup the `orderer.yaml` or `core.yaml` file (for the peer) and replace it with the `orderer.yaml` or `core.yaml` file from the release artifacts. Then port any modified variables from the backed up `orderer.yaml` or `core.yaml` to the new one. Using a utility like `diff` may be helpful. Note that updating the YAML file from the release rather than updating your old YAML file **is the recommended way to update your node YAML files**, as it reduces the likelihood of making errors.

This tutorial assumes a Docker deployment where the YAML files will be baked into the images and environment variables will be used to overwrite the defaults in the configuration files.

Environment variables for the binaries

When you deploy a peer or an ordering node, you had to set a number of environment variables relevant to its configuration. A best practice is to create a file for these environment variables, give it a name relevant to the node being deployed, and save it somewhere on your local file system. That way you can be sure that when upgrading the peer or ordering node you are using the same variables you set when creating it.

Here's a list of some of the **peer** environment variables (with sample values — as you can see from the addresses, these environment variables are for a network deployed locally) that can be set that be listed in the file. Note that you may or may not need to set all of these environment variables:

```
CORE_PEER_TLS_ENABLED=true
CORE_PEER_GOSSIP_USELEADERELECTION=true
CORE_PEER_GOSSIP_ORGLEADER=false
CORE_PEER_PROFILE_ENABLED=true
CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
CORE_PEER_ID=peer0.org1.example.com
CORE_PEER_ADDRESS=peer0.org1.example.com:7051
CORE_PEER_LISTENADDRESS=0.0.0.0:7051
CORE_PEER_CHAINCODEADDRESS=peer0.org1.example.com:7052
CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:7052
CORE_PEER_GOSSIP_BOOTSTRAP=peer0.org1.example.com:7051
CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org1.example.com:7051
CORE_PEER_LOCALMSPID=Org1MSP
```

Here are some **ordering node** variables (again, these are sample values) that might be listed in the environment variable file for a node. Again, you may or may not need to set all of these environment variables:

```
ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
ORDERER_GENERAL_GENESISMETHOD=file
ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block
ORDERER_GENERAL_LOCALMSPID=OrdererMSP
ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
ORDERER_GENERAL_TLS_ENABLED=true
ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
ORDERER_GENERAL_CLUSTER_CLIENTCERTIFICATE=/var/hyperledger/orderer/tls/server.crt
ORDERER_GENERAL_CLUSTER_CLIENTPRIVATEKEY=/var/hyperledger/orderer/tls/server.key
ORDERER_GENERAL_CLUSTER_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
```

However you choose to set your environment variables, note that they will have to be set for each node you want to upgrade.

Ledger backup and restore

While we will demonstrate the process for backing up ledger data in this tutorial, it is not strictly required to backup the ledger data of a peer or an ordering node (assuming the node is part of a larger group of nodes in an ordering service). This is because, even in the worst case of catastrophic failure of a peer (such as a disk failure), the peer can be brought up with no ledger at all. You can then have the peer re-join the desired channels and as a result, the peer will automatically create a ledger for each of the channels and will start receiving the blocks via regular block transfer mechanism from

either the ordering service or the other peers in the channel. As the peer processes blocks, it will also build up its state database.

However, backing up ledger data enables the restoration of a peer without the time and computational costs associated with bootstrapping from the genesis block and reprocessing all transactions, a process that can take hours (depending on the size of the ledger). In addition, ledger data backups may help to expedite the addition of a new peer, which can be achieved by backing up the ledger data from one peer and starting the new peer with the backed up ledger data.

This tutorial presumes that the file path to the ledger data has not been changed from the default value of `/var/hyperledger/production/` (for peers) or `/var/hyperledger/production/orderer` (for ordering nodes). If this location has been changed for your nodes, enter the path to the data on your ledgers in the commands below.

Note that there will be data for both the ledger and chaincodes at this file location. While it is a best practice to backup both, it is possible to skip the `stateLeveldb`, `historyLeveldb`, `chains/index` folders at `/var/hyperledger/production/ledgersData`. While skipping these folders reduces the storage needed for the backup, the peer recovery from the backed up data may take more time as these ledger artifacts will be re-constructed when the peer starts.

