# Amazon Managed Blockchain

## Management Guide

# Amazon Managed Blockchain: Management Guide

# Table of Contents

***This is prerelease documentation for a service in preview release. It is subject to change.*** To sign up for the preview, visit the Amazon Managed Blockchain Preview page. For help or questions, send email to: **amazon-managed-blockchain-help@amazon.com**

# What Is Amazon Managed Blockchain?

Amazon Managed Blockchain is a fully managed service for creating and managing blockchain networks using Hyperledger Fabric and Ethereum open source frameworks (only Hyperledger Fabric is available during preview). Blockchain allows you to build applications where multiple parties can securely and transparently run transactions and share data without the need for a trusted, central authority.

You can create a Managed Blockchain network using the AWS Management Console, the AWS CLI, or the Managed Blockchain SDK to create a scalable blockchain network quickly and efficiently. Managed Blockchain scales to meet the demands of thousands of applications running millions of transactions. After the network is up and running, Managed Blockchain also simplifies network management tasks. Managed Blockchain manages your certificates, lets you invite new members to join the network with a single click or command, and tracks operational metrics such as compute, memory, and storage resources. Managed Blockchain also can replicate an immutable copy of your blockchain network activity into Amazon QLDB, a fully managed ledger database, outside of the network. This lets you efficiently analyze network activity and identify trends for insight.

This guide covers the fundamentals of creating and working with resources in a Managed Blockchain network.

## About the Preview Release

Managed Blockchain is currently a preview release and available only in `us-east-1`, US East (N. Virginia). You must sign up to participate. For more information and to sign up, see Amazon Managed Blockchain Preview. For additional information, see Prerequisites and Considerations (p. 7) and Known Issues and Limits (p. 40).

Your participation in this preview is subject to the AWS Service Terms, including Section 1.10 (Beta Service Participation). You should not use the preview version of Amazon Managed Blockchain for production workloads. Please note that Content you write or provide to an Amazon Managed Blockchain network (for example, the name of your membership in the network) will be accessible to other users of the network. We reserve the right to delete any Content, network or other resources from the preview without notice, but we will try to give you notice so that your evaluation is not adversely impacted.

## How to Get Started with Managed Blockchain

We recommend the following resources to get started developing blockchain applications using Managed Blockchain:

- Key Concepts: Blockchain Networks, Members, and Peer Nodes (p. 3)

  This guide helps you understand the fundamental building blocks of a Managed Blockchain network. It also tells you how to identify and communicate with resources, regardless of the blockchain framework that you're using.
- Get Started Creating a Hyperledger Fabric Blockchain Network Using Amazon Managed Blockchain (p. 7)

This lets you try Managed Blockchain. You create your first network, set up a Hyperledger Fabric client, and use the open source Hyperledger Fabric peer CLI to query and update the ledger. You also invite another member, get them set up, and then query and update the ledger. You can get your first Hyperledger Fabric blockchain application up and running in a short time.

- Hyperledger Fabric v1.2 Documentation

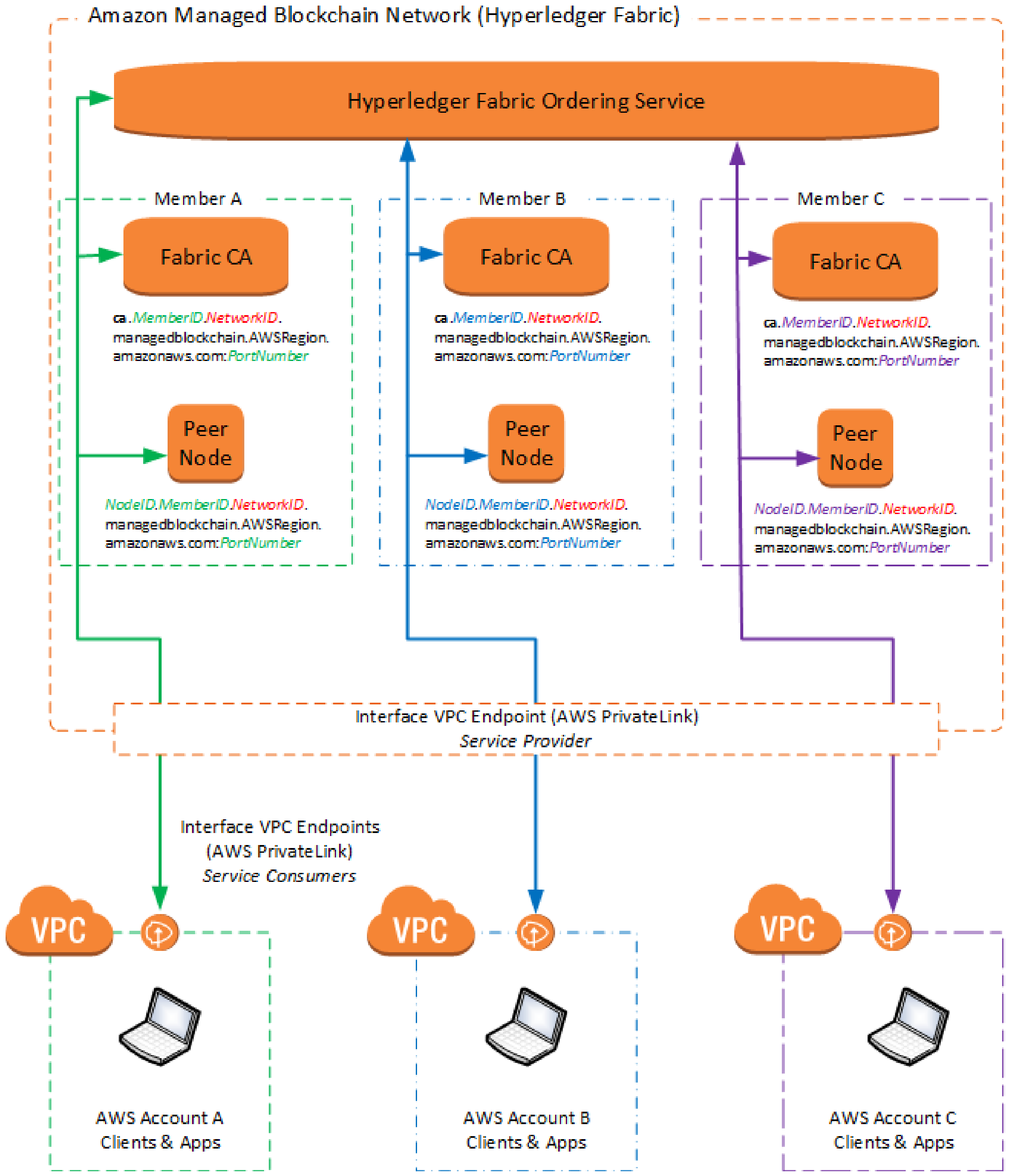
  The open source documentation for Hyperledger Fabric is a starting point for key concepts and the architecture of the Hyperledger Fabric blockchain network that you build using Managed Blockchain. As you develop your blockchain application, you can reference this document for key tasks and code samples.

# Key Concepts: Blockchain Networks, Members, and Peer Nodes

A blockchain network is a peer-to-peer network running a decentralized blockchain framework. A network includes one or more *members*, which are unique identities in the network. For example, a member might be an organization in a consortium of banks. Each member runs one or more blockchain *peer nodes* to run chaincode, endorse transactions, and store a local copy of ledger.

Amazon Managed Blockchain creates and manages these components for each member in a network, and it also creates components shared by all members in a network, such as the Hyperledger Fabric ordering service and the general networking configuration.

The following diagram shows the basic components of a Hyperledger Fabric blockchain running on Managed Blockchain:



## Managed Blockchain Networks and Members

An AWS account initially creates a Managed Blockchain network and the first member on that network. The first member can also create additional members in the same AWS account to simulate a network with multiple AWS accounts. During the preview, the first member is the only member who can invite additional AWS accounts to join the network.

A Managed Blockchain network is decentralized. It's not owned by any AWS Account. It remains active as long as there are members, even if the initial creator deletes its member. Additionally, a member cannot delete a network. A network is deleted only when the last member is deleted. Each member pays an hourly rate (billed per second) for their network membership, and also pays for peer nodes, peer node storage, and the amount of data that it writes to the network. For more information, see Managed Blockchain Pricing.

Different frameworks use slightly different terms for the identities that we call *members* in Managed Blockchain. For example, Hyperledger Fabric uses the term organizations.

## Peer Nodes

When a member joins the network, one of the first things they must do is create at least one *peer node* in the membership. The peer nodes of each member interact to create and endorse transactions proposed in the network and store a local copy of the ledger. Members define the rules in the endorsement process based on their business logic and the blockchain framework being used.

Blockchain networks contain a distributed, cryptographically secure ledger that maintains the history of transactions in the network that is immutable—it can't be changed after-the fact. Every member can independently verify this transaction history without a centralized authority because each peer node has a copy of this ledger. Each peer node also holds the global state of the network for the channels in which they participate, which gets updated with each new transaction.

