Amazon Managed Blockchain

Hyperledger Fabric Developer Guide

Amazon Managed Blockchain: Hyperledger Fabric Developer Guide

Table of Contents

What Is Amazon Managed Blockchain?

Amazon Managed Blockchain is a fully managed service for creating and managing blockchain networks and network resources using open-source frameworks. 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 use Managed Blockchain to create scalable blockchain resources and networks quickly and efficiently using the AWS Management Console, the AWS CLI, or the Managed Blockchain SDK.

Managed Blockchain scales to meet the demands of thousands of applications running millions of transactions. Managed Blockchain also simplifies the management of blockchain networks and resources after they are up and running. Managed Blockchain manages your certificates, lets you easily create proposals for a vote among network members where applicable, and helps you track operational metrics related to requests, computational load, memory usage, and data storage.

This guide covers the fundamentals of creating and working with a Hyperledger Fabric blockchain network using Managed Blockchain. For information about working with Ethereum on Managed Blockchain, see Ethereum on Amazon Managed Blockchain Developer Guide.

How to Get Started with Hyperledger Fabric on Managed Blockchain

We recommend the following resources to get started with Hyperledger Fabric networks and chaincode on Managed Blockchain:

- Key Concepts: Amazon Managed Blockchain Networks, Members, and Peer Nodes (p. 2)

  This overview helps you understand the fundamental building blocks of a Hyperledger Fabric network on Managed Blockchain. It also tells you how to identify and communicate with network resources.
- Get Started Creating a Hyperledger Fabric Blockchain Network Using Amazon Managed Blockchain (p. 6)

  Use this tutorial to create your first Hyperledger Fabric network, set up a Hyperledger Fabric client on EC2, and use the open-source Hyperledger Fabric peer CLI to query and update the ledger. You then invite another member to the network. The member can be from a different AWS account, or you can invite a new member in your own account to simulate a multi-account network. The new member then queries and updates the ledger.
- Hyperledger Fabric Documentation (v1.4)

  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. Use the documentation version that corresponds to the version of Hyperledger Fabric that you use.

Key Concepts: Amazon Managed Blockchain Networks, Members, and Peer Nodes

A blockchain network is a peer-to-peer network running a decentralized blockchain framework. A Hyperledger Fabric network on Amazon Managed Blockchain includes one or more *members*. Members are unique identities in the network. For example, a member might be an organization in a consortium of banks. A single AWS account might have multiple members. Each member runs one or more Hyperledger Fabric *peer nodes*. The peer nodes 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. Managed Blockchain also creates components that all network members share, such as the Hyperledger Fabric ordering service and the general networking configuration.

> **Note**
> What we call *members* in a Hyperledger Fabric network on Managed Blockchain is very similar to what Hyperledger Fabric calls *organizations*.

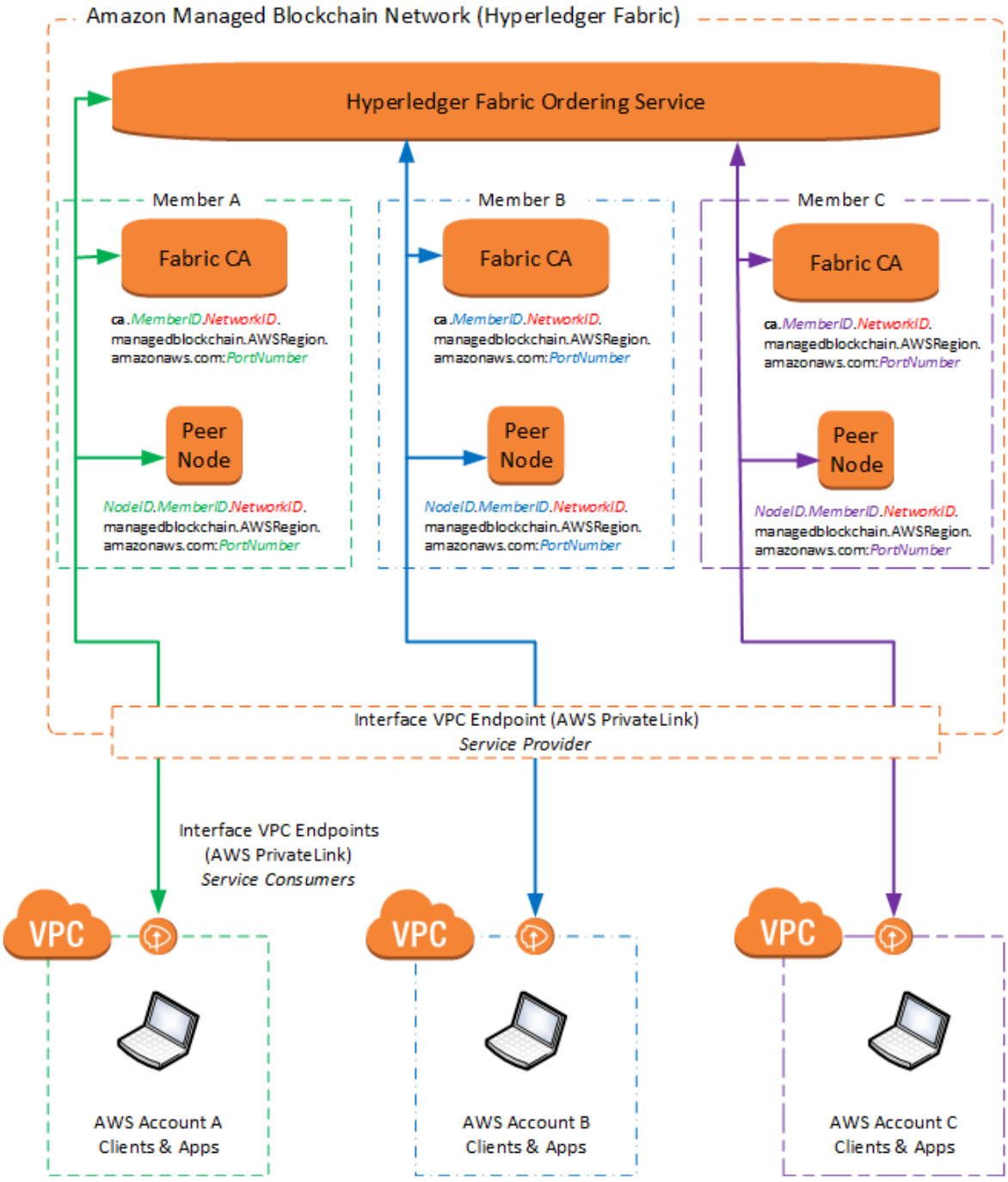
Hyperledger Fabric on Managed Blockchain Networks and Editions

When creating a Hyperledger Fabric network, the creator chooses the framework version and the *edition* of Amazon Managed Blockchain to use. The edition determines the capacity and capabilities of the network as a whole.

The creator also must create the first network member. Additional members are added through a proposal and voting process. There is no charge for the network itself, but each member pays an hourly rate (billed per second) for their network membership. Charges vary depending on the edition of the network. Each member also pays for peer nodes, peer node storage, and the amount of data that the member writes to the network. For more information about available editions and their attributes, see Managed Blockchain Pricing. For more information about the number of networks that each AWS account can create and join, see Managed Blockchain Limits in the *AWS General Reference*.

A Hyperledger Fabric network on Managed Blockchain remains active as long as there are members. The network is deleted only when the last member deletes itself from the network. No member or AWS account, even the creator's AWS account, can delete the network until they are the last member and delete themselves.

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



Inviting and Removing Members

An AWS account initially creates a Hyperledger Fabric network on Managed Blockchain, but the network is not owned by that AWS account or any other AWS account. The network is decentralized, so changes to the network are made by consensus.

To make changes to the network, members make *proposals* that all other members in the network vote on. For another AWS account to join the network, for example, an existing member creates a proposal to

invite the account. Other members then vote Yes or No on the proposal. If the proposal is approved, an invitation is sent to the AWS account. The account then accepts the invitation and creates a member to join the network. A similar proposal process is required to remove a member in a different AWS account. A principal in an AWS account with sufficient permissions can remove a member that the account owns at any time by deleting that member directly, without submitting a proposal.

The network creator also defines a *voting policy* for the network during creation. The voting policy determines the basic rules for all proposal voting on the network. The voting policy includes the percentage of votes required to pass the proposal, and the duration before the vote expires.

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.

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. Each peer node also holds the global state of the network for the channels in which they participate. The global state is updated with each new transaction. When a new peer node in a channel comes online, it fetches the global state and ledger from other peers. Even if there are no other peer nodes on a network, as long as a member exists, ledger data can be restored to a new peer node.

Peer nodes also interact to create and endorse the transactions that are proposed on the network to update the ledger. Members define the rules in the endorsement process based on their business logic. In this way, every member can conduct transactions as allowed by the business logic and independently verify the transaction history without a centralized authority.

Note
Limit transactions to less than 4 MB. Transactions greater than 4 MB result in an error.

To configure Hyperledger Fabric applications on peer nodes and to interact with other network resources, members use a client configured with open-source Hyperledger Fabric tools such as a CLI or SDK. The applications and tools that you choose and your client setup depend on your preferred development environment. For example, in the Getting Started (p. 6) tutorial, you configure an Amazon EC2 instance in a VPC with open-source Hyperledger Fabric CLI tools.

Identifying Managed Blockchain Resources and Connecting from a Client

Because a Hyperledger Fabric blockchain network is decentralized, members must 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 has a unique, addressable endpoint that Managed Blockchain creates from these IDs. Other members of the network, Hyperledger Fabric chaincode, and other tools use these endpoints to identify and interact with resources on the network.