If using CouchDB as state database, there will be no `stateLeveldb` directory, as the state database data would be stored within CouchDB instead. But similarly, if peer starts up and finds CouchDB databases are missing or at lower block height (based on using an older CouchDB backup), the state database will be automatically re-constructed to catch up to current block height. Therefore, if you backup peer ledger data and CouchDB data separately, ensure that the CouchDB backup is always older than the peer backup.

Upgrade ordering nodes

Orderer containers should be upgraded in a rolling fashion (one at a time). At a high level, the ordering node upgrade process goes as follows:

1. Stop the ordering node.
2. Back up the ordering node's ledger and MSP.
3. Remove the ordering node container.
4. Launch a new ordering node container using the relevant image tag.

Repeat this process for each node in your ordering service until the entire ordering service has been upgraded.

Set command environment variables

Export the following environment variables before attempting to upgrade your ordering nodes.

- `ORDERER_CONTAINER`: the name of your ordering node container. Note that you will need to export this variable for each node when upgrading it.

- `LEDGERS_BACKUP`: the place in your local filesystem where you want to store the ledger being backed up. As you will see below, each node being backed up will have its own subfolder containing its ledger. You will need to create this folder.
- `IMAGE_TAG`: the Fabric version you are upgrading to. For example, `2.0`.

Note that you will have to set an **image tag** to ensure that the node you are starting using the correct images. The process you use to set the tag will depend on your deployment method.

Upgrade containers

Let's begin the upgrade process by **bringing down the orderer**:

```
docker stop $ORDERER_CONTAINER
```

Once the orderer is down, you'll want to **backup its ledger and MSP**:

```
docker cp $ORDERER_CONTAINER:/var/hyperledger/production/orderer/ ./$LEDGERS_BACKUP/$ORDERER_CONT
```

Then remove the ordering node container itself (since we will be giving our new container the same name as our old one):

```
docker rm -f $ORDERER_CONTAINER
```

Then you can launch the new ordering node container by issuing:

```
docker run -d -v /opt/backup/$ORDERER_CONTAINER/:/var/hyperledger/production/orderer/ \
           -v /opt/msp/:/etc/hyperledger/fabric/msp/ \
           --env-file ./env<name of node>.list \
           --name $ORDERER_CONTAINER \
           hyperledger/fabric-orderer:$IMAGE_TAG orderer
```

Once all of the ordering nodes have come up, you can move on to upgrading your peers.

Upgrade the peers

Peers should, like the ordering nodes, be upgraded in a rolling fashion (one at a time). As mentioned during the ordering node upgrade, ordering nodes and peers may be upgraded in parallel, but for the purposes of this tutorial we've separated the processes out. At a high level, we will perform the following steps:

1. Stop the peer.
2. Back up the peer's ledger and MSP.
3. Remove chaincode containers and images.
4. Remove the peer container.
5. Launch a new peer container using the relevant image tag.

Set command environment variables

Export the following environment variables before attempting to upgrade your peers.

- `PEER_CONTAINER`: the name of your peer container. Note that you will need to set this variable for each node.
- `LEDGERS_BACKUP`: the place in your local filesystem where you want to store the ledger being backed up. As you will see below, each node being backed up will have its own subfolder containing its ledger. You will need to create this folder.
- `IMAGE_TAG`: the Fabric version you are upgrading to. For example, `2.0`.

Note that you will have to set an **image tag** to ensure that the node you are starting is using the correct images. The process you use to set the tag will depend on your deployment method.

Repeat this process for each of your peers until every node has been upgraded.