To configure blockchain applications on peer nodes and to interact with other network resources, members use a client configured with open source tools such as a CLI or SDK. The applications and tools that you choose and your client setup depend on the blockchain framework that you use and your preferred development environment. For example, in the Getting Started (p. 7) tutorial, you configure an Amazon EC2 instance in a VPC with open source Hyperledger Fabric CLI tools. Regardless of the framework, the way that you identify and connect to Managed Blockchain resources using framework tools is the same.

## Identifying Managed Blockchain Resources and Connecting from a Client

Because the blockchain network is decentralized, members need to interact with each other's peer nodes and network-wide resources to make transactions, endorse transactions, verify members, and so on. When a network is created, Managed Blockchain gives the network a unique ID. Similarly, when an AWS account creates a member on the network and peer nodes, Managed Blockchain gives unique IDs to those resources.

Each network resource exposes a unique, addressable endpoint that Managed Blockchain creates from these IDs to other members in the Managed Blockchain network. Blockchain applications and tools use these endpoints to identify and interact with resources on the Managed Blockchain network.

Resource endpoints on the Managed Blockchain network are in the following format:

```
ResourceID.MemberID.NetworkID.managedblockchain.AWSRegion.amazonaws.com:PortNumber
```

For example, to refer to a peer node with id nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y, owned by a member with ID m-K46ICRRXJRCGRNNS4ES4XUUS5A, in a Hyperledger Fabric network with ID n-MWY63ZJZU5HGNCMBQER7IN6OIU, you use the following peer node endpoint:

```
nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y.m-K46ICRRXJRCGRNNS4ES4XUUS5A.n-
MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.us-east-1.amazonaws.com:PeerEventPort
```

The port that you use with an endpoint depends on the blockchain framework, and the blockchain service that you are calling. During the preview, Managed Blockchain supports only `us-east-1` for *`AWSRegion`*.

Within the blockchain network, access and authorization for each resource is governed by processes defined within the network. Outside the confines of the network—that is, from member's client applications and tools—Managed Blockchain uses AWS PrivateLink to ensure that only network members can access required resources. In this way, each member has a private connection from a client in their VPC to the Managed Blockchain network. The interface VPC endpoint uses private DNS, so you must have a VPC in your account that is enabled for Private DNS. For more information, see Create an Interface VPC Endpoint to Connect to Managed Blockchain Network Resources (p. 31).

# Get Started Creating a Hyperledger Fabric Blockchain Network Using Amazon Managed Blockchain

This tutorial guides you through creating your first Hyperledger Fabric network using Amazon Managed Blockchain. It shows you how to set up the network and create a member in your AWS account, set up chaincode and a channel, and then invite members from other AWS accounts to join a channel. Instructions for invitees is also provided.

**Steps**

## Prerequisites and Considerations

To complete this tutorial, you must have the resources listed in this section. Unless specifically stated otherwise, the requirements apply to both network creators and invited members.

**Topics**

### An AWS account

Before you use Managed Blockchain for the first time, you must sign up for an Amazon Web Services (AWS) account.

If you do not have an AWS account, use the following procedure to create one.

**To sign up for AWS**

1. Open https://aws.amazon.com/, and then choose **Create an AWS Account**.

    **Note**
    If you previously signed in to the AWS Management Console using AWS account root user credentials, choose **Sign in to a different account**. If you previously signed in to the console using IAM credentials, choose **Sign-in using root account credentials**. Then choose **Create a new AWS account**.
2. Follow the online instructions.

    Part of the sign-up procedure involves receiving a phone call and entering a verification code using the phone keypad.

## Access to the Preview

Managed Blockchain is available as a preview release. To sign up for the preview, see Amazon Managed Blockchain Preview.

## A Linux Client (EC2 Instance)

You must have a Linux computer with access to resources in the VPC to serve as your Hyperledger Fabric client. This computer must have the AWS CLI installed. We recommend creating an Amazon Elastic Compute Cloud (Amazon EC2) instance in the same VPC in which the interface VPC endpoint for the Managed Blockchain network exists and in the same AWS Region. This is the setup that is used for this tutorial.

## A VPC

Network must have a VPC with an IPv4 CIDR block and the `enableDnsHostnames` and `enableDnsSupport` options set to `true`. If you will connect to the Hyperledger Fabric client using SSH, the VPC must have an internet gateway, and the security group configuration associated with the Hyperledger Framework client must allow inbound SSH access from your SSH client.

- For more information about creating a suitable network, see Getting Started with IPv4 for Amazon VPC tutorial in the *Amazon VPC User Guide*.
- For information about using SSH to connect to an Amazon EC2 Instance, see Connecting to Your Linux Instance Using SSH in the *Amazon EC2 User Guide for Linux Instances*.
- For instructions about how to verify if DNS options are enabled, see Using DNS with Your VPC in the *Amazon VPC User Guide*.

## Permissions to Create an Interface VPC Endpoint

The IAM principal (user) identity that you are using must have sufficient IAM permissions to create an interface VPC endpoint in your AWS account. For more information, see Controlling Access - Creating and Managing VPC Endpoints in the *Amazon VPC User Guide*.

## The CLI for Managed Blockchain (Installed on the Client)

The Python CLI for Managed Blockchain must be installed on a computer, with the AWS CLI configured with access to your AWS account.

**To install the Python CLI for Managed Blockchain**

1. Use the AWS CLI to copy the `s3://us-east-1.managedblockchain-preview/etc/service-2.json file` to a local directory. For example:

   ```
   aws s3 cp s3://us-east-1.managedblockchain-preview/etc/service-2.json
   ```

2. After you download the `service-2.json` file, use the following command to add the Managed Blockchain CLI commands to the AWS CLI. The example assumes you are running the command in the same directory where `service-2.json` is saved.

   ```
   aws configure add-model --service-model file://service-2.json
   ```

## Additional Considerations

- All commands in the tutorial assume that you are using an Amazon EC2 instance with an Amazon Linux AMI. Unless noted otherwise, instructions also assume that you are running commands in the default home directory (`/home/ec2-user`). If you have a different configuration, modify instructions to fit your home directory as necessary.
- Command examples are preceded by a prompt (`[ec2-user@ip-192-0-2-17 ~]$`) where appropriate for clarity. Remove the prompt if you copy and paste examples.
- For more information, see Known Issues and Limits (p. 40).

# Step 1: Create the Network and First Member

Use the following AWS CLI command to create the Hyperledger Fabric network using Managed Blockchain or, to use the console, see Create a Managed Blockchain Network (p. 29).

When you create the network, you also must create the first member in the network using the `--member-configuration` option. Because Managed Blockchain creates a certificate authority (CA) with each member that handles authenticating users, you must provide a user name and password for the administrator. You specify these using the `AdminUsername` and `AdminPassword` properties. The password you use must be a minimum of 8 characters, and contain at least one number and one capital letter. Remember the user name and password. You need them later any time you create users and resources that need to authenticate.

Create the network using an AWS CLI command similar to the following:

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain create-network \
--network-configuration Name='MyTestTaigaNetwork,\
Description=MyTaigaNetDescription,\
Framework=HYPERLEDGER_FABRIC,FrameworkVersion=1.2' \
--member-configuration 'Name=MyFirstMembersName,\
Description=MyDesc,\
FrameworkConfiguration={Fabric={AdminUsername=AdminUser,\
AdminPassword=Password123}}'
```

The command returns output that includes the unique ID for the network and the member that you create, as shown in the following example. Make a note of these. You need them later.

```
{
    "NetworkId": "n-MWY63ZJZU5HGNCMBQER7IN6OIU",
```

```
    "MemberId": "m-K46ICRRXJRCGRNNS4ES4XUUS5A"
}
```

You can use the `list-networks` command, as shown in the following example, to confirm the network status.

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain list-networks
```

The command returns information about the network, including an `ACTIVE` status.

```
{
    "Networks": [
        {
            "Status": "ACTIVE",
            "Description": "MyNetDescription",
            "Framework": "HYPERLEDGER_FABRIC",
            "CreationDate": 1541497086.888,
            "FrameworkVersion": "1.2",
            "Id": "n-MWY63ZJZU5HGNCMBQER7IN6OIU",
            "Name": "MyTestNetwork"
        }
    ]
}
```

## Step 2: Create and Configure an Interface VPC Endpoint

Now that the network is up and running in your VPC, you set up an interface VPC endpoint (AWS PrivateLink). This allows the Amazon EC2 instance that you use as a Hyperledger Fabric client to interact with the Hyperledger Fabric endpoints that Amazon Managed Blockchain exposes for your member and network resources. For more information, see Interface VPC Endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*. Applicable charges for interface VPC endpoints apply. For more information, see AWS PrivateLink Pricing.

The AWS Identity and Access Management (IAM) principal (user) identity that you use must have sufficient IAM permissions to create an interface VPC endpoint in your AWS account. For more information, see Controlling Access - Creating and Managing VPC Endpoints in the *Amazon VPC User Guide*.