Resource endpoints for a Hyperledger Fabric network on Managed Blockchain 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:30003
```

The port that you use with an endpoint depends on the Hyperledger Fabric service that you are calling and your unique network setup. *AWSRegion* is the Region you are using. For a list of supported Regions, see Amazon Managed Blockchain Endpoints and Quotas in the *Amazon Web Services General Reference*.

Within the Hyperledger Fabric network, access and authorization for each resource is governed by processes defined in the chaincode and network configurations such as Hyperledger Fabric channels. 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 Hyperledger Fabric network on Managed Blockchain. 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 for Hyperledger Fabric on Amazon Managed Blockchain (p. 43).

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 are 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, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

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

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 version 1.16.149 or later of the AWS CLI installed. Earlier versions of the AWS CLI do not have the `managedblockchain` command. We recommend that you use the latest version of the AWS CLI available. For information about updating the AWS CLI, see Update the AWS CLI version 2 on Linux in the *AWS Command Line Interface User Guide*.

We recommend creating an Amazon Elastic Compute Cloud (Amazon EC2) instance in the same VPC and AWS Region as the VPC endpoint for the Hyperledger Fabric network on Managed Blockchain. This is the setup that the tutorial uses. For instructions to set up a Hyperledger Fabric client using this configuration, see Step 4: Create an Amazon EC2 Instance and Set Up the Hyperledger Fabric Client (p. 12).

An AWS CloudFormation template to create a Hyperledger Fabric client is available in amazon-managed-blockchain-client-templates repository on Github. For more information, see the readme.md in that repository. For more information about using AWS CloudFormation, see Getting Started in the *AWS CloudFormation User Guide*.

A VPC

You must have a VPC with an IPv4 CIDR block, and the `enableDnsHostnames` and `enableDnsSupport` options must be 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*.

EC2 Security Groups That Allow Communication on Required Ports

The EC2 security groups associated with the Hyperledger Fabric client Amazon EC2 instance and the Interface VPC Endpoint that you create during this tutorial must have rules that allow traffic between

them for required Hyperledger Fabric services. EC2 security groups are restrictive by default, so you need to create security group rules that allow required access. In addition, a security group associated with the Hyperledger Fabric client Amazon EC2 instance must have an inbound rule that allows SSH traffic (Port 22) from trusted SSH clients.

For the purposes of simplicity in this tutorial, we recommend that you create an EC2 security group that you associate only with the Hyperledger Fabric client Amazon EC2 instance and the Interface VPC Endpoint. Then create an inbound rule that allows all traffic from within the security group. In addition, create another security group to associate with the Hyperledger Fabric client Amazon EC2 instance that allows inbound SSH traffic from trusted clients.

> **Important**
> This security group configuration is recommended for this tutorial only. Carefully consider security group settings for your desired security posture. For information about the minimum required rules, see Configuring Security Groups for Hyperledger Fabric on Amazon Managed Blockchain (p. 111).

To create a security group that allows traffic between the Hyperledger Fabric client and the interface VPC endpoint for use in this tutorial

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose **Security groups** in the navigation pane, and then choose **Create security group**.
3. Enter a **Security group name** and **Description** for the security group that helps you find it. For example, **`HFClientAndEndpoint`**.
4. Make sure that the VPC you select is the default VPC for your account. This is the VPC in which Hyperledger Fabric network resources and the interface VPC endpoint are created.
5. Choose **Create**.
6. Select the security group that you just created from the list, choose **Inbound**, and then choose **Edit**.
7. Under **Type**, select **All traffic** from the list.
8. Under **Source**, leave **Custom** selected, and then begin typing the name or ID of this same security group—for example, **HFClientAndEndpoint**—and then select the security group so that its ID appears under **Source**.
9. Choose **Save**.

   You reference this security group later in this tutorial in Step 2: Create and Configure the Interface VPC Endpoint (p. 11) and Step 4: Create an Amazon EC2 Instance and Set Up the Hyperledger Fabric Client (p. 12).

To create a security group for the Hyperledger Fabric client that allows inbound SSH connections from the computer that you are working with

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose **Security groups** in the navigation pane, and then choose **Create security group**.
3. Enter a **Security group name** and **Description** for the security group that helps you find it. For example, **`HFClientSSH`**.
4. Make sure that the VPC you select is the same VPC that you will select for the interface VPC endpoint.
5. Choose **Inbound**, and then choose **Add rule**.
6. Under **Type**, select **SSH** from the list.
7. Under **Source**, select **My IP**. This adds the detected IP address of your current computer. Optionally, you can create additional rules for SSH connections from additional IP addresses or sources if required.

8. Choose **Create**.

   You will reference this security group later in this tutorial in Step 4: Create an Amazon EC2 Instance and Set Up the Hyperledger Fabric Client (p. 12).

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.
- Hyperledger Fabric requires that a channel ID contain only lowercase ASCII alphanumeric characters, dots (.), and dashes (-). It must start with a letter, and must be fewer than 250 characters.

Step 1: Create the Network and First Member

When you create the network, you specify the following parameters along with basic information such as names and descriptions:

- The open-source framework and version. This tutorial uses Hyperledger Fabric version 1.4.
- The voting policy for proposals on the network. For more information, see Work with Proposals for a Hyperledger Fabric Network on Amazon Managed Blockchain (p. 51).
- The first member of the network, including the administrative user and administrative password that are used to authenticate to the member's certificate authority (CA).

  Important
  Each member that is created accrues charges according to the membership rate for the network. For more information, see Amazon Managed Blockchain Pricing.

Create the network using the AWS CLI or Managed Blockchain console according to the following instructions. It takes around 30 minutes for Managed Blockchain to provision resources and bring the network online.

To create a Hyperledger Fabric network using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. Choose **Create private network**.
3. Under **Blockchain frameworks**:

   a. Select the blockchain framework to use. This tutorial is based on **Hyperledger Fabric version** .

   b. Select the **Network edition** to use. The network edition determines attributes of the network, such as the maximum number of members, nodes per member, and transaction throughput. Different editions have different rates associated with the membership. For more information, see Amazon Managed Blockchain Pricing.
4. Enter a **Network name and description**.
5. Under **Voting Policy**, choose the following:

   a. Enter the **Approval threshold percentage** along with the comparator, either **Greater than** or **Greater than or equal to**. For a proposal to pass, the Yes votes cast must meet this threshold before the vote duration expires.

b. Enter the **Proposal duration in hours**. If enough votes are not cast within this duration to either approve or reject a proposal, the proposal status is `EXPIRED`, no further votes on this proposal are allowed, and the proposal does not pass.

6. Choose **Next**, and then, under **Create member**, do the following to define the first member for the network, 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. Remember the user name and password. You need them later any time that you create users and resources that need to authenticate.

   c. Choose **Next**.

7. 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 takes around 30 minutes for Managed Blockchain to create your network, after which the **Status** is **Available**.

To create a Hyperledger Fabric network using the AWS CLI

Use the `create-network` command as shown in the following example. Consider the following:

- The example shows `HYPERLEDGER_FABRIC` as the `Framework` and as the `FrameworkVersion`. The `FrameworkConfiguration` properties for `--network-configuration` and `--member-configuration` options may be different for other frameworks and versions.
- The `AdminPassword` must be at least 8 characters long and no more than 32 characters. It must contain at least one uppercase letter, one lowercase letter, and one digit. It cannot have a single quote(‘), double quote(“), forward slash(/), backward slash(\), @, percent sign (%), or a space.
- The member name must not contain any special characters.
- Remember the user name and password. You need them later any time you create users and resources that need to authenticate.

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain create-network \
--cli-input-json '{\"Name\":\"OurBlockchainNet\", \"Description\":\"OurBlockchainNetDesc\",
 \"Framework\":\"HYPERLEDGER_FABRIC\",\"FrameworkVersion\": \"\", \"FrameworkConfiguration
\": {\"Fabric\": {\"Edition\": \"STARTER\"}}, \"VotingPolicy\": {\"ApprovalThresholdPolicy
\": {\"ThresholdPercentage\": 50, \"ProposalDurationInHours\": 24, \"ThresholdComparator
\": \"GREATER_THAN\"}}, “MemberConfiguration”: {\"Name\":\"org1\", \"Description\":\"Org1
 first member of network\", \"FrameworkConfiguration\":{\"Fabric\":\n{\"AdminUsername
\":\"MyAdminUser\",\"AdminPassword\":\"Password123\"}}, \"LogPublishingConfiguration\":
 {\"Fabric\":{\"CaLogs\":{\"Cloudwatch\": {\"Enabled\": true}}}}}}'
```

The command returns the Network ID and the Member ID, as shown in the following example:

```
{
    "NetworkId": "n-MWY63ZJZU5HGNCMBQER7IN6OIU",
    "MemberId": "m-K46ICRRXJRCGRNNS4ES4XUUS5A"
}
```

The **Networks** page on the console shows a **Status** of **Available** when the network is ready. Alternatively, you can use the `list-networks` command, as shown in the following example, to confirm the network status.

```
aws managedblockchain list-networks
```

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

```
{
    "Networks": [
        {
            "Id": "n-MWY63ZJZU5HGNCMBQER7IN6OIU",
            "Name": "MyTestNetwork",
            "Description": "MyNetDescription",
            "Framework": "HYPERLEDGER_FABRIC",
            "FrameworkVersion": "1.4",
            "Status": "AVAILABLE",
            "CreationDate": 1541497086.888,
        }
    ]
}
```

Step 2: Create and Configure the Interface VPC Endpoint

Now that the network is up and running in your VPC, you set up an interface VPC endpoint (AWS PrivateLink) for your member. 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**.

Step 3: Create a Peer Node in Your Membership

Now that your network and the first member are up and running, you can use the Managed Blockchain console or the AWS CLI 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.

Use one of the following procedures to create a peer node.

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 guidelines in Work with Hyperledger Fabric Peer Nodes on Managed Blockchain (p. 45), 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`, `--member-id`, and `AvailabilityZone` as appropriate.

```
[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"
}
```

Step 4: Create an Amazon EC2 Instance and Set Up the Hyperledger Fabric Client

To complete this step, you launch an Amazon EC2 instance using the Amazon Linux AMI. Consider the following requirements and recommendations when you create the Hyperledger Fabric client Amazon EC2 instance:

- We recommend that you launch the client Amazon EC2 instance in the same VPC and using the same security group as the VPC Endpoint that you created in Step 2: Create and Configure the Interface VPC Endpoint (p. 11). This simplifies connectivity between the Amazon EC2 instance and the Interface VPC Endpoint.
- We recommend that the EC2 security group shared by the VPC Endpoint and the client Amazon EC2 instance have rules that allow all inbound and outbound traffic between members of the security group. This also simplifies connectivity. In addition, ensure that this security group or another security group associated with the client 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 about security groups and required rules, see Configuring Security Groups for Hyperledger Fabric on Amazon Managed Blockchain (p. 111).
- Make sure that the client Amazon EC2 instance is configured with an automatically assigned public IP address and that you can connect to it using SSH. For more information, see Getting Started with Amazon EC2 Linux Instances and Connect to your Linux instance in the *Amazon EC2 User Guide for Linux Instances*.

- Make sure that the service role associated with the EC2 instance allows access to the Amazon S3 bucket where Managed Blockchain certificates are stored and that it has required permissions for working with Managed Blockchain resources. For more information, see Example IAM Role Permissions Policy for Hyperledger Fabric Client EC2 Instance (p. 106).

Note
An AWS CloudFormation template to create a Hyperledger Fabric client is available in amazon-managed-blockchain-client-templates repository on Github. For more information, see the readme.md in that repository. For more information about using AWS CloudFormation, see Getting Started in the *AWS CloudFormation User Guide*.

Step 4.1: 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 utilities, install docker, and configure the Docker user to be the default user for the Amazon EC2 instance:

```
sudo yum update -y
```

```
sudo yum install jq telnet emacs docker libtool libtool-ltdl-devel git -y
```

```
sudo service docker start
```

```
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.

```
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
```

```
sudo chmod a+x /usr/local/bin/docker-compose
```

Run the following commands to install golang.

```
wget https://dl.google.com/go/go1.14.4.linux-amd64.tar.gz
```

```
tar -xzf go1.14.4.linux-amd64.tar.gz
```

```
sudo mv go /usr/local
```

```
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

# CASERVICEENDPOINT is the endpoint to reach your member's CA
# for example ca.m-K46ICRRXJRCGRNNS4ES4XUUS5A.n-
MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.us-east-1.amazonaws.com:30002
export CASERVICEENDPOINT=MyMemberCaEndpoint

# ORDERER is the endpoint to reach your network's orderer
# for example orderer.n-MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.amazonaws.com:30001
export ORDERER=MyNetworkOrdererEndpoint

# 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:

```
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–1.14.x

To check the Docker version, run the following command:

```
sudo docker version
```

The command returns output similar to the following:

```
Client:
 Version: 18.06.1-ce
 API version: 1.38
 Go version: go1.14.4
 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: go1.14.4
 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:

```
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:

```
go version
```

The command returns output similar to the following:

```
go version go1.14.4 linux/amd64
```

Step 4.2: 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 the Interface VPC Endpoint (p. 11). 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).

