Updating the capability level of a channel

Audience: network administrators, node administrators

If you're not familiar with capabilities, check out Capabilities before proceeding, paying particular attention to the fact that **peers and orderers that belong to the channel must be upgraded before enabling capabilities**.

For information about any new capability levels in the latest release of Fabric, check out Upgrading your components.

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.

Prerequisites and considerations

If you haven't already done so, ensure you have all of the dependencies on your machine as described in Prerequisites. This will ensure that you have the latest versions of the tools required to make a channel configuration update.

Although Fabric binaries can and should be upgraded in a rolling fashion, it is important to **finish upgrading binaries before enabling capabilities**. Any binaries which are not upgraded to at least the level of the relevant capabilities will crash to indicate a misconfiguration which could otherwise result in a ledger fork.

Once a capability has been enabled, it becomes part of the permanent record for that channel. This means that even after disabling the capability, old binaries will not be able to participate in the channel because they cannot process beyond the block which enabled the capability to get to the block which disables it. As a result, once a capability has been enabled, disabling it is neither recommended nor supported.

For this reason, think of enabling channel capabilities as a point of no return. Please experiment with the new capabilities in a test setting and be confident before proceeding to enable them in production.

Overview

In this tutorial, we will show the process for updating capabilities in all of the parts of the configuration of both the ordering system channel and any application channels.

Whether you will need to update every part of the configuration for all of your channels will depend on the contents of the latest release as well as your own use case. For more information, check out Upgrading to the latest version of Fabric. Note that it may be necessary to update to the newest capability levels before using the features in the latest release, and it is considered a best practice to always be at the latest binary versions and capability levels.

Because updating the capability level of a channel involves the configuration update transaction process, we will be relying on our Updating a channel configuration topic for many of the commands.

As with any channel configuration update, updating capabilities is, at a high level, a three step process (for each channel):

1. Get the latest channel config
2. Create a modified channel config
3. Create a config update transaction

We will enable these capabilities in the following order:

1. Orderer system channel

- Orderer group
- Channel group

1. Application channels

- Orderer group
- Channel group
- Application group

While it is possible to edit multiple parts of the configuration of a channel at the same time, in this tutorial we will show how this process is done incrementally. In other words, we will not bundle a change to the `Orderer` group and the `Channel` group of the system channel into one configuration change. This is because not every release will have both a new `Orderer` group capability and a `Channel` group capability.

Note that in production networks, it will not be possible or desirable for one user to be able to update all of these channels (and parts of configurations) unilaterally. The orderer system channel, for example, is administered exclusively by ordering organization admins (though it is possible to add peer organizations as ordering service organizations). Similarly, updating either the `Orderer` or `Channel` groups of a channel configuration requires the signature of an ordering service organization in addition to peer organizations. Distributed systems require collaborative management.

Create a capabilities config file

Note that this tutorial presumes that a file called `capabilities.json` has been created and includes the capability updates you want to make to the various sections of the config. It also uses `jq` to

apply the edits to the modified config file.

Note that you are not obligated to create a file like `capabilities.json` or to use a tool like `jq`. The modified config can also be edited manually (after it has been pulled, translated, and scoped). Check out this sample channel configuration for reference.

However, the process described here (using a JSON file and a tool like `jq`) does have the advantage of being scriptable, making it suitable for proposing configuration updates to a large number of channels. This is why it is **the recommended way to update channels**.

In this example, the `capabilities.json` file looks like this (note: if you are updating your channel as part of Upgrading to the latest version of Fabric you will need to set the capabilities to the levels appropriate to that release):

```
{
  "channel": {
      "mod_policy": "Admins",
          "value": {
              "capabilities": {
                  "V2_0": {}
              }
          },
      "version": "0"
  },
  "orderer": {
      "mod_policy": "Admins",
          "value": {
              "capabilities": {
                  "V2_0": {}
              }
          },
      "version": "0"
  },
  "application": {
      "mod_policy": "Admins",
          "value": {
              "capabilities": {
                  "V2_0": {}
              }
          },
      "version": "0"
  }
}
```

Note that by default peer organizations are not admins of the orderer system channel and will therefore be unable to propose configuration updates to it. An orderer organization admin would have to create a file like this (without the `application` group capability, which does not exist in the system channel) to propose updating the system channel configuration. Note that because application channel copy the system channel configuration by default, unless a different channel profile is created which specifies capability levels, the `Channel` and `Orderer` group capabilities for the application channel will be the same as those in the network's system channel.

Orderer system channel capabilities

Because application channels copy the configuration of the orderer system channel by default, it is considered a best practice to update the capabilities of the system channel before any application

channels. This mirrors the process of updating ordering nodes to the newest version before peers, as described in Upgrading your components.

Note that the orderer system channel is administered by ordering service organizations. By default this will be a single organization (the organization that created the initial nodes in the ordering service), but more organizations can be added here (for example, if multiple organizations have contributed nodes to the ordering service).

Make sure all of the ordering nodes in your ordering service have been upgraded to the required binary level before updating the `Orderer` and `Channel` capability. If an ordering node is not at the required level, it will be unable to process the config block with the capability and will crash. Similarly, note that if a new channel is created on this ordering service, all of the peers that will be joined to it must be at least to the node level corresponding to the `Channel` and `Application` capabilities, otherwise they will also crash when attempting to process the config block. For more information, check out Capabilities.