You can create the interface VPC endpoint using a shortcut in the Managed Blockchain console.

### To create an interface VPC endpoint using the Managed Blockchain console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/
2. Choose **Networks**, select your network from the list, and then choose **View details**.
3. Choose **Create VPC endpoint**.
4. Choose a **VPC**.
5. For **Subnets**, choose a subnet from the list, and then choose additional subnets as necessary.
6. For **Security groups**, choose an EC2 security group from the list, and then choose additional security groups as necessary. We recommend that you select the same security group that your framework client EC2 instance is associated with.
7. Choose **Create**.

To create the interface VPC endpoint using the Amazon VPC console, you need some network information. Use the `get-network` command. Specify the Network ID that you set up earlier, as shown in the following example:

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain get-network \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU
```

The command returns output similar to the following example:

```
{
    "Network": {
        "VpcEndpointServiceName": "com.amazonaws.vpce.us-east-1.vpce-
svc-123b4cd567e890123",
        "Status": "ACTIVE",
        "Description": "MyTaigaNetDescription",
        "FrameworkAttributes": {
            "Fabric": {
                "OrderingServiceEndpoint": "orderer.n-
MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.amazonaws.com:OrderingServicePort"
            }
        },
        "StatusReason": "Network created successfully",
        "Framework": "HYPERLEDGER_FABRIC",
        "CreationDate": 1541497086.888,
        "FrameworkVersion": "1.2",
        "Id": "n-MWY63ZJZU5HGNCMBQER7IN6OIU",
        "Name": "MyTestTaigaNetwork"
    }
}
```

You need the value of `VpcEndpointServiceName` for the following step.

**To create an interface VPC Endpoint for the Managed Blockchain network**

1. Find the **VPC endpoint service name** of the network. This value is returned by `get-network` command using the Managed Blockchain CLI, and is available on the network **Details** page using the Managed Blockchain console (choose **Networks**, select the network from the list, and then choose **View details**).
2. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
3. Choose **Endpoints**, **Create Endpoint**.
4. Choose **Find service by name**. For **Service Name**, enter the VPC Endpoint Service Name from step 1.
5. Choose **Verify** and then choose **Create endpoint**.
6. Make sure that **Enable Private DNS Name** is selected. This option is only available if the VPC you selected has **Enable DNS hostnames** and **Enable DNS support** set to true for the VPC. This is a requirement for the VPC.
7. We recommend that the EC2 security group that you specify for the VPC endpoint is the same as the EC2 security group for the blockchain client that you create to work with the Managed Blockchain network.

## Step 3: Create an EC2 Instance and Set Up the Hyperledger Fabric Client

To complete this step, you launch an Amazon EC2 instance using the Amazon Linux AMI. The Amazon EC2 instance has the following requirements:

- We recommend that you launch the client EC2 instance in the same VPC and security group as the VPC Endpoint that you created in Step 2: Create and Configure an Interface VPC Endpoint (p. 10). This simplifies connectivity between the EC2 instance and the endpoint.
- It is configured with an automatically assigned public IP address and an Amazon EC2 key pair so that you can connect to it using SSH.
- The Amazon EC2 security group that you use for the Amazon EC2 instance has a rule that allows inbound SSH connections from a source that includes your SSH client's IP address.

For more information, see Getting Started with Amazon EC2 Linux Instances.

## Step 3.1: Install Packages

Your Hyperledger Fabric client needs some packages and samples installed so that you can work with the Hyperledger Fabric resources. In this step, you install Go, Docker, Docker Compose, and some other utilities. You also create variables in the `~/.bash_profile` for your development environment. These are prerequisites for installing and using Hyperledger tools.

While connected to the Hyperledger Fabric client using SSH, run the following commands to install utilties, install docker, and configure the Docker user to be the default user for an Amazon EC2 instance:

```
[ec2-user@ip-192-0-2-17 ~]$ sudo yum update -y
[ec2-user@ip-192-0-2-17 ~]$ sudo yum install -y telnet
[ec2-user@ip-192-0-2-17 ~]$ sudo yum -y install emacs
[ec2-user@ip-192-0-2-17 ~]$ sudo yum install -y docker
[ec2-user@ip-192-0-2-17 ~]$ sudo service docker start
[ec2-user@ip-192-0-2-17 ~]$ sudo usermod -a -G docker ec2-user
```

Log out and log in again for the `usermod` command to take effect.

Run the following commands to install Docker Compose:

```
[ec2-user@ip-192-0-2-17 ~]$ sudo curl -L \
https://github.com/docker/compose/releases/download/1.20.0/docker-compose-`uname \
-s`-`uname -m` -o /usr/local/bin/docker-compose
[ec2-user@ip-192-0-2-17 ~]$ sudo chmod a+x /usr/local/bin/docker-compose
[ec2-user@ip-192-0-2-17 ~]$ sudo yum install libtool -y
```

Run the following commands to install golang:

```
[ec2-user@ip-192-0-2-17 ~]$ wget https://dl.google.com/go/go1.10.3.linux-amd64.tar.gz
[ec2-user@ip-192-0-2-17 ~]$ tar -xzf go1.10.3.linux-amd64.tar.gz
[ec2-user@ip-192-0-2-17 ~]$ sudo mv go /usr/local
[ec2-user@ip-192-0-2-17 ~]$ sudo yum install libtool-ltdl-devel -y
[ec2-user@ip-192-0-2-17 ~]$ sudo yum install git -y
```

Use a text editor to set up variables such as `GOROOT` and `GOPATH` in your `~/.bashrc` or `~/.bash_profile` and save the updates. The following example shows entries in `.bash_profile`.

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs
```

```
PATH=$PATH:$HOME/.local/bin:$HOME/bin

# GOROOT is the location where Go package is installed on your system
export GOROOT=/usr/local/go

# GOPATH is the location of your work directory
export GOPATH=$HOME/go

# Update PATH so that you can access the go binary system wide
export PATH=$GOROOT/bin:$PATH
export PATH=$PATH:/home/ec2-user/go/src/github.com/hyperledger/fabric-ca/bin
```

After you update `.bash_profile`, apply the changes:

```
[ec2-user@ip-192-0-2-17 ~]$ source ~/.bash_profile
```

After the installation, verify that you have the correct versions installed:

- Docker–17.06.2-ce or later
- Docker-compose–1.14.0 or later
- Go–1.10.x

To check the Docker version, run the following command:

```
[ec2-user@ip-192-0-2-17 ~]$ sudo docker version
```

The command returns output similar to the following:

```
Client:
 Version: 18.06.1-ce
 API version: 1.38
 Go version: go1.10.3
 Git commit: CommitHash
 Built: Tue Oct 2 18:06:45 2018
 OS/Arch: linux/amd64
 Experimental: false

Server:
 Engine:
 Version: 18.06.1-ce
 API version: 1.38 (minimum version 1.12)
 Go version: go1.10.3
 Git commit: e68fc7a/18.06.1-ce
 Built: Tue Oct 2 18:08:26 2018
 OS/Arch: linux/amd64
 Experimental: false
```

To check the version of Docker Compose, run the following command:

```
[ec2-user@ip-192-0-2-17 ~]$ sudo /usr/local/bin/docker-compose version
```

The command returns output similar to the following:

```
docker-compose version 1.22.0, build f46880fe
docker-py version: 3.4.1
CPython version: 3.6.6
OpenSSL version: OpenSSL 1.1.0f  25 May 2017
```

To check the version of go, run the following command:

```
[ec2-user@ip-192-0-2-17 ~]$ go version
```

The command returns output similar to the following:

```
go version go1.10.3 linux/amd64
```

## Step 3.2: Set Up the Hyperledger Fabric CA Client

In this step, you verify that you can connect to the Hyperledger Fabric CA using the VPC endpoint you configured in Step 2: Create and Configure an Interface VPC Endpoint (p. 10). You then install the Hyperledger Fabric CA client. The Fabric CA issues certificates to administrators and network peers.

To verify connectivity to the Hyperledger Fabric CA, you need the `CAEndpoint`. Use the `get-member` command to get the CA endpoint for your member, as shown in the following example. Replace the values of `--network-id` and `--member-id` with the values returned in Step 1: Create the Network and First Member (p. 9).

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain get-member \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
--member-id m-K46ICRRXJRCGRNNS4ES4XUUS5A
```

Use `curl` or `telnet` to verify that the endpoint resolves. In the following example, replace *`CAEndpoint`* with the **CAEndpoint** returned by the `get-member` command.