```
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, the value of the variable `$CASERVICEENDPOINT` is the **CAEndpoint** returned by the `get-member` command.

```
curl https://$CASERVICEENDPOINT/cainfo -k
```

The command should return output similar to the following:

```
{"result":
{"CAName":"abcd1efghijkllmn5op3q52rst","CAChain":"LongStringOfCharacters","Version":"1.4.7-
snapshot-"}
,"errors":[],"messages":[],"success":true}
```

Alternatively, you can connect to the Fabric CA using Telnet as shown in the following example. Use the same endpoint in the `curl` example, but separate the endpoint and the port as shown in the following example.

```
telnet CaEndpoint-Without-Port CaPort
```

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 '^]'.
```

If you are unable to connect to the Fabric CA, double-check your network settings to ensure that the client Amazon EC2 instance has connectivity with the VPC Endpoint. In particular, ensure that the security groups associated with both the VPC Endpoint and the client Amazon EC2 instance have inbound and outbound rules that allow traffic between them.

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

Note
If you are working with Hyperledger Fabric v1.2 networks, you need to install and build the correct client version, which is available at https://github.com/hyperledger/fabric-ca/releases/download/v1.2.1/hyperledger-fabric-ca-linux-amd64-1.2.1.tar.gz.

```
mkdir -p /home/ec2-user/go/src/github.com/hyperledger/fabric-ca
```

```
cd /home/ec2-user/go/src/github.com/hyperledger/fabric-ca
```

```
wget https://github.com/hyperledger/fabric-ca/releases/download/v1.4.7/hyperledger-fabric-
ca-linux-amd64-1.4.7.tar.gz
```

```
tar -xzf hyperledger-fabric-ca-linux-amd64-1.4.7.tar.gz
```

Step 4.3: Clone the Samples Repository

Note
If you are working with Hyperledger Fabric v1.2 networks, use `--branch v1.2.0` instead of `--branch v1.4.7` in the following commmands.

```
cd /home/ec2-user
```

```
git clone --branch v1.4.7 https://github.com/hyperledger/fabric-samples.git
```

Step 4.4: 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 replace the *placeholder values* according to the following guidance:

- *MyMemberID* is the `MemberID` returned by the `aws managedblockchain list-members` AWS CLI command and shown on the member details page of the Managed Blockchain console—for example, `m-K46ICRRXJRCGRNNS4ES4XUUS5A`.
- *MyPeerNodeEndpoint* is the `PeerEndpoint` returned by the `aws managedblockchain get-node` command and listed on the node details page of the Managed Blockchain console —for example, nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y.m-K46ICRRXJRCGRNNS4ES4XUUS5A.n-MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.*us-east-1*.amazonaws.com:*30003*.

When you subsequently use the `cli` container to run commands—for example, `docker exec cli peer channel create`—you can use the `-e` option to override an environment variable that you establish in the `docker-compose-cli.yaml` file.

Note
If you are working with Hyperledger Fabric v1.2 networks, use `image: hyperledger/fabric-tools:1.2` in the following example instead of `image: hyperledger/fabric-tools:1.4`. In addition, use `CORE_LOGGING_LEVEL=info` instead of `FABRIC_LOGGING_SPEC=info`.

```
version: '2'
services:
  cli:
    container_name: cli
    image: hyperledger/fabric-tools:1.4
    tty: true
    environment:
      - GOPATH=/opt/gopath
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      - FABRIC_LOGGING_SPEC=info # Set logging level to debug for more verbose logging
      - CORE_PEER_ID=cli
      - CORE_CHAINCODE_KEEPALIVE=10
      - CORE_PEER_TLS_ENABLED=true
      - CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem
      - CORE_PEER_LOCALMSPID=MyMemberID
      - CORE_PEER_MSPCONFIGPATH=/opt/home/admin-msp
      - CORE_PEER_ADDRESS=MyPeerNodeEndpoint
    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:

```
docker-compose -f docker-compose-cli.yaml up -d
```

If you restarted or logged out and back in after the `usermod` command in Step 4.1: Install Packages (p. 13), 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:

```
sudo /usr/local/bin/docker-compose -f docker-compose-cli.yaml up -d
```

Step 5: Enroll an Administrative User

In this step, you use a pre-configured certificate to enroll a user with administrative permissions 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 user that you created in Step 1: Create the Network and First Member (p. 9).

Step 5.1: Create the Certificate File

Run the following command to copy the `managedblockchain-tls-chain.pem` to the `/home/ec2-user` directory. Replace *`MyRegion`* with the AWS Region you are using—for example, `us-east-1`.

```
aws s3 cp s3://MyRegion.managedblockchain/etc/managedblockchain-tls-chain.pem  /home/ec2-user/managedblockchain-tls-chain.pem
```

If the command fails with a permissions error, ensure that a service role associated with the EC2 instance allows access to the Amazon S3 bucket location. For more information see Example IAM Role Permissions Policy for Hyperledger Fabric Client EC2 Instance (p. 106).

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

```
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 5.2: Enroll the Administrative User

Managed Blockchain registers the user identity that you specified when you created the member as an administrator. In Hyperledger Fabric, this user is known as the *bootstrap identity* because the identity is used to enroll itself. To enroll, 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). For information about registering other user identities as administrators before you enroll them, see Register and Enroll a Hyperledger Fabric Admin (p. 60).

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).

```
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": "AVAILABLE",
        "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:

```
fabric-ca-client enroll \
-u 'https://AdminUsername:AdminPassword@$CASERVICEENDPOINT' \
--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:

```
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
```

Important
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 5.3: 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 `/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. The example assumes you are in the `/home/ec2-user` directory when running the command.

```
cp -r /home/ec2-user/admin-msp/signcerts admin-msp/admincerts
```

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 Multi-Member Channel (p. 24), you go through a similar process to set up a channel that includes another member.

Wait a minute or two for the administrative permissions from previous steps to propagate, and then perform these tasks to create a channel.

Note
All Hyperledger Fabric networks on Managed Blockchain support a maximum of 8 channels per network, regardless of network edition.

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.

This `configtx.yaml` enables application features associated with Hyperledger Fabric 1.4. It is not compatible with Hyperledger Fabric 1.2. For a `configtx.yaml` compatible with Hyperledger Fabric 1.2, see Work with Channels (p. 62).

Use a text editor to create a file with the following contents and save it as `configtx.yaml` on your Hyperledger Fabric client. Note the following placeholders and values.

- Replace *`MemberID`* with the MemberID you returned previously. For example *`m-K46ICRRXJRCGRNNS4ES4XUUS5A`*.
- The `MSPDir` is set to the same directory location, `/opt/home/admin-msp`, that you established using the `CORE_PEER_MSPCONFIGPATH` environment variable in the Docker container for the Hyperledger Fabric CLI in step 4.4 (p. 16).

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: MemberID
        # ID to load the MSP definition as
        ID: MemberID
        #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:
################################################################################
#
#   CAPABILITIES
#
#   This section defines the capabilities of fabric network. This is a new
#   concept as of v1.1.0 and should not be utilized in mixed networks with
#   v1.0.x peers and orderers.  Capabilities define features which must be
#   present in a fabric binary for that binary to safely participate in the
#   fabric network.  For instance, if a new MSP type is added, newer binaries
#   might recognize and validate the signatures from this type, while older
#   binaries without this support would be unable to validate those
#   transactions.  This could lead to different versions of the fabric binaries
#   having different world states.  Instead, defining a capability for a channel
#   informs those binaries without this capability that they must cease
#   processing transactions until they have been upgraded.  For v1.0.x if any
#   capabilities are defined (including a map with all capabilities turned off)
#   then the v1.0.x peer will deliberately crash.
#
################################################################################
Capabilities:
    # Channel capabilities apply to both the orderers and the peers and must be
    # supported by both.
    # Set the value of the capability to true to require it.
    # Note that setting a later Channel version capability to true will also
    # implicitly set prior Channel version capabilities to true. There is no need
    # to set each version capability to true (prior version capabilities remain
    # in this sample only to provide the list of valid values).
    Channel: &ChannelCapabilities
        # V1.4.3 for Channel is a catchall flag for behavior which has been
        # determined to be desired for all orderers and peers running at the v1.4.3
        # level, but which would be incompatible with orderers and peers from
        # prior releases.
        # Prior to enabling V1.4.3 channel capabilities, ensure that all
        # orderers and peers on a channel are at v1.4.3 or later.
        V1_4_3: true
        # V1.3 for Channel enables the new non-backwards compatible
        # features and fixes of fabric v1.3
        V1_3: false
        # V1.1 for Channel enables the new non-backwards compatible
        # features and fixes of fabric v1.1
        V1_1: false
    # Application capabilities apply only to the peer network, and may be safely
    # used with prior release orderers.
    # Set the value of the capability to true to require it.
    # Note that setting a later Application version capability to true will also
    # implicitly set prior Application version capabilities to true. There is no need
    # to set each version capability to true (prior version capabilities remain
    # in this sample only to provide the list of valid values).
    Application: &ApplicationCapabilities
        # V1.4.2 for Application enables the new non-backwards compatible
        # features and fixes of fabric v1.4.2
        V1_4_2: true
        # V1.3 for Application enables the new non-backwards compatible
        # features and fixes of fabric v1.3.
        V1_3: false
        # V1.2 for Application enables the new non-backwards compatible
        # features and fixes of fabric v1.2 (note, this need not be set if
        # later version capabilities are set)
        V1_2: false
        # V1.1 for Application enables the new non-backwards compatible
        # features and fixes of fabric v1.1 (note, this need not be set if
        # later version capabilities are set).
```

```
        V1_1: false
################################################################################
#
# 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:
    Capabilities:
        <<: *ApplicationCapabilities
################################################################################
#
# 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 command to generate the configtx peer block:

```
docker exec cli configtxgen \
-outputCreateChannelTx /opt/home/mychannel.pb \
-profile OneOrgChannel -channelID mychannel \
--configPath /opt/home/
```

Important
Hyperledger Fabric requires that a channel ID contain only lowercase ASCII alphanumeric characters, dots (.), and dashes (-). It must start with a letter, and must be fewer than 250 characters.

Step 6.2: Set Environment Variables for the Orderer

Set the `$ORDERER` environment variable for convenience. Replace *`orderer.n-MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.amazonaws.com:30001`* with the `OrderingServiceEndpoint` returned by the `aws managedblockchain get-network` command and listed on the network details page of the Managed Blockchain console.

```
export ORDERER=orderer.n-MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.amazonaws.com:30001
```

This variable must be exported each time you log out of the client. To persist the variable across sessions, add the export statement to your `~/.bash_profile` as shown in the following example.