Set environment variables

You will need to export the following variables:

- `CH_NAME`: the name of the system channel being updated.
- `CORE_PEER_LOCALMSPID`: the MSP ID of the organization proposing the channel update. This will be the MSP of one of the orderer organizations.
- `TLS_ROOT_CA`: the absolute path to the TLS cert of your ordering node(s).
- `CORE_PEER_MSPCONFIGPATH`: the absolute path to the MSP representing your organization.
- `ORDERER_CONTAINER`: the name of an ordering node container. When targeting the ordering service, you can target any particular node in the ordering service. Your requests will be forwarded to the leader automatically.

`Orderer` group

For the commands on how to pull, translate, and scope the channel config, navigate to Step 1: Pull and translate the config. Once you have a `modified_config.json`, add the capabilities to the `Orderer` group of the config (as listed in `capabilities.json`) using this command:

```
jq -s '.[0] * {"channel_group":{"groups":{"Orderer": {"values": {"Capabilities": .[1].orderer}}}}}
```

Then, follow the steps at Step 3: Re-encode and submit the config.

Note that because you are updating the system channel, the `mod_policy` for the system channel will only require the signature of ordering service organization admins.

`Channel` group

Once again, navigate to Step 1: Pull and translate the config. Once you have a `modified_config.json`, add the capabilities to the `Channel` group of the config (as listed in `capabilities.json`) using this command:

```
jq -s '.[0] * {"channel_group":{"values": {"Capabilities": .[1].channel}}}' config.json ./capabil
```

Then, follow the steps at Step 3: Re-encode and submit the config.

Note that because you are updating the system channel, the `mod_policy` for the system channel will only require the signature of ordering service organization admins. In an application channel, as you'll see, you would normally need to satisfy both the `MAJORITY` `Admins` policy of both the `Application` group (consisting of the MSPs of peer organizations) and the `Orderer` group (consisting of ordering service organizations), assuming you have not changed the default values.

Enable capabilities on existing channels

Now that we have updating the capabilities on the orderer system channel, we need to updating the configuration of any existing application channels you want to update.

As you will see, the configuration of application channels is very similar to that of the system channel. This is what allows us to re-use `capabilities.json` and the same commands we used for updating the system channel (using different environment variables which we will discuss below).

Make sure all of the ordering nodes in your ordering service and peers on the channel have been upgraded to the required binary level before updating capabilities. If a peer or an ordering node is not at the required level, it will be unable to process the config block with the capability and will crash. For more information, check out Capabilities.

Set environment variables

You will need to export the following variables:

- `CH_NAME`: the name of the application channel being updated. You will have to reset this variable for every channel you update.
- `CORE_PEER_LOCALMSPID`: the MSP ID of the organization proposing the channel update. This will be the MSP of your peer organization.
- `TLS_ROOT_CA`: the absolute path to the TLS cert of your peer organization.
- `CORE_PEER_MSPCONFIGPATH`: the absolute path to the MSP representing your organization.
- `ORDERER_CONTAINER`: the name of an ordering node container. When targeting the ordering service, you can target any particular node in the ordering service. Your requests will be forwarded to the leader automatically.

`Orderer` group

Navigate to Step 1: Pull and translate the config. Once you have a `modified_config.json`, add the capabilities to the `Orderer` group of the config (as listed in `capabilities.json`) using this command:

```
jq -s '.[0] * {"channel_group":{"groups":{"Orderer": {"values": {"Capabilities": .[1].orderer}}}}
```

Then, follow the steps at Step 3: Re-encode and submit the config.

Note the `mod_policy` for this capability defaults to the `MAJORITY` of the `Admins` of the `Orderer` group (in other words, a majority of the admins of the ordering service). Peer organizations can propose an update to this capability, but their signatures will not satisfy the relevant policy in this case.

`Channel` group

Navigate to Step 1: Pull and translate the config. Once you have a `modified_config.json`, add the capabilities to the `Channel` group of the config (as listed in `capabilities.json`) using this command:

```
jq -s '.[0] * {"channel_group":{"values": {"Capabilities": .[1].channel}}}' config.json ./capabil
```

Then, follow the steps at Step 3: Re-encode and submit the config.

Note that the `mod_policy` for this capability defaults to requiring signatures from both the `MAJORITY` of `Admins` in the `Application` and `Orderer` groups. In other words, both a majority of the peer organization admins and ordering service organization admins must sign this request.

`Application` group

Navigate to Step 1: Pull and translate the config. Once you have a `modified_config.json`, add the capabilities to the `Application` group of the config (as listed in `capabilities.json`) using this command:

```
jq -s '.[0] * {"channel_group":{"groups":{"Application": {"values": {"Capabilities": .[1].applica
```

Then, follow the steps at Step 3: Re-encode and submit the config.

Note that the `mod_policy` for this capability defaults to requiring signatures from the `MAJORITY` of `Admins` in the `Application` group. In other words, a majority of peer organizations will need to approve. Ordering service admins have no say in this capability.

As a result, be very careful to not change this capability to a level that does not exist. Because ordering nodes neither understand nor validate `Application` capabilities, they will approve a configuration to any level and send the new config block to the peers to be committed to their ledgers. However, the peers will be unable to process the capability and will crash. And even it was

possible to drive a corrected configuration change to a valid capability level, the previous config block with the faulty capability would still exist on the ledger and cause peers to crash when trying to process it.

This is one reason why a file like `capabilities.json` can be useful. It prevents a simple user error — for example, setting the `Application` capability to `V20` when the intent was to set it to `V2_0` — that can cause a channel to be unusable and unrecoverable.

Verify a transaction after capabilities have been enabled

It's a best practice to ensure that capabilities have been enabled successfully with a chaincode invoke on all channels. If any nodes that do not understand new capabilities have not been upgraded to a sufficient binary level, they will crash. You will have to upgrade their binary level before they can be successfully restarted.