```
[ec2-user@ip-192-0-2-17 ~]$ curl https://CAEndpoint/cainfo -k
```

The command should return output similar to the following:

```
{"result":
{"CAName":"abcd1efghijkllmn5op3q52rst","CAChain":"LongStringOfCharacters","Version":"1.2.1-
snapshot-"}
,"errors":[],"messages":[],"success":true}
```

Alternatively, you can connect to the Fabric CA using Telnet, using the same endpoint in the `curl` example:

```
[ec2-user@ip-192-0-2-17 ~]$ curl https://CAEndpoint:30002
```

The command should return output similar to the following:

```
Trying 10.0.1.228...
Connected to ca.m-K46ICRRXJRCGRNNS4ES4XUUS5A.n-
MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.us-east-1.amazonaws.com.
Escape character is '^]'.
```

Now that you have verified that you can connect to the Hyperledger Fabric CA, run the following commands to configure the CA client:

```

[ec2-user@ip-192-0-2-17 ~]$ go get -u github.com/hyperledger/fabric-ca/cmd/...
[ec2-user@ip-192-0-2-17 ~]$ cd /home/ec2-user/go/src/github.com/hyperledger/fabric-ca
[ec2-user@ip-192-0-2-17 ~]$ git fetch
```

```
[ec2-user@ip-192-0-2-17 ~]$ git checkout release-1.2
[ec2-user@ip-192-0-2-17 ~]$ make fabric-ca-client
```

## Step 3.3: Clone the Samples Repository

```
[ec2-user@ip-192-0-2-17 ~]$ cd /home/ec2-user
[ec2-user@ip-192-0-2-17 ~]$ git clone https://github.com/hyperledger/fabric-samples.git
```

## Step 3.4: Configure and Run Docker Compose to Start the Hyperledger Fabric CLI

Use a text editor to create a configuration file for Docker Compose named `docker-compose-cli.yaml` in the `/home/ec2-user` directory, which you use to run the Hyperledger Fabric CLI. You use this CLI to interact with peer nodes that your member owns. Copy the following contents into the file, and then save the file in your home directory.

```
version: '2'
services:
  cli:
    container_name: cli
    image: hyperledger/fabric-tools:1.2.0
    tty: true
    environment:
      - GOPATH=/opt/gopath
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      - CORE_LOGGING_LEVEL=info # Set logging level to debug for more verbose logging
      - CORE_PEER_ID=cli
      - CORE_CHAINCODE_KEEPALIVE=10
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
    command: /bin/bash
    volumes:
        - /var/run/:/host/var/run/
        - /home/ec2-user/fabric-samples/chaincode:/opt/gopath/src/github.com/
        - /home/ec2-user:/opt/home
```

Run the following command to start the Hyperledger Fabric peer CLI container:

```
[ec2-user@ip-192-0-2-17 ~]$ docker-compose -f docker-compose-cli.yaml up -d
```

If you restarted or logged out and back in after the `usermod` command in Step 3.1: Install Packages (p. 12), you shouldn't need to run this command with `sudo`. If the command fails, you can log out and log back in. Alternatively, you can run the command using `sudo`, as shown in the following example:

```
[ec2-user@ip-192-0-2-17 ~]$ sudo /usr/local/bin/docker-compose -f docker-compose-cli.yaml
 up -d
```

# Step 4: Enroll the Member Administrator

In this step, you use a pre-configured certificate to enroll the member administrator to your member's certificate authority (CA). To do this, you must create a certificate file. You also need the endpoint for the CA of your member, and the user name and password for the administrator that you created in Step 1: Create the Network and First Member (p. 9).

## Step 4.1: Create the Certificate File

Run the following command to copy the `managedblockchain-tls-chain.pem` to the `/home/ec2-user` directory:

```
aws s3 cp s3://us-east-1.managedblockchain-preview/etc/managedblockchain-tls-chain.pem  /
home/ec2-user/managedblockchain-tls-chain.pem
```

Run the following command to test that you copied the contents to the file correctly:

```
[ec2-user@ip-192-0-2-17 ~]$ openssl x509 -noout -text -in /home/ec2-user/managedblockchain-
tls-chain.pem
```

The command should return the contents of the certificate in human-readable format.

## Step 4.2: Enroll the Member Administrator

To enroll the member administrator, you need the CA endpoint, as well as the user name and password for the administrator that you created in Step 1: Create the Network and First Member (p. 9).

Use the `get-member` command to get the CA endpoint for your membership as shown in the following example. Replace the values of `--network-id` and `--member-id` with the values returned in Step 1: Create the Network and First Member (p. 9).

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain get-member \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
--member-id m-K46ICRRXJRCGRNNS4ES4XUUS5A
```

The command returns information about the initial member that you created in the network, as shown in the following example. Make a note of the `CaEndpoint`. You also need the `AdminUsername` and password that you created along with the network.

The command returns output similar to the following:

```
{
    "Member": {
        "NetworkId": "n-MWY63ZJZU5HGNCMBQER7IN6OIU",
        "Status": "ACTIVE",
        "Description": "MyNetDescription",
        "FrameworkAttributes": {
            "Fabric": {
                "CaEndpoint": "ca.m-K46ICRRXJRCGRNNS4ES4XUUS5A.n-
MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.us-east-1.amazonaws.com:30002",
                "AdminUsername": "AdminUser"
            }
        },
        "StatusReason": "Network member created successfully",
        "CreationDate": 1542255358.74,
        "Id": "m-K46ICRRXJRCGRNNS4ES4XUUS5A",
        "Name": "org1"
    }
}
```

Use the CA endpoint, administrator profile, and the certificate file to enroll the member administrator using the `fabric-ca-client enroll` command, as shown in the following example:

```
[ec2-user@ip-192-0-2-17 ~]$ fabric-ca-client enroll \
-u https://AdminUsername:AdminPassword@SampleCAEndpointAndPort \
--tls.certfiles /home/ec2-user/managedblockchain-tls-chain.pem -M /home/ec2-user/admin-msp
```

An example command with fictitious administrator name, password, and endpoint is shown in the following example:

```
[ec2-user@ip-192-0-2-17 ~]$ fabric-ca-client-enroll \
-u https://AdminUser:Password123@ca.m-K46ICRRXJRCGRNNS4ES4XUUS5A.n-
MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.us-east-1.amazonaws.com:30002 \
--tls.certfiles /home/ec2-user/managedblockchain-tls-chain.pem -M /home/ec2-user/admin-msp
```

The command returns output similar to the following:

```
2018/11/16 02:21:40 [INFO] Created a default configuration file at /home/ec2-user/.fabric-
ca-client/fabric-ca-client-config.yaml
2018/11/16 02:21:40 [INFO] TLS Enabled
2018/11/16 02:21:40 [INFO] generating key: &{A:ecdsa S:256}
2018/11/16 02:21:40 [INFO] encoded CSR
2018/11/16 02:21:40 [INFO] Stored client certificate at /home/ec2-user/admin-msp/signcerts/
cert.pem
2018/11/16 02:21:40 [INFO] Stored root CA certificate at /home/ec2-user/admin-msp/cacerts/
ca-abcd1efghijkllmn5op3q52rst-uqz2f2xakfd7vcfewqhckr7q5m-managedblockchain-us-east-1-
amazonaws-com-30002.pem
```

### Step 4.3: Copy Certificates for the MSP

In Hyperledger Fabric, the Membership Service Provider (MSP) identifies which root CAs and intermediate CAs are trusted to define the members of a trust domain. Certificates for the administrator's MSP are in `$FABRIC_CA_CLIENT_HOME`, which is `/home/ec2-user/admin-msp` in this tutorial. Because this MSP is for the member administrator, copy the certificates from `signcerts` to `admincerts` as shown in the following example:

```
[ec2-user@ip-192-0-2-17 ~]$ cp -r admin-msp/signcerts admin-msp/admincerts
```

**Note**
It may take a minute or two after you enroll for you to be able to use your administrator certificate to create a channel with the ordering service.

## Step 5: Create a Peer Node in Your Membership

Now that you are enrolled as an administrator for your member, you can use your client to create a peer node. Your member's peer nodes interact with other members' peer nodes on the blockchain to query and update the ledger, and store a local copy of the ledger.

Wait a minute or two for the administrative permissions from previous steps to propagate, and then run the following command. Replace `--network-id` and `--member-id` values with the values from Step 1: Create the Network and First Member (p. 9):

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain create-node \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
--member-id m-K46ICRRXJRCGRNNS4ES4XUUS5A \
--node-configuration InstanceType=bc.t3.small,AvailabilityZone=us-east-1a,DataVolumeSize=10
 \
```

The command returns output that includes the peer node's `NodeID` as shown in the following example:

```
{
     "NodeId": "nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y"
}
```

The peer node takes a few minutes to provision and become healthy. You can check the node status using the `get-node` command as shown in the following example:

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain get-node \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
--member-id m-K46ICRRXJRCGRNNS4ES4XUUS5A \
--node-id nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y
```

The command returns output that includes the peer node's `Endpoint` as shown in the following example. You need this later.

```
{
    "Node": {
        "NetworkId": "n-MWY63ZJZU5HGNCMBQER7IN6OIU",
        "Status": "HEALTHY",
        "Endpoint": "nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y.m-K46ICRRXJRCGRNNS4ES4XUUS5A.n-
MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.us-east-1.amazonaws.com:PeerEventPort",
        "FrameworkAttributes": {
            "Fabric": {
                "PeerEventPort": "30004"
            }
        },
        "MemberId": "m-K46ICRRXJRCGRNNS4ES4XUUS5A",
        "StatusReason": "Node created successfully",
        "Id": "nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y",
        "AvailabilityZone": "us-east-1a",
        "CreationDate": 1542586614.352,
        "InstanceType": "bc.t3.small"
    }
}
```

## Step 6: Create a Hyperledger Fabric Channel

In Hyperledger Fabric, a ledger exists in the scope of a channel. The ledger can be shared across the entire network if every member is operating on a common channel. A channel also can be privatized to include only a specific set of participants. Members can be in your AWS account, or they can be members that you invite from other AWS accounts.