```
# .bash_profile
...other configurations
export ORDERER=orderer.n-MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.amazonaws.com:30001
```

After updating `.bash_profile`, apply the changes:

```
source ~/.bash_profile
```

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 cli peer channel create -c mychannel \
-f /opt/home/mychannel.pb -o $ORDERER \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

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

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:

```
docker exec cli peer channel join -b mychannel.block \
-o $ORDERER --cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

Step 7: Install and Run Chaincode

In this section, you install sample chaincode on your peer. You then use the chaincode's `init` command to instantiate initial values attributed to entities `a` and `b` in the ledger, followed by the `query` command to confirm that instantiation was successful. Next, you use the chaincode's `invoke` command to transfer 10 units from `a` to `b` in the ledger. Finally, you use the chaincode's `query` command again to confirm that the value attributed to `a` was decremented by 10 units in the ledger.

Step 7.1: Install Chaincode

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

```
docker exec 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:

```
docker exec 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:

```
docker exec 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:

```
docker exec 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:

```
docker exec 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:

```
docker exec 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 Multi-Member Channel

Now that you have a Hyperledger Fabric network set up using Amazon Managed Blockchain, with an initial member in your AWS account and a VPC endpoint with a service name, you are ready to invite additional members. You invite additional members by creating a proposal for an invitation that existing members vote on. Since the blockchain network at this point consists of only one member, the first member always has the only vote on the invitation proposal for the second member. In the steps that follow, the network creator has an initial member named `org1` and the invited member is named `org2`. For proof of concept, you can create an invitation proposal for an additional member in the same AWS account that you used to create the network, or you can create an invitation proposal for a different AWS account.

After the invitation proposal is approved, the invited account can create a member. Invited members are free to reject the invitation or ignore it until the invitation proposal expires. The invited account needs the network ID and VPC endpoint service name of the blockchain network to create a member. For more information, see Work with Invitations (p. 39). The invited account also needs to fulfill the prerequisites listed in Prerequisites and Considerations (p. 6).

Step 8.1: Create an Invitation Proposal

Create a proposal to invite an AWS account to create a member and join the network according to the following procedures. You need the AWS account ID of the member you want to invite. You can also invite your own account to create an additional member. If you are using the CLI, you also need the Network ID and Member ID that you created in Step 1: Create the Network and First Member (p. 9).

To create an invitation proposal using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. From the navigation pane, choose **Networks**, and then choose the network to which you want to invite an AWS account.
3. Choose **Proposals** and then choose **Propose invitation**.
4. For **Submit proposal as**, choose the member in your account that submits the proposal.

   Note
   The member who submits the proposal must also vote on it. A Yes vote is not automatically assumed.
5. Enter an optional **Description** . The description appears to other members. It's a good way to communicate key points or a reminder about the proposal before they vote.
6. For each AWS account that you want to invite, enter the account number in the space provided. Choose **Add** to enter additional accounts.
7. Choose **Create**.

To create an invitation proposal using the AWS CLI

- Type a command similar to the following. Replace the value of `Principal` with the AWS account ID that you want to invite. Replace the value of `--member-id` with the value for the member in your account that submits the proposal.

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain create-proposal \
--actions Invitations=[{Principal=123456789012}] \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
--member-id m-K46ICRRXJRCGRNNS4ES4XUUS5A
```

  The command returns the proposal ID, as shown in the following example:

```
{
    "ProposalId": "p-ZR7KUD2YYNESLNG6RQ33X3FUFE"
}
```

Step 8.2: Vote Yes on the Proposal

After you create the invitation proposal, use the first member that you created to vote Yes and approve the proposal. You must do this within the duration defined by the network voting policy.

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.

2. From the navigation pane, choose **Networks**, and then choose the **Network** for which the proposal was made.
3. Choose **Proposals**.
4. Under **Active**, choose the **Proposal ID** to vote on.
5. Under **Vote on proposal**, select the member in your account to vote as. If your account has multiple members, each member gets a vote.
6. Choose **Yes** to vote to approve the proposal. Voting yes is a requirement for the second member to be created in the next step. Choosing **No** rejects the proposal and an invitation is not created.
7. Choose to **Confirm** your vote.

Step 8.3: Create the New Member

To accept an invitation to create a member and join 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. You first create the member as shown in the following procedures. If you use the AWS CLI, make sure that you have the relevant information, including the Network ID and the Invitation ID that the network sent to your account. When you create a member, you specify the name that identifies your member on the network. You also specify the admin user and password to authenticate to your member certificate authority (CA).

To accept an invitation to create a member and join a network using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. From the navigation pane, choose **Invitations**.
3. Select the invitation that you want to accept from the list, and then choose **Accept invitation**. To view more information about the network you are invited to join, choose the network **Name** from the list
4. Under **Create member and join network**, configure your network member according to the following guidelines:

   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. Remember the user name and password. You need them later any time that you create users and resources that need to authenticate.
5. Choose **Create member and join network**.

To accept an invitation to create a member and join a network using the AWS CLI

- Use the `create-member` command similar to the example below. Replace the value of `--network-id` with the Network ID that you are joining and `--invitation-id` with the Invitation ID sent to your account from the network.

```
aws managedblockchain create-member \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
--invitation-id i-XL9MDD6LVWWDNA9FF94Y4TFTE \
--member-configuration 'Name=org2,Description=MyMemberDesc,\
FrameworkConfiguration={Fabric={AdminUsername=MyAdminUsername,\
AdminPassword=Password123}}'
```

The command returns output similar to the following:

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

Additional Steps to Configure a Member

After you create the member, perform the following steps 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.

- Step 2: Create and Configure the Interface VPC Endpoint (p. 11)

  This step is only required if you are creating the second member in a different AWS account.
- Step 3: Create a Peer Node in Your Membership (p. 11)
- Step 4: Create an Amazon EC2 Instance and Set Up the Hyperledger Fabric Client (p. 12)

  If you are creating an additional member in the same AWS account, and 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 4.2: Set Up the Hyperledger Fabric CA Client (p. 15), using the new CA endpoint for the new member.
- Step 5: Enroll an Administrative User (p. 17)

Step 8.4: Share Artifacts and Information with the Network Creator

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 5: Enroll an Administrative User (p. 17). This is referenced in the following steps as `Org2AdminCerts`
- **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 `Org2CACerts`
- **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 3: Create a Peer Node in Your Membership (p. 11) is complete.

Step 8.5: 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:

```
mkdir /home/ec2-user/org2-msp
 mkdir /home/ec2-user/org2-msp/admincerts
 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:

```
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 8.6: Create configtx for the Multi-Member 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. 20), 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.

- Replace *Org1MemberID* with the MemberID of the first member that you created when you created the network (p. 9). For example, *m-K46ICRRXJRCGRNNS4ES4XUUS5A*.
- For `&Org1`, the `MSPDir` is set to the same directory location, `/opt/home/admin-msp`, that you established using the `CORE_PEER_MSPCONFIGPATH` environment variable in the Docker container for the Hyperledger Fabric CLI in step 4.4 (p. 16) above.

- Replace *Org2MemberID* with the MemberID of the second member that you created in step 8.4 (p. 26). For example, *m-J46DNSFRTVCCLONS9DT5TTLS2A*.
- For `&Org2`, the `MSPDir` is set to the same directory location, `/opt/home/org2-msp`, that you created and copied artifacts to in step 8.4 (p. 27).

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: Org1MemberID
        # ID to load the MSP definition as
        ID: Org1MemberID
        #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: Org2MemberID
        ID: Org2MemberID
        #msp dir of org2 in the docker container
        MSPDir: /opt/home/Org2AdminMSPDir
        AnchorPeers:
            - Host:
              Port:
################################################################################
#
#   CAPABILITIES
#
#   This section defines the capabilities of fabric network. This is a new
#   concept as of v1.1.0 and should not be utilized in mixed networks with
#   v1.0.x peers and orderers.  Capabilities define features which must be
#   present in a fabric binary for that binary to safely participate in the
#   fabric network.  For instance, if a new MSP type is added, newer binaries
#   might recognize and validate the signatures from this type, while older
#   binaries without this support would be unable to validate those
#   transactions.  This could lead to different versions of the fabric binaries
#   having different world states.  Instead, defining a capability for a channel
#   informs those binaries without this capability that they must cease
#   processing transactions until they have been upgraded.  For v1.0.x if any
#   capabilities are defined (including a map with all capabilities turned off)
#   then the v1.0.x peer will deliberately crash.
#
################################################################################
Capabilities:
    # Channel capabilities apply to both the orderers and the peers and must be
    # supported by both.
```

```
    # Set the value of the capability to true to require it.
    # Note that setting a later Channel version capability to true will also
    # implicitly set prior Channel version capabilities to true. There is no need
    # to set each version capability to true (prior version capabilities remain
    # in this sample only to provide the list of valid values).
    Channel: &ChannelCapabilities
        # V1.4.3 for Channel is a catchall flag for behavior which has been
        # determined to be desired for all orderers and peers running at the v1.4.3
        # level, but which would be incompatible with orderers and peers from
        # prior releases.
        # Prior to enabling V1.4.3 channel capabilities, ensure that all
        # orderers and peers on a channel are at v1.4.3 or later.
        V1_4_3: true
        # V1.3 for Channel enables the new non-backwards compatible
        # features and fixes of fabric v1.3
        V1_3: false
        # V1.1 for Channel enables the new non-backwards compatible
        # features and fixes of fabric v1.1
        V1_1: false
    # Application capabilities apply only to the peer network, and may be safely
    # used with prior release orderers.
    # Set the value of the capability to true to require it.
    # Note that setting a later Application version capability to true will also
    # implicitly set prior Application version capabilities to true. There is no need
    # to set each version capability to true (prior version capabilities remain
    # in this sample only to provide the list of valid values).
    Application: &ApplicationCapabilities
        # V1.4.2 for Application enables the new non-backwards compatible
        # features and fixes of fabric v1.4.2
        V1_4_2: true
        # V1.3 for Application enables the new non-backwards compatible
        # features and fixes of fabric v1.3.
        V1_3: false
        # V1.2 for Application enables the new non-backwards compatible
        # features and fixes of fabric v1.2 (note, this need not be set if
        # later version capabilities are set)
        V1_2: false
        # V1.1 for Application enables the new non-backwards compatible
        # features and fixes of fabric v1.1 (note, this need not be set if
        # later version capabilities are set).
        V1_1: false
################################################################################
#
# 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:
    Capabilities:
        <<: *ApplicationCapabilities
################################################################################
#
# 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:

```
docker exec cli configtxgen \
-outputCreateChannelTx /opt/home/ourchannel.pb \
-profile TwoOrgChannel -channelID ourchannel \
--configPath /opt/home/
```

Step 8.7: Create the Channel

The channel creator (org1) uses the following command on their Hyperledger Fabric client to submit the channel to the orderer, which creates the channel `ourchannel`. The command example assumes that Docker environment variables have been configured as described in Step 4.4: Configure and Run Docker Compose to Start the Hyperledger Fabric CLI (p. 16) and that the `$ORDERER` environment variable has been set on the client.

```
docker exec cli peer channel create -c ourchannel \
-f /opt/home/ourchannel.pb -o $ORDERER \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

Step 8.8: Get Channel Genesis Block

Both org1 and org2 need to run the following command on their respective Hyperledger Fabric clients to join their peer nodes to the channel. For more information about the `peer channel` command, see peer channel in Hyperledger Fabric documentation.

```
docker exec cli peer channel fetch oldest /opt/home/ourchannel.block \
-c ourchannel -o $ORDERER \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

Step 8.9: 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. For more information about the `peer channel` command, see peer channel in Hyperledger Fabric documentation.