Upgrade containers

Let's **bring down the first peer** with the following command:

```
docker stop $PEER_CONTAINER
```

We can then **backup the peer's ledger and MSP**:

```
docker cp $PEER_CONTAINER:/var/hyperledger/production ./$LEDGERS_BACKUP/$PEER_CONTAINER
```

With the peer stopped and the ledger backed up, **remove the peer chaincode containers**:

```
CC_CONTAINERS=$(docker ps | grep dev-$PEER_CONTAINER | awk '{print $1}')
if [ -n "$CC_CONTAINERS" ] ; then docker rm -f $CC_CONTAINERS ; fi
```

And the peer chaincode images:

```
CC_IMAGES=$(docker images | grep dev-$PEER | awk '{print $1}')
if [ -n "$CC_IMAGES" ] ; then docker rmi -f $CC_IMAGES ; fi
```

Then remove the peer container itself (since we will be giving our new container the same name as our old one):

```
docker rm -f $PEER_CONTAINER
```

Then you can launch the new peer container by issuing:

```
docker run -d -v /opt/backup/$PEER_CONTAINER/:/var/hyperledger/production/ \
            -v /opt/msp/:/etc/hyperledger/fabric/msp/ \
            --env-file ./env<name of node>.list \
            --name $PEER_CONTAINER \
            hyperledger/fabric-peer:$IMAGE_TAG peer node start
```

You do not need to relaunch the chaincode container. When the peer gets a request for a chaincode, (invoke or query), it first checks if it has a copy of that chaincode running. If so, it uses it. Otherwise, as in this case, the peer launches the chaincode (rebuilding the image if required).

Verify peer upgrade completion

It's a best practice to ensure the upgrade has been completed properly with a chaincode invoke. Note that it should be possible to verify that a single peer has been successfully updated by querying one of the ledgers hosted on the peer. If you want to verify that multiple peers have been upgraded, and are updating your chaincode as part of the upgrade process, you should wait until peers from enough organizations to satisfy the endorsement policy have been upgraded.

Before you attempt this, you may want to upgrade peers from enough organizations to satisfy your endorsement policy. However, this is only mandatory if you are updating your chaincode as part of the upgrade process. If you are not updating your chaincode as part of the upgrade process, it is possible to get endorsements from peers running at different Fabric versions.

Upgrade your CAs

To learn how to upgrade your Fabric CA server, click over to the CA documentation.

Upgrade Node SDK clients

Upgrade Fabric and Fabric CA before upgrading Node SDK clients. Fabric and Fabric CA are tested for backwards compatibility with older SDK clients. While newer SDK clients often work with older Fabric and Fabric CA releases, they may expose features that are not yet available in the older Fabric and Fabric CA releases, and are not tested for full compatibility.

Use NPM to upgrade any `Node.js` client by executing these commands in the root directory of your application:

```
npm install fabric-client@latest

npm install fabric-ca-client@latest
```

These commands install the new version of both the Fabric client and Fabric-CA client and write the new versions to `package.json`.

Upgrading CouchDB

If you are using CouchDB as state database, you should upgrade the peer's CouchDB at the same time the peer is being upgraded.

To upgrade CouchDB:

1. Stop CouchDB.
2. Backup CouchDB data directory.
3. Install the latest CouchDB binaries or update deployment scripts to use a new Docker image.
4. Restart CouchDB.

Upgrade Node chaincode shim

To move to the new version of the Node chaincode shim a developer would need to:

1. Change the level of `fabric-shim` in their chaincode `package.json` from their old level to the new one.
2. Repackage this new chaincode package and install it on all the endorsing peers in the channel.
3. Perform an upgrade to this new chaincode. To see how to do this, check out Peer chaincode commands.

Upgrade Chaincodes with vendored shim

For information about upgrading the Go chaincode shim specific to the v2.0 release, check out Chaincode shim changes.

A number of third party tools exist that will allow you to vendor a chaincode shim. If you used one of these tools, use the same one to update your vendored chaincode shim and re-package your chaincode.

If your chaincode vendors the shim, after updating the shim version, you must install it to all peers which already have the chaincode. Install it with the same name, but a newer version. Then you should execute a chaincode upgrade on each channel where this chaincode has been deployed to move to the new version.