In this step, you set up a basic channel. Later on in the tutorial, in Step 8: Invite Another AWS Account to be a Member and Create a Joint Channel (p. 22), you to through a similar process to set up a channel that includes another member.

### Step 6.1: Create configtx for Hyperledger Fabric Channel Creation

The `configtx.yaml` file contains details of the channel configuration. For more information, see Channel Configuration (configtx) in the Hyperledger Fabric documentation.

Use a text editor to create a file with the following contents and save it as `configtx.yaml` on your Hyperledger File client. Replace *`MemberID`* with the MemberID you returned previously. For example *`m-K46ICRRXJRCGRNNS4ES4XUUS5A`*.

**Important**

This file is sensitive. Artifacts from pasting can cause the file to fail with marshalling errors. We recommend using `emacs` to edit it. You can also use `VI`, but before using `VI`, enter `:set paste`, press `i` to enter insert mode, paste the contents, press escape, and then enter `:set nopaste` before saving.

```
################################################################################
#
#   Section: Organizations
#
#   - This section defines the different organizational identities which will
#   be referenced later in the configuration.
#
################################################################################
Organizations:
    - &Org1
            # DefaultOrg defines the organization which is used in the sampleconfig
            # of the fabric.git development environment
        Name: MemberID
            # ID to load the MSP definition as
        ID: MemberID
        MSPDir: /opt/home/admin-msp
            # AnchorPeers defines the location of peers which can be used
            # for cross org gossip communication.  Note, this value is only
            # encoded in the genesis block in the Application section context
        AnchorPeers:
            - Host:
              Port:

################################################################################
#
#   SECTION: Application
#
#   - This section defines the values to encode into a config transaction or
#   genesis block for application related parameters
#
################################################################################
Application: &ApplicationDefaults
        # Organizations is the list of orgs which are defined as participants on
        # the application side of the network
    Organizations:

################################################################################
#
#   Profile
#
#   - Different configuration profiles may be encoded here to be specified
#   as parameters to the configtxgen tool
#
################################################################################
Profiles:
    OneOrgChannel:
        Consortium: AWSSystemConsortium
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *Org1
```

Run the following commmand to generate the configtx peer block:

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec cli configtxgen \
-outputCreateChannelTx /opt/home/mychannel.pb \
-profile OneOrgChannel -channelID mychannel \
--configPath /opt/home/
```

## Step 6.2: Set Environment Variables for Convenience

Set the following environment variables for convenience. Replace the following values as indicated:

- *`MemberID`* is the `MemberID` returned by the `get-member` command. For example, m-K46ICRRXJRCGRNNS4ES4XUUS5A.
- *`OrderingServiceEndpoint`* is the endpoint returned by the `get-network` command. For example, orderer.n-MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.amazonaws.com:*`OrderingServicePort`*.
- *`PeerNodeEndpoint`* is the `Endpoint` returned by the `get-node` command. For example, `nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y.m-K46ICRRXJRCGRNNS4ES4XUUS5A.n-MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.us-east-1.amazonaws.com:`*`PeerEventPort`*.

```
[ec2-user@ip-192-0-2-17 ~]$ export MSP_PATH=/opt/home/admin-msp
[ec2-user@ip-192-0-2-17 ~]$ export MSP=MemberID
[ec2-user@ip-192-0-2-17 ~]$ export ORDERER=OrderingServiceEndpoint
[ec2-user@ip-192-0-2-17 ~]$ export PEER=PeerNodeEndpoint
```

## Step 6.3: Create the Channel

Run the following command to create a channel using the variables that you established and the configtx peer block that you created:

```
docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_ADDRESS=$PEER" \
-e "CORE_PEER_LOCALMSPID=$MSP" \
-e "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
cli peer channel create -c mychannel \
-f /opt/home/mychannel.pb -o $ORDERER \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

## Step 6.4: Join Your Peer Node to the Channel

Run the following command to join the peer node that you created earlier to the channel:

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_ADDRESS=$PEER" \
-e "CORE_PEER_LOCALMSPID=$MSP" \
-e "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
cli peer channel join -b mychannel.block \
-o $ORDERER --cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

# Step 7: Install and Run Chaincode

This section shows you how to install sample chaincode on your peer, instantiating the chaincode, querying the chaincode, and invoking the chaincode to update values.

## Step 7.1: Install Chaincode

Run the following command to install example chaincode on the peer node:

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_LOCALMSPID=$MSP" \
-e "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
-e "CORE_PEER_ADDRESS=$PEER" \
cli peer chaincode install \
-n mycc -v v0 -p github.com/chaincode_example02/go
```

## Step 7.2: Instantiate Chaincode

Run the following command to instantiate the chaincode:

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_LOCALMSPID=$MSP" \
-e "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
-e "CORE_PEER_ADDRESS=$PEER" \
cli peer chaincode instantiate \
-o $ORDERER -C mychannel -n mycc -v v0 \
-c '{"Args":["init","a","100","b","200"]}' \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

You may have to wait a minute or two for the instantiation to propagate to the peer node. Use the following command to verify instantiation:

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_LOCALMSPID=$MSP" \
-e  "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
-e "CORE_PEER_ADDRESS=$PEER"  \
cli peer chaincode list --instantiated \
-o $ORDERER -C mychannel \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

The command returns the following when the chaincode is instantiated:

```
Get instantiated chaincodes on channel mychannel:
Name: mycc, Version: v0, Path: github.com/chaincode_example02/go, Escc: escc, Vscc: vscc
```

## Step 7.3: Query the Chaincode

You may need to wait a brief moment for the instantiation from the previous step to complete before you run the following command to query a value:

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_ADDRESS=$PEER" \
-e "CORE_PEER_LOCALMSPID=$MSP" \
-e "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
cli peer chaincode query -C mychannel \
-n mycc -c '{"Args":["query","a"]}'
```

The command should return the value of `a`, which you instantiated to a value of `100`.

### Step 7.4: Invoke the Chaincode

In the previous steps, we instantiated the key `a` with a value of `100` and queried to verify. Using the `invoke` command in the following example, we remove `10` from that initial value:

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_ADDRESS=$PEER"  -e "CORE_PEER_LOCALMSPID=$MSP" \
-e "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
cli peer chaincode invoke -C mychannel \
-n mycc -c  '{"Args":["invoke","a","b","10"]}' \
-o $ORDERER --cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

When we query again using the following command:

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_ADDRESS=$PEER" \
-e "CORE_PEER_LOCALMSPID=$MSP" \
-e "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
cli peer chaincode query -C mychannel \
-n mycc -c '{"Args":["query","a"]}'
```

The command should return the value of `a` as the new value `90`.

## Step 8: Invite Another AWS Account to be a Member and Create a Joint Channel

Now that you have a Hyperledger Fabric network set up using Amazon Managed Blockchain with an initial member in your AWS account, you are ready to add additional members. You can add members in your own account without having to send an invitation to yourself, or you can create a network invitation for a member in a different AWS account. In the steps that follow, the network creator has an initial member named `org1` and the additional member is named `org2`.

If you are creating your own member, you can skip the first step.

### Step 8.1: Invite Another AWS Account to be a Member (Optional)

Use the following command to invite another AWS account to participate as a member in the network that you created in Step 1: Create the Network and First Member (p. 9). In the example below, replace the value of the `--network-id` value with the ID of the network returned by the `list-networks` or `get-network` commands.

Invited members do not need to accept the invitation. After the invitation is created, the invited account can create a member. They need the network ID and VPC endpoint service name. For more information, see Get Networks That You Are Invited to Join (p. 33).

**Important**
The AWS account that you invite must be signed up to participate in the Amazon Managed Blockchain preview.

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain create-network-invitation \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
--invitee-account-id 123456789012
```

## Step 8.2: Set Up an Additional Member

To create an additional member for a network, the steps are similar whether you are creating a member in a Managed Blockchain network in a different AWS account or your own AWS account.

First, use the `create-member` command as shown in the following example. Replace the value of `--network-id` with the network ID provided by the network creator and other values shown as *`replaceable`*:

```
aws managedblockchain create-member \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
--member-configuration 'Name=org2,Description=MyMemberDesc,\
FrameworkConfiguration={Fabric={AdminUsername=MyAdminUsername,\
AdminPassword=Password123}}'
```

The command returns output similar to the following:

```
{
"MemberId": "m-J46DNSFRTVCCLONS9DT5TTLS2A"
}
```

Perform the steps listed below from earlier in this tutorial to configure the member. As you perform the steps, replace values with those specific to your member configuration, including the Member ID returned by the previous command. The network ID and `OrderingServiceEndpoint` are the same for all members.