```
docker exec cli peer channel join -b /opt/home/ourchannel.block \
-o $ORDERER --cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

Optionally, after you join a peer to a channel, you can set up the peer node as an *anchor peer*. Anchor peers support the gossip protocol, which is required for some features of Hyperledger Fabric, such as private data collections and service discovery. For more information, see Add an Anchor Peer to a Channel (p. 67).

Step 8.10: 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:

```
docker exec cli peer chaincode install -n myjointcc -v v0 \
-p github.com/chaincode_example02/go
```

Step 8.11: 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.

```
docker exec cli peer chaincode instantiate -o $ORDERER \
-C ourchannel -n myjointcc -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:

```
docker exec cli peer chaincode query -C ourchannel \
-n myjointcc -c '{"Args":["query","a"]}'
```

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

Step 8.12: 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. 24). 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 *Org1PeerEndpoint*.

```
docker exec cli peer chaincode invoke \
-C ourchannel -n myjointcc -c '{"Args":["invoke","a","b","10"]}' \
--peerAddresses Org1PeerEndpoint \
--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:

```
docker exec cli peer chaincode query -C ourchannel \
-n myjointcc -c '{"Args":["query","a"]}'
```

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

Create a Hyperledger Fabric Blockchain Network on Amazon Managed Blockchain

When a user in an AWS account creates a Hyperledger Fabric 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 use that member to create an invitation proposal for other members in the same AWS account or in other AWS accounts. Any member can create an invitation proposal.

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 for Hyperledger Fabric on Amazon Managed Blockchain (p. 43).
- 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 Work with Hyperledger Fabric Peer Nodes on Managed Blockchain (p. 45).
- Create an invitation proposal for other AWS accounts to be members of the network, or invite an additional member in your account to simulate a multi-AWS account network. Vote Yes on your own proposal to approve it and create the invitation. For more information about inviting members, see Create an Invitation Proposal (p. 56).

Create a Hyperledger Fabric Network

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

To create a Hyperledger Fabric network using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. Choose **Create private network**.
3. Under **Blockchain frameworks**:
   a. Select the blockchain framework to use. This tutorial is based on **Hyperledger Fabric version** .
   b. Select the **Network edition** to use. The network edition determines attributes of the network, such as the maximum number of members, nodes per member, and transaction throughput. Different editions have different rates associated with the membership. For more information, see Amazon Managed Blockchain Pricing.
4. Enter a **Network name and description**.
5. Under **Voting Policy**, choose the following:

a. Enter the **Approval threshold percentage** along with the comparator, either **Greater than** or **Greater than or equal to**. For a proposal to pass, the Yes votes cast must meet this threshold before the vote duration expires.

b. Enter the **Proposal duration in hours**. If enough votes are not cast within this duration to either approve or reject a proposal, the proposal status is `EXPIRED`, no further votes on this proposal are allowed, and the proposal does not pass.

6. Choose **Next**, and then, under **Create member**, do the following to define the first member for the network, 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. Remember the user name and password. You need them later any time that you create users and resources that need to authenticate.

   c. Choose **Next**.

7. 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 takes around 30 minutes for Managed Blockchain to create your network, after which the **Status** is **Available**.

To create a Hyperledger Fabric network using the AWS CLI

Use the `create-network` command as shown in the following example. Consider the following:

- The example shows `HYPERLEDGER_FABRIC` as the `Framework` and as the `FrameworkVersion`. The `FrameworkConfiguration` properties for `--network-configuration` and `--member-configuration` options may be different for other frameworks and versions.
- The `AdminPassword` must be at least 8 characters long and no more than 32 characters. It must contain at least one uppercase letter, one lowercase letter, and one digit. It cannot have a single quote(‘), double quote(“), forward slash(/), backward slash(\), @, percent sign (%), or a space.
- The member name must not contain any special characters.
- Remember the user name and password. You need them later any time you create users and resources that need to authenticate.

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain create-network \
--cli-input-json '{\"Name\":\"OurBlockchainNet\", \"Description\":\"OurBlockchainNetDesc\",
 \"Framework\":\"HYPERLEDGER_FABRIC\",\"FrameworkVersion\": \"\", \"FrameworkConfiguration
\": {\"Fabric\": {\"Edition\": \"STARTER\"}}, \"VotingPolicy\": {\"ApprovalThresholdPolicy
\": {\"ThresholdPercentage\": 50, \"ProposalDurationInHours\": 24, \"ThresholdComparator
\": \"GREATER_THAN\"}}, “MemberConfiguration”: {\"Name\":\"org1\", \"Description\":\"Org1
 first member of network\", \"FrameworkConfiguration\":{\"Fabric\":\n{\"AdminUsername
\":\"MyAdminUser\",\"AdminPassword\":\"Password123\"}}, \"LogPublishingConfiguration\":
 {\"Fabric\":{\"CaLogs\":{\"Cloudwatch\": {\"Enabled\": true}}}}}}'
```

The command returns the Network ID and the Member ID, as shown in the following example:

```
{
    "NetworkId": "n-MWY63ZJZU5HGNCMBQER7IN6OIU",
    "MemberId": "m-K46ICRRXJRCGRNNS4ES4XUUS5A"
}
```

Delete a Hyperledger Fabric Network on Amazon Managed Blockchain

A Hyperledger Fabric network on Amazon Managed Blockchain remains active as long as there are members. A network is deleted only when the last member deletes itself from the network. No member or AWS account, even the creator's AWS account, can delete the network until they are the last member and delete themselves. When you delete the last member, all resources for that member and the blockchain network are deleted. For more information, see Delete a Member in Your AWS Account (p. 38).

Invite or Remove Hyperledger Fabric Network Members on Amazon Managed Blockchain

To invite and remove other Hyperledger Fabric network members, any member can create a *proposal* that is submitted for a vote to all network members. If a proposal is approved within the duration and with the percentage of Yes votes specified in the voting policy for the network, the appropriate action is carried out.

A member can only join the network through an approved invitation proposal. The exception is the first member, which is created along with the network. The first member then submits a proposal and is the sole voter on the proposal to invite the second member.

> **Note**
> Each member that is created accrues charges according to the membership rate for the network. For more information, see Amazon Managed Blockchain Pricing.

An AWS account can delete members from the network that they own directly. A proposal is not required. To delete a member in a different AWS account, a proposal to remove the member is required. Information about all proposals, including the member who created the proposal, the current vote count, and more is available to all network members.

This topic provides basic information for creating proposals to invite or remove members, and to delete a member that your AWS account owns. For more detailed information about proposals, including how to vote on a proposal, see Work with Proposals for a Hyperledger Fabric Network on Amazon Managed Blockchain (p. 51).

Create a Proposal to Invite an AWS Account to the Network

You can use the AWS Management Console, the AWS CLI, or the Managed Blockchain API to create an invitation proposal. When a proposal to invite a member is approved, an invitation is sent to the specified AWS accounts. An administrator with the appropriate permissions in that account can then choose to either create a member and join the network or reject the invitation.

To create an invitation proposal using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. From the navigation pane, choose **Networks**, and then choose the network to which you want to invite an AWS account.
3. Choose **Proposals** and then choose **Propose invitation**.

4. For **Submit proposal as**, choose the member in your account that submits the proposal.

   Note
   The member who submits the proposal must also vote on it. A Yes vote is not automatically assumed.

5. Enter an optional **Description** . The description appears to other members. It's a good way to communicate key points or a reminder about the proposal before they vote.
6. For each AWS account that you want to invite, enter the account number in the space provided. Choose **Add** to enter additional accounts.
7. Choose **Create**.

To create an invitation proposal using the AWS CLI

- Type a command similar to the following. Replace the value of `Principal` with the AWS account ID that you want to invite. Replace the value of `--member-id` with the value for the member in your account that submits the proposal.

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain create-proposal \
--actions Invitations=[{Principal=123456789012}] \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
--member-id m-K46ICRRXJRCGRNNS4ES4XUUS5A
```

The command returns the proposal ID, as shown in the following example:

```
{
    "ProposalId": "p-ZR7KUD2YYNESLNG6RQ33X3FUFE"
}
```

Create a Proposal to Remove a Member From the Network

You can use the AWS Management Console, the AWS CLI, or the Managed Blockchain API to create a proposal to remove a member owned by another AWS account.

To create a proposal to remove a member using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. From the navigation pane, choose **Networks**, and then choose the network.
3. Choose **Proposals** and then choose **Propose removal**.
4. For **Submit proposal as**, choose the member in your account that submits the proposal.

   Note
   The member who submits the proposal must also vote on it. A Yes vote is not automatically assumed.

5. Enter an optional **Description** . The description appears to other members. It's a good way to communicate key points or a reminder about the proposal before they vote.
6. For each member that you want to remove, enter the member ID in the space provided. Choose **Add** to enter additional members.

To create a removal proposal using the AWS CLI

- Type a command similar to the following. Replace the value of `Principal` with the AWS account ID that you want to invite. Replace the value of `--member-id` with the value for the member in your account that submits the proposal.

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain create-proposal \
--actions Removals=[{MemberID=m-K46ICRRXJRCGRNNS4ES4XUUS5A}] \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
--member-id m-J46DNSFRTVCCLONS9DT5TTLS2A
```

The command returns the proposal ID, as shown in the following example:

```
{
    "ProposalId": "p-ZR7KUD2YYNESLNG6RQ33X3FUFE"
}
```

Delete a Member in Your AWS Account

You can use the AWS Management Console, the AWS CLI, or the Managed Blockchain API to directly remove members that your AWS account owns from a network.

Warning
Removing a member in your account deletes all associated resources, such as peer nodes. For your AWS account to rejoin the network, an existing member must create a proposal to invite your AWS account, and the proposal must be approved.

To delete a member in your account using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. Choose **Networks**, choose the network **Name**, and then choose **Members**.
3. Under **Members owned by you**, select a member and then choose **Delete member**.
4. Choose **Delete** when prompted to confirm.

To delete a member in your account using the AWS CLI

- Use the `delete-member` command as shown in the following example. Replace the values of `--network-id` and `--member-id` as appropriate.

```
aws managedblockchain delete-member --network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU --member-id m-J46DNSFRTVCCLONS9DT5TTLS2A
```

Accept an Invitation to Create a Member and Join a Hyperledger Fabric Network on Amazon Managed Blockchain

In Amazon Managed Blockchain, a member is a distinct identity within the Hyperledger Fabric network associated with an AWS account. An AWS account can have multiple members on the network. Every member in a network must be invited to participate through a proposal made by an existing member and approved according to the network's voting policy. The exception is the first member, which is created along with the network. For more information, see Work with Proposals for a Hyperledger Fabric Network on Amazon Managed Blockchain (p. 51). After the invitation is approved, the invited AWS account can create a member and join the network using the invitation ID.

Each member pays an hourly rate, 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. Charges may vary depending on the network edition selected when the network was created. 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, but each member must have the following resources:

- **An interface VPC endpoint in the account**–Managed Blockchain is a PrivateLink-powered service, so you must have an interface VPC endpoint in your account to communicate with the service endpoint that the Managed Blockchain network makes available. For more information, see Create an Interface VPC Endpoint for Hyperledger Fabric on Amazon Managed Blockchain (p. 43) and Key Concepts: Amazon Managed Blockchain Networks, Members, and Peer Nodes (p. 2).
- **One or more peer nodes**–Each member 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. Peer nodes run on Managed Blockchain instances. Custom Amazon EC2 instances or on-premises instances cannot participate as peer nodes on a Hyperledger Fabric network on Managed Blockchain. For more information, see Work with Hyperledger Fabric Peer Nodes on Managed Blockchain (p. 45).

Topics

Work with Invitations

If you are invited to join a Hyperledger Fabric network on Amazon Managed Blockchain, you can accept the invitation by creating a member using the invitation ID. You can also reject the invitation. After you reject an invitation, the invitation ID is no longer valid. A new invitation proposal must be approved, and

a new invitation ID is required to create a member. If don't accept or reject an invitation before it expires, the invitation lapses. As with a rejected invitation, a new invitation ID is required.

You can see all pending, accepted, and rejected invitations for your AWS account in the AWS Management Console. Alternatively, you can use the AWS CLI or the Managed Blockchain SDK ListInvitations action.

You can set up Amazon CloudWatch Events along with Amazon Simple Notification Service so that you receive an alert when there is an invitation for your account. For more information, see Automating Managed Blockchain Proposal Notifications with CloudWatch Events (p. 58).

To list blockchain network member invitations for your AWS account using the console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. Choose **Invitations**, and then do one of the following:

To...	Do this...
View details about the network, such as the network ID, the description, endpoints, voting policy details, and current members.	Select the invitation from the list and choose **View details**.
Use the invitation to create a member and join the network.	Select the invitation from the list and choose **Accept Invitation**. For next steps, see Create a Member and Join a Network (p. 41)
Reject the invitation.	Select the invitation from the list and choose **Reject Invitation**.

To list blockchain network member invitations for your AWS account using the AWS CLI

- Use the following command:

```
aws managedblockchain list-invitations
```

The command returns a list of invitations, along with detail for each invitation, as shown in the following example for an invitation in the `PENDING` status:

```
{
   "Invitations": [
      {
         "CreationDate": 2019-04-08T23:40:20.628Z,
         "ExpirationDate": 2019-04-09T23:40:20.628Z,
         "InvitationId": "i-XL9MDD6LVWWDNA9FF94Y4TFTE",
         "NetworkSummary": {
            "CreationDate": 2019-04-03T13:15:22.345Z,
            "Description": "Test network for supply chain blockchain.",
            "Framework": "HYPERLEDGER_FABRIC",
            "FrameworkVersion": "1.4.7",
            "Id": "n-MWY63ZJZU5HGNCMBQER7IN6OIU",
            "Name": "Example Corp.",
            "Status": "AVAILABLE"
         },
```

```
            "Status": "PENDING"
        }
    ]
}
```

You can use the `InvitationID` with the `create-member` command to create a member and join the network. For next steps, see Create a Member and Join a Network (p. 41).

Create a Member and Join a Network

You can use the Managed Blockchain console, the AWS CLI, or the Managed Blockchain SDK CreateMember action to create a member in a network that your account is invited to. If you created the network, you create the first member when you create the network. All subsequent members must be invited to join by way of a member proposal.

After you create the member, for the member to be functional on the network, your account must have a VPC endpoint associated with the VPC endpoint service name published by the network. For more information, see Create an Interface VPC Endpoint for Hyperledger Fabric on Amazon Managed Blockchain (p. 43). You also must create at least one peer node in your membership. For more information, see Work with Hyperledger Fabric Peer Nodes on Managed Blockchain (p. 45).

To accept an invitation to create a member and join a network using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. From the navigation pane, choose **Invitations**.
3. Select the invitation that you want to accept from the list, and then choose **Accept invitation**. To view more information about the network you are invited to join, choose the network **Name** from the list
4. Under **Create member and join network**, configure your network member according to the following guidelines:

   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. Remember the user name and password. You need them later any time that you create users and resources that need to authenticate.
5. Choose **Create member and join network**.

To accept an invitation to create a member and join a network using the AWS CLI

- Use the `create-member` command similar to the example below. Replace the value of `--network-id` with the Network ID that you are joining and `--invitation-id` with the Invitation ID sent to your account from the network.

```
aws managedblockchain create-member \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
--invitation-id i-XL9MDD6LVWWDNA9FF94Y4TFTE \
--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.

Create an Interface VPC Endpoint for Hyperledger Fabric on Amazon Managed Blockchain

Each member of a Hyperledger Fabric 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 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 Hyperledger Fabric network on Managed Blockchain 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 Hyperledger Fabric 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 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 a 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 network.

Work with Hyperledger Fabric Peer Nodes on Managed Blockchain

Hyperledger Fabric peer nodes do the work for your member on the network. 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. A new member has no peer nodes. Create at least one peer node per member.

Each peer node runs on a *Managed Blockchain instance type*. You cannot add a custom Amazon EC2 instance to your member, nor can you connect an on-premises machine. The number of peer nodes and the Managed Blockchain instance type of peer nodes available to each member depends on the network edition specified when the network was created. For more information, see Amazon Managed Blockchain Pricing.

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

- **Managed Blockchain instance type**

  This determines the computational and memory capacity allocated to this node for the blockchain workload. You can 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 types are subject to different pricing. You can monitor CPU and memory utilization to determine if your Managed Blockchain instance type is appropriate. For more information, see Use Hyperledger Fabric Peer Node Metrics on Amazon Managed Blockchain (p. 48)
- **Availability Zone**

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

- **State DB configuration**

  This selection is only available for peer nodes running on networks using Hyperledger Fabric 1.4 and later. This determines the type of state database that the peer node uses. **LevelDB** stores chaincode data as simple key-value pairs. It supports only key, key range, and composite key queries. **CouchDB** models data in the ledger as JSON. It supports richer queries and indexing for efficiency. CouchDB is the default for networks running Hyperledger Fabric 1.4 and later. Hyperledger Fabric 1.2 uses only LevelDB. For more information, see Query Chaincode Data in the State Database (p. 81) and CouchDB as the State Database in Hyperledger Fabric documentation.
- **Logging configuration**

  You can enable peer node logs, chaincode logs, or both for a member. Peer node logs allow you to debug timeout errors associated with proposals and identify rejected proposals. Chaincode logs help you analyze and debug business logic. For more information, see Monitoring Hyperledger Fabric on Managed Blockchain Using CloudWatch Logs (p. 117).

Create a Hyperledger Fabric Peer Node on Amazon Managed Blockchain

You can create a Hyperledger Fabric 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 guidelines in Work with Hyperledger Fabric Peer Nodes on Managed Blockchain (p. 45), 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`, `--member-id`, and `AvailabilityZone` as appropriate.

```
[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"
}
```

View Hyperledger Fabric Peer Node Properties on Amazon Managed Blockchain

You can view information about each Hyperledger Fabric peer node that belong to your member using the AWS Management Console, the AWS CLI or the Managed Blockchain API GetNode command. Details include basic information like the Managed Blockchain instance type, Availability Zone, and creation date, along with the following important properties:

- Status
  - **Creating**

    Managed Blockchain is provisioning and configuring the Managed Blockchain instance for the peer node.
  - **Available**

    The peer node is running and available on the network.
  - **Failed**

    The peer node has an issue that has caused Managed Blockchain to add it to the deny list on the network. This usually indicates that the peer node has reached memory or storage capacity. As a first step, we recommend that you delete the instance and provision an instance with more capacity.
  - **Create Failed**

    The node could not be created with the Managed Blockchain instance type and the Availability Zone specified. We recommend trying another availability zone, a different instance type, or both.

- **Deleting**

  The node is being deleted. This can happen because the node was deleted by the member, the member was deleted by the AWS account, or the member was deleted through an approved removal proposal.
- **Deleted**

  The node has been deleted. See the previous item for possible reasons.

- Endpoints

  Hyperledger Fabric uses endpoints associated with each peer node to identify the peer node on the network for different processes. Managed Blockchain assigns unique peer node endpoints to each peer node on each network when the peer node is created. The peer node endpoint consists of the applicable port and the domain name of the peer node derived from the network ID, member ID, and peer node ID. For more information, see Identifying Managed Blockchain Resources and Connecting from a Client (p. 4). Do not assume that the ports for a service are the same among members; different members may use different ports for the same service. Conversely, peer nodes in different networks may use the same ports, but their endpoints are always unique.
  - **Peer endpoint**

    Use this endpoint, including the port, within Hyperledger Fabric to address the peer node when using all services other than peer channel-based event services.
  - **Peer event endpoint**

    Use this endpoint, including the port, within Hyperledger Fabric to address the peer node for peer channel-based event services.

You can also view and monitor **Metrics** related to peer node performance. For more information, see Use Hyperledger Fabric Peer Node Metrics on Amazon Managed Blockchain (p. 48).

You can check the peer node status using the `get-node` command, as shown in the following example:

```
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 `PeerEndpoint` and `PeerEventEndpoint`, as shown in the following example. You need this endpoint and port when communicating with the node using your blockchain framework client or addressing the node within an application.

```
{
   "Node": {
      "AvailabilityZone": "us-east-1a",
      "CreationDate": 2019-04-08T23:40:20.628Z,
      "FrameworkAttributes": {
         "Fabric": {
            "PeerEndpoint": "nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y.m-K46ICRRXJRCGRNNS4ES4XUUS5A.n-
MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.us-east-1.amazonaws.com:30003",
            "PeerEventEndpoint": "nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y.m-
K46ICRRXJRCGRNNS4ES4XUUS5A.n-MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.us-
east-1.amazonaws.com:30004"
         }
      },
      "Id": "nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y",
      "InstanceType": "bc.t3.small",
      "LogPublishingConfiguration": {
         "Fabric": {
```