### Steps to Configure a Member

- Step 2: Create and Configure an Interface VPC Endpoint (p. 10)

  Substitute the network ID and member ID from the `create-member` command above, rather than using values from network creation.
- Step 3: Create an EC2 Instance and Set Up the Hyperledger Fabric Client (p. 11)

  If you already have a Hyperledger Fabric client, you can skip most of these steps. However, you should verify connectivity to the Hyperledger Fabric CA, as described in Step 3.2: Set Up the Hyperledger Fabric CA Client (p. 14), using the new CA endpoint for the new member.
- Step 4: Enroll the Member Administrator (p. 15)
- Step 5: Create a Peer Node in Your Membership (p. 17)

## Step 8.3: Share Artifacts and Information with the Network Creator

During the preview, before a shared channel can be created, the following artifacts and information need to be shared with `org1` by `org2`:

- **org1 needs the org2 administrative certificate**—This certificate is saved the `/home/ec2-user/admin-msp/admincerts` directory on org2's Hyperledger Fabric client after Step 4: Enroll the Member Administrator (p. 15). This is referenced in the following steps as `Org2AdminCertFile`
- **org1 needs the org2 root CA**—This certificate is saved to org2's `/home/ec2-user/admin-msp/cacerts` directory on org2's Hyperledger Fabric client after the same step as previous. This is referenced in the following steps as `Org2CACertFile`

- **org1 needs the `Endpoint` of the peer node that will join the channel**—This `Endpoint` value is output by the `get-node` command after Step 5: Create a Peer Node in Your Membership (p. 17) is complete.

## Step 8.4: The Channel Creator (org1) Creates Artifacts for org2's MSP

In the following example, the channel creator is org1. The CA administrator for org1 copies the certificates from the step above to a location on the Hyperledger Fabric client computer. The Membership Service Provider (MSP) uses the certificates to authenticate the member.

On the channel creator's Hyperledger Fabric client, use the following commands to create directories to store the certificates, and then copy the certificates from the previous step to the new directories:

```
[ec2-user@ip-192-0-2-17 ~]$ mkdir /home/ec2-user/org2-msp
[ec2-user@ip-192-0-2-17 ~]$ mkdir /home/ec2-user/org2-msp/admincerts
[ec2-user@ip-192-0-2-17 ~]$ mkdir /home/ec2-user/org2-msp/cacerts

cp Org2AdminCerts /home/ec2-user/org2-msp/admincerts
cp Org2CACerts /home/ec2-user/org2-msp/cacerts
```

Org1 needs org2's member ID. You can get this by running the `list-members` command on org1's Hyperledger Fabric client as shown in the following example:

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain list-members \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU
```

The channel creator (org1) should verify that the required artifacts for channel creation are saved on the Hyperledger Fabric client as shown in the following list:

- Org1 MSP artifacts:
  - /home/ec2-user/admin-msp/signcerts/*certname*.pem
  - /home/ec2-user/admin-msp/admincerts/*certname*.pem
  - /home/ec2-user/admin-msp/cacerts/*certname*.pem
  - /home/ec2-user/admin-msp/keystore/*keyname*_sk
- Org2 MSP artifacts
  - /home/ec2-user/org2-msp/admincerts/*certname*.pem
  - /home/ec2-user/org2-msp/cacerts/*certname*.pem
- The TLS CA cert used for the Region:
  - /home/ec2-user/managedblockchain-tls-chain.pem
- Addresses of all peer nodes to join the channel for both org1 and org2.
- The respective member IDs of org1 and org2.
- A `configtx.yaml` file, which you create in the following step, saved to the `/home/ec2-user` directory on the channel creator's Hyperledger Fabric client.

  **Note**
  If you created this configtx file earlier, delete the old file, rename it, or replace it.

## Step 8.5: Create configtx for the Joint Channel

The `configtx.yaml` file contains details of the channel configuration. For more information, see Channel Configuration (configtx) in the Hyperledger Fabric documentation.

The channel creator creates this file on the Hyperledger File client. If you compare this file to the file created in Step 6.1: Create configtx for Hyperledger Fabric Channel Creation (p. 18), you see that this `configtx.yaml` specifies two members in the channel.

Use a text editor to create a file with the following contents and save it as `configtx.yaml` on your Hyperledger File client. In the example below, replace *`Member1ID`* with the member ID of org1, which was created with the network in Step 1: Create the Network and First Member (p. 9). For example *`m-K46ICRRXJRCGRNNS4ES4XUUS5A`*. Replace *`Member2ID`* with the member ID of org2, which was created with Step 8.2: Set Up an Additional Member (p. 23).

**Important**
This file is sensitive. Artifacts from pasting can cause the file to fail with marshalling errors. We recommend using `emacs` to edit it. You can also use `VI`, but before using `VI`, enter `:set paste`, press `i` to enter insert mode, paste the contents, press escape, and then enter `:set nopaste` before saving.

```
################################################################################
#
#   Section: Organizations
#
#   - This section defines the different organizational identities which will
#   be referenced later in the configuration.
#
################################################################################
Organizations:
    - &Org1
            # member id defines the organization
        Name: Member1ID
            # ID to load the MSP definition as
        ID: Member1ID
            #msp dir of org1 in the docker container
        MSPDir: /opt/home/admin-msp
            # AnchorPeers defines the location of peers which can be used
            # for cross org gossip communication.  Note, this value is only
            # encoded in the genesis block in the Application section context
        AnchorPeers:
            - Host:
              Port:
    - &Org2
        Name: Member2ID
        ID: Member2ID
        MSPDir: /opt/home/org2-msp
        AnchorPeers:
            - Host:
              Port:

################################################################################
#
#   SECTION: Application
#
#   - This section defines the values to encode into a config transaction or
#   genesis block for application related parameters
#
################################################################################
Application: &ApplicationDefaults
        # Organizations is the list of orgs which are defined as participants on
        # the application side of the network
     Organizations:

################################################################################
#
#   Profile
#
#   - Different configuration profiles may be encoded here to be specified
#   as parameters to the configtxgen tool
#
################################################################################
Profiles:
    TwoOrgChannel:
        Consortium: AWSSystemConsortium
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *Org1
                - *Org2
```

Run the following command to generate the configtx peer block:

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec cli configtxgen \
-outputCreateChannelTx /opt/home/ourchannel.pb \
-profile TwoOrgChannel -channelID ourchannel \
--configPath /opt/home/
```

## 8.6 Create the Channel

The channel creator (org1) uses the following command on their Hyperledger Fabric client to create the channel `ourchannel`. The command example assumes that environment variables have been configured as described in Step 6.2: Set Environment Variables for Convenience (p. 20).

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_ADDRESS=$PEER" \
-e "CORE_PEER_LOCALMSPID=$MSP" \
-e "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
cli peer channel create -c ourchannel \
-f /opt/home/ourchannel.pb -o $ORDERER \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

## Step 8.7: Get Channel Genesis Block

A member who joins the channel must get the channel genesis block. In this example, org2 runs the following command from their Hyperledger Fabric client to get the genesis block.

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_ADDRESS=$PEER" \
-e "CORE_PEER_LOCALMSPID=$MSP" -e "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
cli peer channel fetch oldest /opt/home/ourchannel.block \
-c ourchannel -o $ORDERER \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

## Step 8.8: Join Peer Nodes to the Channel

Both org1 and org2 need to run the following command on their respective Hyperledger Fabric clients to join their peer nodes to the channel:

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_ADDRESS=$PEER" \
-e "CORE_PEER_LOCALMSPID=$MSP" \
-e "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
cli peer channel join -b ourchannel.block \
-o $ORDERER --cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

## Step 8.9: Install Chaincode

Both org1 and org2 run the following command on their respective Hyperledger Fabric clients to install example chaincode on their respective peer nodes:

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_LOCALMSPID=$MSP" \
-e "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
-e "CORE_PEER_ADDRESS=$PEER" \
cli peer chaincode install  -n mycc -v v0 \
-p github.com/chaincode_example02/go
```

## Step 8.10: Instantiate Chaincode

The channel creator (org1) runs the following command to instantiate the chaincode with an endorsement policy that requires both org1 and org2 to endorse all transactions. Replace *Member1ID* with the member ID of org1 and *Member2ID* with the member ID of org2. You can use the `list-members` command to get them.

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_LOCALMSPID=$MSP" \
-e  "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
-e "CORE_PEER_ADDRESS=$PEER" \
cli peer chaincode instantiate -o $ORDERER \
-C ourchannel -n mycc -v v0 \
-c '{"Args":["init","a","100","b","200"]}' \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls \
-P "AND ('Member1ID.member','Member2ID.member')"
```

You may need to wait a brief moment for the instantiation from the previous step to complete before you run the following command to query a value:

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_ADDRESS=$PEER" \
-e "CORE_PEER_LOCALMSPID=$MSP" \
-e "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
cli peer chaincode query -C mychannel \
-n mycc -c '{"Args":["query","a"]}'
```

The command should return the value of `a`, which you instantiated to a value of `100`.

## Step 8.11: Invoke Chaincode

With the channel created and configured with both members, and the chaincode instantiated with values and an endorsement policy, channel members can invoke chaincode. This example command is similar to the example in Step 7.4: Invoke the Chaincode (p. 22). However, the command uses the `--peerAddresses` option to specify the endpoints of peer nodes that belong to members in the endorsement policy. The example specifies *Org2PeerNodeEndpoint* in addition to `$PEER`, which indicates the command is run from an org1 Hyperledger Fabric client. If the command runs from the org1 Hyperledger Fabric client, *Org2PeerNodeEndpoint* is specified.