```
            "ChaincodeLogs": {
               "Cloudwatch": {
                  "Enabled": true
               }
            },
            "PeerLogs": {
               "Cloudwatch": {
                  "Enabled": true
               }
            }
         }
      },
      "StateDB": "CouchDB"
      "Status": "AVAILABLE"
   }
}
```

Use Hyperledger Fabric Peer Node Metrics on Amazon Managed Blockchain

You can use peer node metrics to track the activity and health of Hyperledger Fabric peer nodes on Amazon Managed Blockchain that belong to your member. You can use the Managed Blockchain console to view the metrics for a peer node. Managed Blockchain also reports metrics to Amazon CloudWatch. You can use CloudWatch to set up dashboards, receive alarms, and view log files for peer node metrics. For more information, see Using Amazon CloudWatch Metrics in the *Amazon CloudWatch User Guide*.

In addition to using peer node metrics, you optionally can enable CloudWatch Logs for peer nodes and for instances of chaincode running on a peer node to view **Peer node logs** and **Chaincode logs**. These logs are useful for troubleshooting and analysis of chaincode activity. For more information, see Monitoring Hyperledger Fabric on Managed Blockchain Using CloudWatch Logs (p. 117).

Managed Blockchain collects the following metrics for each peer node in the `aws/managedblockchain` namespace. Available metrics in Managed Blockchain correspond to Hyperledger Fabric metrics.

Metric name	Description
Fabric metrics	
ChaincodeExecuteTimeouts	The number of chaincode executions (`Init` or `Invoke`) that have timed out. Units: Count
EndorserProposalDuration	For each proposal, the time to complete the proposal. Units: Seconds
EndorserProposalValidationFailures	The number of proposals that have failed initial validation. Units: Count
EndorserProposalsReceived	The number of proposals received. Units: Count

Metric name	Description
EndorserSuccessfulProposals	The number of successful proposals. Units: Count
Transactions	The number of transactions that a peer node receives per minute. Units: Count
Utilization metrics	
CPUUtilization	The percentage of total CPU capacity used on the peer node's Managed Blockchain instance at any given instant. Units: Percent
MemoryUtilization	The percentage of total available memory used on the peer node's Managed Blockchain instance at any given instant. Units: Percent

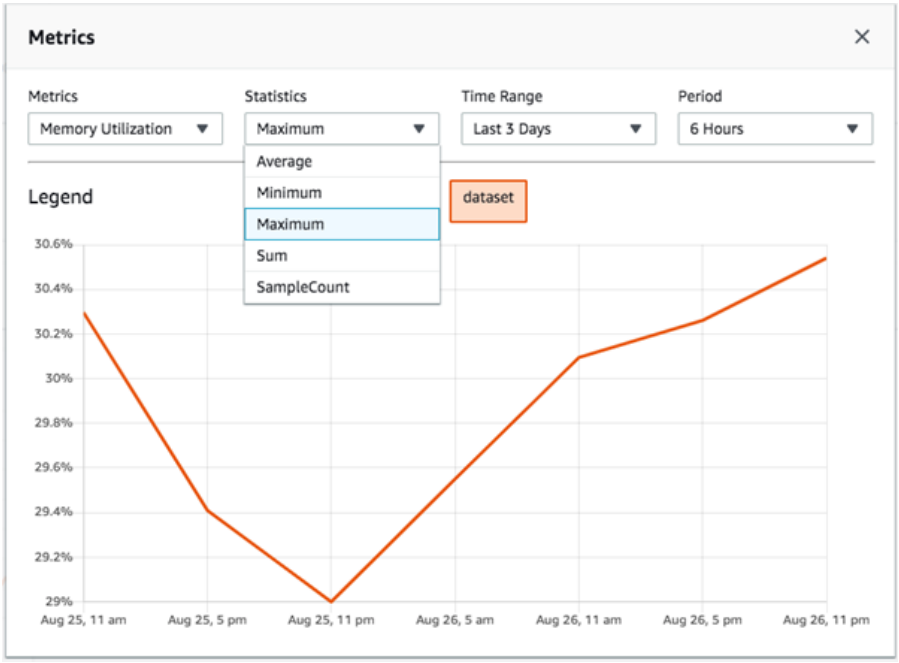
Viewing Peer Node Metrics

You can use the Amazon Managed Blockchain console to view graphs for peer node metrics. Metrics are available on the peer node details page.

To view metrics using the Managed Blockchain console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. Under **Network**, choose the **Name** of the network.
3. Choose **Members**. Under **Members owned by you**, choose the **Name** of the member to which the node belongs.
4. Under **Peer Nodes**, choose the **Node ID** you want to view.

   Under **Metrics**, tabs for **Channel Metrics** and **Utilization Metrics** are available.
5. For **Channel Metrics**, choose the channels you want to view or compare from the list.
6. Choose a chart and then use **Statistics**, **Time Range**, and **Period** to customize the chart.


Metrics
Metrics
Memory Utilization
Statistics
Maximum
Time Range
Last 3 Days
Period
6 Hours
Average
Minimum
Maximum
Sum
SampleCount
Legend
dataset
30.6%
30.4%
30.2%
30%
29.8%
29.6%
29.4%
29.2%
29%
Aug 25, 11 am
Aug 25, 5 pm
Aug 25, 11 pm
Aug 26, 5 am
Aug 26, 11 am
Aug 26, 5 pm
Aug 26, 11 pm

Work with Proposals for a Hyperledger Fabric Network on Amazon Managed Blockchain

To make a change to a Hyperledger Fabric network on Amazon Managed Blockchain that requires consensus among network members, network members create *proposals*. For example, a member can create a proposal to invite another AWS account to become a member, to invite multiple accounts, or to remove one or more members in different AWS accounts.

A proposal is submitted to all network members to cast a Yes or No vote. If the proposal is approved within the duration and with the percentage of Yes votes specified in the voting policy for the network, the proposed action is carried out. The *voting policy* is established when the network is created and governs votes on all proposals. It can't be updated after the network is created. For more information, see Create a Hyperledger Fabric Blockchain Network on Amazon Managed Blockchain (p. 33).

Understanding the Proposal Lifecycle

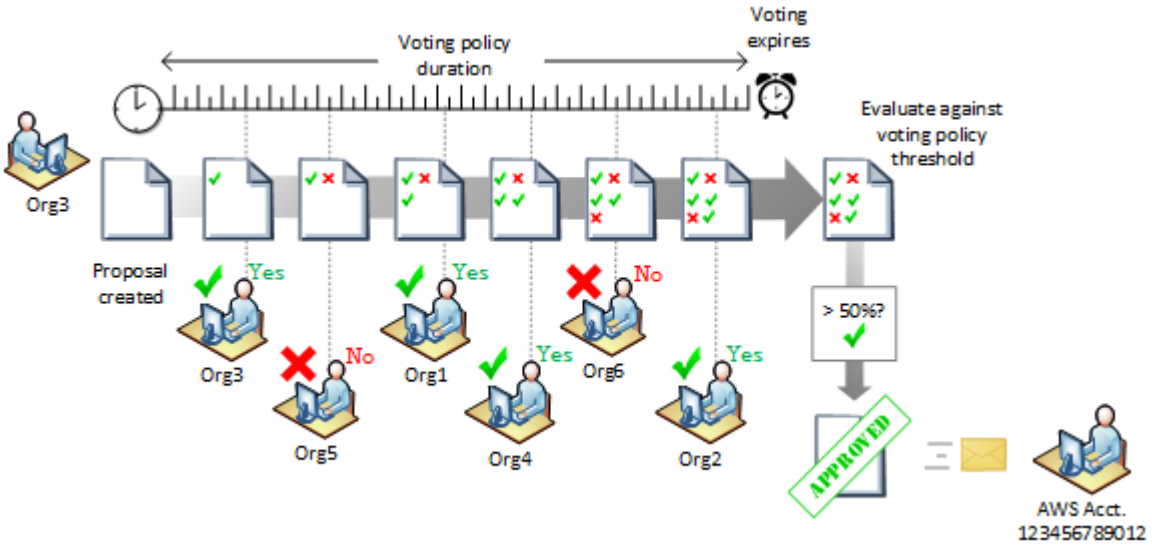
To understand the proposal lifecycle, consider a hypothetical proposal to invite AWS account 123456789012 to join a Hyperledger Fabric network on Managed Blockchain made by a member named Org3. The network currently has six members: Org1, Org2, Org3, and so on. The network was created by Org1, who specified a voting policy with a **50% approval threshold**, a **greater than** comparator, and a proposal duration of 24 hours.

The following flow diagrams depict the possible outcomes of a proposal using this example:

- Approved with Full Vote (p. 51)
- Approved with Partial Vote (p. 52)
- Rejected with Full Vote (p. 52)
- Rejected with Partial Vote (p. 53)
- Expired, Does Not Pass (p. 53)

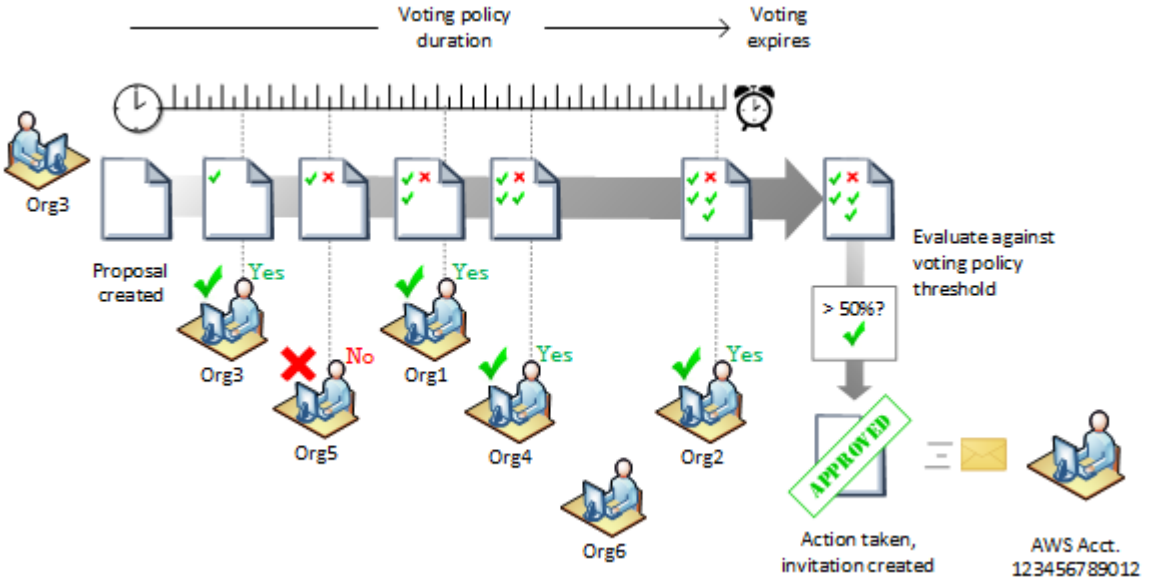
Example – Proposal approved with full member vote

For the following proposal, all members cast a vote before the duration expired. The proposal is `APPROVED`, and an invitation is extended to the AWS account.



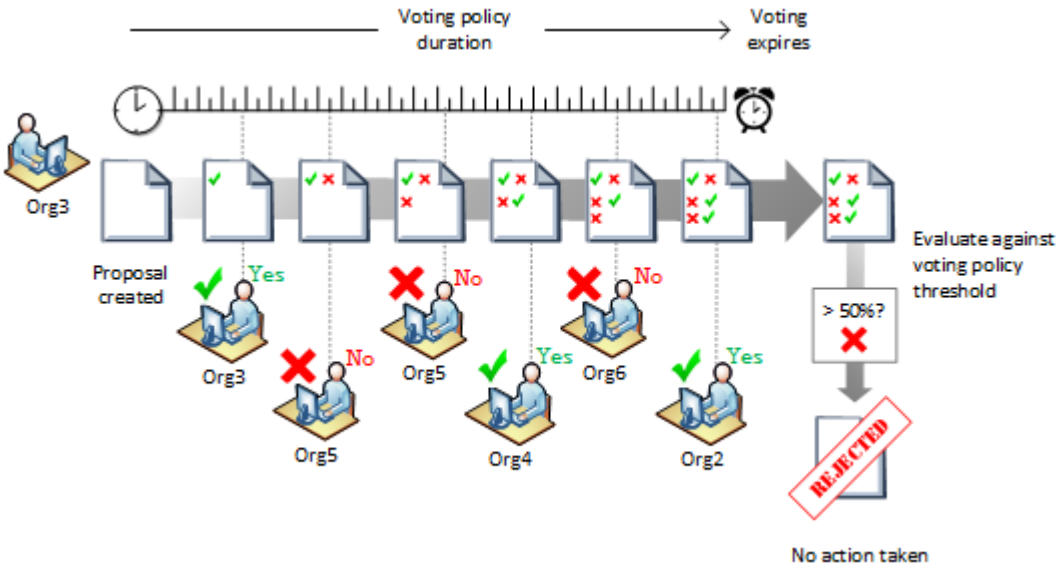
Example – Proposal approved with partial member vote

For the following proposal, not all members cast a vote before the duration expired. However, enough Yes votes were cast to approve the proposal according to the voting policy. The proposal is `APPROVED`, and an invitation is extended to the AWS account.



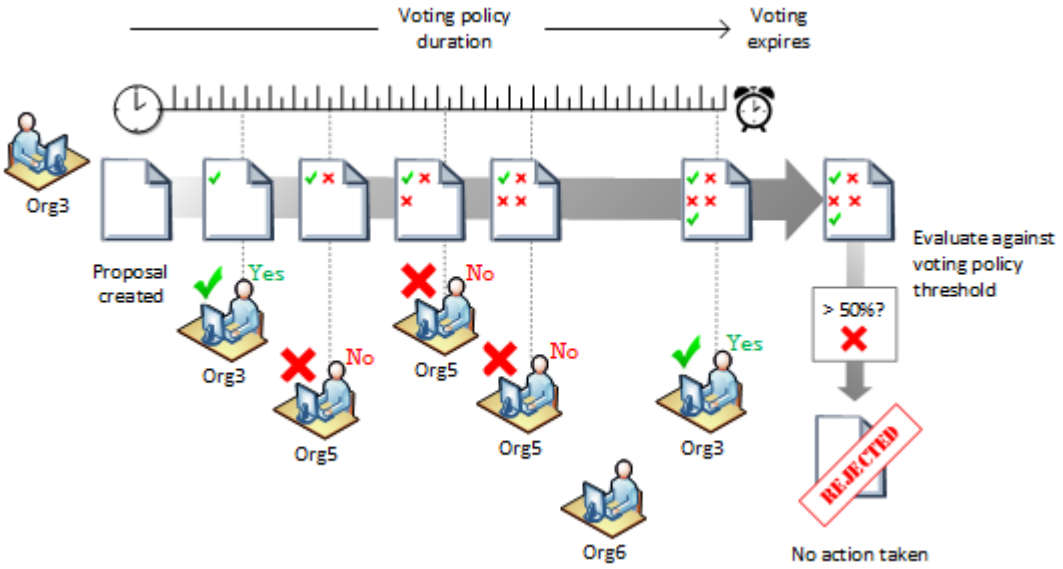
Example – Proposal rejected with full member vote

For the following proposal, all members cast a vote before the duration expired. Because the comparator in the voting policy is **greater than**, a three-to-three vote does not pass the threshold for approval. The proposal is `REJECTED`, and an invitation is not extended to the AWS account.



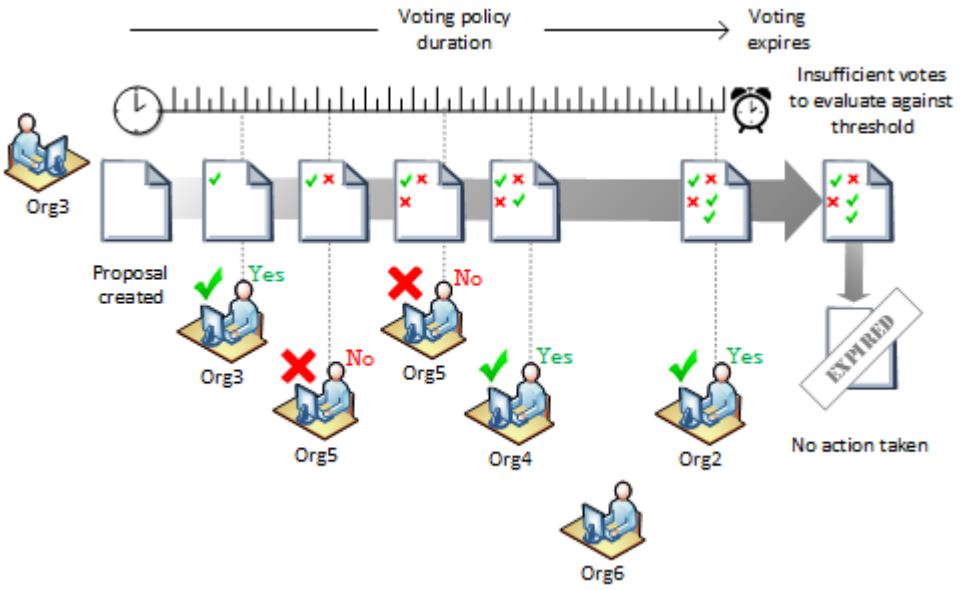
Example – Proposal rejected with partial member vote

For the following proposal, not all members cast a vote before the duration expired. However, enough No votes were cast to reject the proposal according to the voting policy. The proposal is `REJECTED`, and an invitation is not extended to the AWS account.



Example – Proposal expires and is not approved

For the following proposal, not all members cast a vote before the duration expired, and neither the number of Yes nor No votes were cast to determine the outcome of the proposal. The proposal is `EXPIRED`, and an invitation is not extended to the AWS account.



View Proposals

All proposals made on a network are shown on the **Proposals** page for a network. Both **Active** proposals and **Completed** proposals are listed. Active proposals are still open for voting. You can also list proposals from the AWS CLI using the `list-proposals` command, or using the ListProposals action with the Managed Blockchain API.

The **Proposals** page for a network shows both **Active** and **Completed** proposals, listing the **Proposal ID**, the name of the member that created the proposal, and the **Expiration Date (UTC)**, which is the creation time plus the proposal duration specified in the network's voting policy. You can choose a **Proposal ID** to vote on active proposals and to see more detail about any proposal, including the actions proposed and a voting summary by member.

Proposals have one of the following statuses:

- `IN_PROGRESS` - The proposal is active and open for member voting.
- `APPROVED` - The proposal was approved with sufficient `YES` votes among members according to the `VotingPolicy` specified for the `Network`. The specified proposal actions are carried out.
- `REJECTED` - The proposal was rejected with insufficient `YES` votes among members according to the `VotingPolicy` specified for the `Network`. The specified `ProposalActions` are not carried out.
- `EXPIRED` - Members did not cast the number of votes required to determine the proposal outcome before the proposal expired. The specified `ProposalActions` are not carried out.
- `ACTION_FAILED` - One or more of the specified `ProposalActions` in a proposal that was approved could not be completed because of an error. The `ACTION_FAILED` status occurs even if only one proposal action fails and other actions are successful.

To view proposals for a network using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. Choose **Networks**, choose a network **Name**, and then choose **Proposals**.
3. Choose a **Proposal ID** from the list to view more detailed information, such as the description, a summary of **Actions**, and a **Voting Summary**.

4. Under **Voting Summary**, expand **Votes** to see individual member's votes on the proposal to date.

To view proposals for a network using the AWS CLI

- Enter a command similar to the following example. Replace *n-MWY63ZJZU5HGNCMBQER7IN6OIU* with the network ID for which you want to list proposals.