```
docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_LOCALMSPID=$MSP" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_ADDRESS=$PEER" \
-e "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
cli peer chaincode invoke \
-C ourchannel -n mycc -c '{"Args":["invoke","a","b","10"]}' \
--peerAddresses $PEER \
--tlsRootCertFiles /opt/home/managedblockchain-tls-chain.pem \
--peerAddresses Org2PeerNodeEndpoint \
--tlsRootCertFiles /opt/home/managedblockchain-tls-chain.pem \
-o $ORDERER --cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

When we query again using the following command:

```
[ec2-user@ip-192-0-2-17 ~]$ docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_ADDRESS=$PEER" \
-e "CORE_PEER_LOCALMSPID=$MSP" \
-e "CORE_PEER_MSPCONFIGPATH=$MSP_PATH" \
cli peer chaincode query -C mychannel \
-n mycc -c '{"Args":["query","a"]}'
```

The command should return the value of `a` as the new value `90`.

# Create an Amazon Managed Blockchain Network

When a user in an AWS account creates a blockchain network on Amazon Managed Blockchain, they also create the first member in the network. This first member has no peer nodes associated with it until you create them. After you create the network and the first member, you can invite members in other AWS accounts. During the preview, the first member is the only member who can invite additional AWS accounts to join the network. The first member can also create additional members in the same AWS account to simulate a network with multiple AWS accounts.

When you create the network and the first member in your AWS account, the network exists. However, transactions cannot be conducted and the ledger does not exist because there are no peer nodes. Do the following tasks to make your network functional:

- Create an interface VPC endpoint based on the network's **VPC service name** so that you can privately connect to resources. For more information, see Create an Interface VPC Endpoint to Connect to Managed Blockchain Network Resources (p. 31).
- Create at least one peer node in your first membership to interact with the network and to create and endorse transactions. For more information, see Create Peer Nodes (p. 35).
- Invite other AWS accounts to be members of the network, or create additional members in your account to simulate a multi-AWS account network. For more information about inviting members, see Invite AWS Accounts to Join the Network (p. 30) later in this topic.

## Create a Managed Blockchain Network

You can create a Managed Blockchain network using the AWS Management Console, the AWS CLI, or the Managed Blockchain SDK CreateNetwork action.

**To create a Managed Blockchain network using the AWS Management Console**

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/
2. Choose **Create a network**.
3. Under **Blockchain frameworks**, select the blockchain framework to use. Currently, **Hyperledger Fabric version 1.2** is available.
4. Enter a **Network name and description**, and choose **Next**.
5. Under **Create member**, do the following to define the network's first member, which you own:

   a. Enter a **Member name** that will be visible to all members and an optional **Description**.
   b. Under **Hyperledger Fabric certificate authority (CA) configuration** specify a username and password to be used as the administrator on the Hyperledger Fabric CA. You need to provide this user name and password when working with resources and users in your membership.
   c. Choose **Next**.
6. Review **Network options** and **Member options**, and then choose **Create network and member**.

   The **Networks** list shows the name and **Network ID** of the network you created, with a **Status** of **Creating**. It may take a minute or two for Managed Blockchain to create your network, after which the **Status** is **Active**.

**To create a Managed Blockchain network using the AWS CLI**

- Use the `create-network` command. The following example creates a Hyperledger Fabric blockchain network on Managed Blockchain.

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain create-network \
--network-configuration Name='MyTestTaigaNetwork,\
Description=MyTaigaNetDescription\
,Framework=HYPERLEDGER_FABRIC,FrameworkVersion=1.2' \
--member-configuration 'Name=MyFirstMembersName,\
Description=MyDesc,\
FrameworkConfiguration={Fabric={CaAdminUsername=AdminUser,\
CaAdminPassword=Password123}}'
```

## Invite AWS Accounts to Join the Network

During the preview, only the first member of a Managed Blockchain network can create an invitation to another AWS account using the AWS CLI, the AWS Management Console, or the Managed Blockchain SDK CreateNetworkInvitation action.

For more information about how to create a member after being invited, see Get Networks That You Are Invited to Join (p. 33).

**To invite another AWS account to create a member on the network using the Managed Blockchain console**

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/
2. Choose **Networks**, select a network from the list, and then choose **View details**.
3. Choose **Invite account**, enter the AWS account number, and then choose **Invite**.

   A green status bar indicates that the invitation was successful.

**To invite another AWS account to create a member on the network using the AWS CLI**

- Use the `create-network-invitation` command, as shown in the following example:

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain create-network-invitation --network-
id n-MWY63ZJZU5HGNCMBQER7IN6OIU --invitee-account-id 123456789012
```

# Create an Interface VPC Endpoint to Connect to Managed Blockchain Network Resources

Each member of a blockchain network on Managed Blockchain needs to privately access resource endpoints from their client applications and tools. Amazon Managed Blockchain uses Interface VPC Endpoints (AWS PrivateLink) to accomplish this.

Managed Blockchain creates a *VPC service name* for each network when it is created. Each Managed Blockchain network is a unique *endpoint service* with its own VPC service name. Each member then uses the VPC service name to create an interface VPC endpoint in their account. This interface VPC endpoint lets you access resources on the Managed Blockchain network through their endpoints. AWS accounts that are not invited to the network don't have access to the VPC service name and cannot set up an interface VPC endpoint for access.

The IAM principal (user) identity that you are using must have sufficient IAM permissions to create an interface VPC endpoint in your AWS account. For more information, see Controlling Access - Creating and Managing VPC Endpoints in the *Amazon VPC User Guide*.

Any blockchain framework clients that access resources on the network need access to the interface VPC endpoint. For example, if you use an Amazon EC2 instance as a blockchain framework client, you can create it in a subnet and security group that are shared with the interface VPC endpoint.

Applicable charges for interface VPC endpoints apply. For more information, see AWS PrivateLink Pricing.

The interface VPC endpoint that you set up to access a Managed Blockchain network must be enabled for private DNS names. This requires that you create the endpoint in a VPC that has the `enableDnsHostnames` and `enableDnsSupport` options set to `true`.

**To create an interface VPC endpoint using the Managed Blockchain console**

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/
2. Choose **Networks**, select your network from the list, and then choose **View details**.
3. Choose **Create VPC endpoint**.
4. Choose a **VPC**.
5. For **Subnets**, choose a subnet from the list, and then choose additional subnets as necessary.
6. For **Security groups**, choose an EC2 security group from the list, and then choose additional security groups as necessary. We recommend that you select the same security group that your framework client EC2 instance is associated with.
7. Choose **Create**.

**To create an interface VPC Endpoint for the Managed Blockchain network**

1. Find the **VPC endpoint service name** of the network. This value is returned by `get-network` command using the Managed Blockchain CLI, and is available on the network **Details** page using the Managed Blockchain console (choose **Networks**, select the network from the list, and then choose **View details**).
2. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.

3. Choose **Endpoints**, **Create Endpoint**.
4. Choose **Find service by name**. For **Service Name**, enter the VPC Endpoint Service Name from step 1.
5. Choose **Verify** and then choose **Create endpoint**.
6. Make sure that **Enable Private DNS Name** is selected. This option is only available if the VPC you selected has **Enable DNS hostnames** and **Enable DNS support** set to true for the VPC. This is a requirement for the VPC.
7. We recommend that the EC2 security group that you specify for the VPC endpoint is the same as the EC2 security group for the blockchain client that you create to work with the Managed Blockchain network.

# Create a Member and Peer Nodes

In Amazon Managed Blockchain, a member is a distinct identity within the blockchain network. During the preview, a member must be invited to join the network by the network creator. Only other AWS accounts can be invited as members. The network creator can create additional members in their account to simulate a Managed Blockchain network with multiple AWS accounts.

Each member pays an hourly charge, which is billed per second, for their network membership, peer nodes, and peer node storage. Charges also apply to the amount of data written to the network. For more information, see Amazon Managed Blockchain Pricing.

The resources associated with a member's account depend on the specific blockchain framework and application requirements. At minimum, all members must have at least one peer node to actively participate in the blockchain network. When you create a member it has no peer nodes by default. You create peer nodes after you create the member.

**Topics**

## Get Networks That You Are Invited to Join

If you are invited to join a Managed Blockchain network by another AWS account, you can simply create a member on that network. You need the Network ID, which you can get using the AWS Management Console, the AWS CLI, or the Managed Blockchain SDK ListInvitedNetworks action. You also need the VPC endpoint service name so that you can create an interface VPC endpoint to access the network. If you are creating a member in a network that you created, you can skip these steps.

### To list networks to which your AWS account has been invited using the console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/
2. Choose **Invited networks**.

### To list networks to which your AWS account has been invited using the AWS CLI

- Use the following command:

```
aws managedblockchain list-invited-networks
```

After you have the required network ID, you can create a member, as described later in this topic. If you are using the Managed Blockchain console, you can select the network from the **Invited networks** list and create a member. After you create a member in the network, it moves from the **Invited networks** list to the **Networks** list.

You need the **VPC endpoint service name** of the Managed Blockchain network so that you can create an interface VPC endpoint to access network resources. For more information, see Create an Interface VPC Endpoint to Connect to Managed Blockchain Network Resources (p. 31).

You can use the `get-network` AWS CLI command to get the **VPC endpoint service name** and other information about a network to which you have been invited. You only need the Network ID. Replace *n-MWY63ZJZU5HGNCMBQER7IN6OIU* with the ID of the network to which you were invited:

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain get-network \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU
```

The command returns output similar to the following example:

```
{
    "Network": {
        "VpcEndpointServiceName": "com.amazonaws.vpce.us-east-1.vpce-
svc-123b4cd567e890123",
        "Status": "ACTIVE",
        "Description": "MyTaigaNetDescription",
        "FrameworkAttributes": {
            "Fabric": {
                "OrderingServiceEndpoint": "orderer.n-
MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.amazonaws.com:OrderingServicePort"
            }
        },
        "StatusReason": "Network created successfully",
        "Framework": "HYPERLEDGER_FABRIC",
        "CreationDate": 1541497086.888,
        "FrameworkVersion": "1.2",
        "Id": "n-MWY63ZJZU5HGNCMBQER7IN6OIU",
        "Name": "MyTestTaigaNetwork"
    }
}
```

## Create a Member in a Managed Blockchain network

You can create a member in a Managed Blockchain network that you are invited to or that you created. You can use the Managed Blockchain console, the AWS CLI, or the Managed Blockchain SDK CreateMember action to create a member. If you created the Managed Blockchain network, the first member of the network is created automatically and you are the owner. To make the member functional, you must create at least one peer node for it. For more information, see Create Peer Nodes (p. 35).

### To create a member using the Managed Blockchain console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/
2. Choose **Invited networks** if you are creating a member in a network outside your AWS account.

   —or—

   Choose **Networks** if you are creating an additional member in a network where your account already has a member.
3. Select the network you for which you want to create a member, and then choose **View details**.
4. Choose **Create member**.
5. Enter a **Member name** that will be visible to all members and an optional **Description**.
6. Under **Hyperledger Fabric certificate authority (CA) configuration** specify a username and password to be used as the administrator on the Hyperledger Fabric CA. You need to provide this user name and password when working with resources and users in your membership.

7. Choose **Next**.

### To create a member using the AWS CLI

- Use the `create-member` command, as shown in the following example. Replace the value of `--network-id` with the network ID provided by the network creator and other values shown as *`replaceable`*. The password you specify must be a minimum of 8 characters, and use at least one number and one uppercase letter.

  ```
  aws managedblockchain create-member \
  --network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
  --member-configuration 'Name=org2,Description=MyMemberDesc,\
  FrameworkConfiguration={Fabric={AdminUsername=MyAdminUsername,\
  AdminPassword=Password123}}'
  ```

  The command returns output similar to the following.

  ```
  {
  "MemberId": "m-J46DNSFRTVCCLONS9DT5TTLS2A"
  }
  ```

After you create the member, you can use the `get-member` command to return important details about the member configuration and details of the member configuration are listed.

## Create Peer Nodes

A new member has no peer nodes. Peer nodes are essential. They keep a local copy of the shared ledger, let you query the ledger, and interact with clients and other peer nodes to perform transactions. You should create at least one peer node per member. During the preview, you can create one peer node per member.

When you create a peer node, you select the following characteristics:

- **Blockchain instance type**

  This determines the computational and memory capacity allocated to this node for the blockchain workload. You will be able to choose more CPU and RAM if you anticipate a more demanding workload for each node. For example, your nodes may need to process a higher rate of transactions. Different instance classes are subject to different pricing. During the preview, only one blockchain instance type, `bc.t3.small`, is available.
- **Allocated storage**

  This is the amount of storage in GiB that is available to the peer node for storing local copies of the ledger. During the preview, the minimum is 10 GiB and the maximum is 30 GiB. Storage rates apply.
- **Availability Zone**

  You can select the Availability Zone to launch the peer node in. If you distribute peer nodes in a membership across different Availability Zones for peer nodes in your membership let you design your blockchain application for resiliency. For more information, see Regions and Availability Zones in the *Amazon EC2 User Guide for Linux Instances*.

You can create a peer node in a member that is in your AWS account using the AWS Management Console, the AWS CLI, or the Managed Blockchain SDK CreateNode action.

### To create a peer node using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/
2. Choose **Networks**, select the network from the list, and then choose **View details**.
3. Select a **Member** from the list, and then choose **Create peer node**.
4. Choose configuration parameters for your peer node according to the previous guidelines, and then choose **Create peer node**.

### To create a peer node using the AWS CLI

- Use the `create-node` command, as shown in the following example. Replace the value of `--network-id` and `--member-id` and `AvailabilityZone` as appropriate. Currently, the only instance type available is b3.t3.small.

  ```
  [ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain create-node \
  --node-configuration '{"InstanceType":"bc.t3.small","AvailabilityZone":"us-east-1a"}' \
  --network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
  --member-id m-K46ICRRXJRCGRNNS4ES4XUUS5A
  ```

  The command returns output that includes the peer node's `NodeID`, as shown in the following example:

  ```
  {
       "NodeId": "nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y"
  }
  ```

The peer node takes a few minutes to provision and become healthy. You can check the node status using the `get-node` command, as shown in the following example:

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain get-node \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
--member-id m-K46ICRRXJRCGRNNS4ES4XUUS5A \
--node-id nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y
```

The command returns output that includes the peer node's `Endpoint`, as shown in the following example. The Hyperledger Fabric peer event port is 30004. You need this endpoint and port when communicating with the node using your blockchain framework client or addressing the node within an application.

```
{
    "Node": {
        "NetworkId": "n-MWY63ZJZU5HGNCMBQER7IN6OIU",
        "Status": "HEALTHY",
        "Endpoint": "nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y.m-K46ICRRXJRCGRNNS4ES4XUUS5A.n-
MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.us-east-1.amazonaws.com:PeerEventPort",
        "FrameworkAttributes": {
            "Fabric": {
                "PeerEventPort": "30004"
            }
        },
        "MemberId": "m-K46ICRRXJRCGRNNS4ES4XUUS5A",
        "StatusReason": "Node created successfully",
        "Id": "nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y",
        "AvailabilityZone": "us-east-1a",
        "CreationDate": 1542586614.352,
        "InstanceType": "bc.t3.small"
```

```
    }
}
```

# Configure a Blockchain Framework Client

You access services and applications in the Managed Blockchain network from a blockchain framework client. The framework client runs tools and applications that you install for the blockchain framework, and framework version, that you run on the Managed Blockchain network.

The client accesses Managed Blockchain network resource endpoints using an interface VPC endpoint that you set up in your account. For more information, see Create an Interface VPC Endpoint to Connect to Managed Blockchain Network Resources (p. 31). The client must have access to the interface VPC endpoint.

You can get the endpoints that networks, members, and clients make available using the AWS Management Console, or using `get` commands and actions with the AWS CLI or Managed Blockchain SDK. The available endpoints depend on the blockchain framework and may vary from client to client.

# Document History for Amazon Managed Blockchain Management Guide

The following table describes important additions to the Amazon Managed Blockchain Management Guide. For notification about updates to this documentation, you can subscribe to the RSS feed.

| update-history-change | update-history-description | update-history-date |
|---|---|---|
| Updates to getting started steps | Removed redundant steps in 3.2. The step to update .bash_profile with path to fabric-ca was already covered in step 3.1. | December 3, 2018 |
| Initial release of Amazon Managed Blockchain (Preview) | Initial documentation for Amazon Managed Blockchain. | November 28, 2018 |

# Known Issues and Limits

The following are known issues and limits during the preview release of Managed Blockchain:

**Limits**

- Availability is limited to `us-east-1`.
- Hyperledger Fabric is the only supported blockchain framework.
- The maximum number of networks per AWS account is 1.
- The maximum number of networks that an AWS account can be a member of is 1.
- The maximum number of members per network is 5.
- The maximum number of peer nodes per member is 1.
- Peer nodes are limited to the `bc.t3.small` instance type.

**Known Issues**

- There are currently no known issues.

# AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.