```
aws managedblockchain list-proposals --network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU
```

The command returns output similar to the following:

```
{
    "Proposals": [
        {
            "CreationDate": 2019-04-08T23:40:20.628Z,
            "Description": "Proposal to add Example Corp. member",
            "ExpirationDate": 2019-04-09T23:40:20.628Z,
            "ProposalId": "p-ZR7KUD2YYNESLNG6RQ33X3FUFE",
            "ProposedByMemberId": "m-J46DNSFRTVCCLONS9DT5TTLS2A",
            "ProposedByMemberName": "org1",
            "Status": "IN_PROGRESS"

        }
    ]
}
```

To view the details of a proposal using the AWS CLI

- Enter a command similar to the following. Replace *n-MWY63ZJZU5HGNCMBQER7IN6OIU* with the network ID and *p-ZR7KUD2YYNESLNG6RQ33X3FUFE* with the proposal ID to view.

```
aws managedblockchain get-proposal --network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU --
proposal-id p-ZR7KUD2YYNESLNG6RQ33X3FUFE
```

The command returns output similar to the following:

```
{
   "Proposal": {
      "Actions": {
         "Invitations": [
            {
               "Principal": "0123456789012"
            }
         ],
      "CreationDate": 2019-04-08T23:40:20.628Z,
      "Description": "Proposal to invite AWS Acct 0123456789012",
      "ExpirationDate": 2019-04-08T23:40:20.628Z,
      "NetworkId": "n-MWY63ZJZU5HGNCMBQER7IN6OIU",
      "NoVoteCount": 1,
      "OutstandingVoteCount": 3,
      "ProposalId": "p-ZR7KUD2YYNESLNG6RQ33X3FUFE",
      "ProposedByMemberId": "m-J46DNSFRTVCCLONS9DT5TTLS2A",
      "ProposedByMemberName": "org1",
      "Status": "IN_PROGRESS",
      "YesVoteCount": 2
   }
}
```

Vote on a Proposal

You can use the AWS Management Console, the AWS CLI `vote-on-proposal` command, or the VoteOnProposal action of the Managed Blockchain API to vote Yes or No on an active proposal. You cannot change a vote after you make it.

To vote on a proposal using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. Choose **Networks**, choose a network **Name**, and then choose **Proposals**.
3. From the list of **Active** proposals, choose a **Proposal ID**.
4. Under **Vote on proposal**, choose the member to vote as from the list, and then choose **Yes** or **No**.
5. When prompted, choose **Confirm**.

To vote on a proposal using the AWS CLI

- Use the `vote-on-proposal` command as shown in the following example. Replace the values of `--network-id`, `--member-id`, and `--vote` as appropriate.

```
aws managedblockchain vote-on-proposal --network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU --
proposal-id p-ZR7KUD2YYNESLNG6RQ33X3FUFE --member-id m-K46ICRRXJRCGRNNS4ES4XUUS5A --
vote YES
```

Create an Invitation Proposal

You can use the AWS Management Console, the AWS CLI, or the Managed Blockchain API to create an invitation proposal.

To create an invitation proposal using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. From the navigation pane, choose **Networks**, and then choose the network to which you want to invite an AWS account.
3. Choose **Proposals** and then choose **Propose invitation**.
4. For **Submit proposal as**, choose the member in your account that submits the proposal.

   Note
   The member who submits the proposal must also vote on it. A Yes vote is not automatically assumed.
5. Enter an optional **Description** . The description appears to other members. It's a good way to communicate key points or a reminder about the proposal before they vote.
6. For each AWS account that you want to invite, enter the account number in the space provided. Choose **Add** to enter additional accounts.
7. Choose **Create**.

To create an invitation proposal using the AWS CLI

- Type a command similar to the following. Replace the value of `Principal` with the AWS account ID that you want to invite. Replace the value of `--member-id` with the value for the member in your account that submits the proposal.

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain create-proposal \
--actions Invitations=[{Principal=123456789012}] \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
--member-id m-K46ICRRXJRCGRNNS4ES4XUUS5A
```

The command returns the proposal ID, as shown in the following example:

```
{
    "ProposalId": "p-ZR7KUD2YYNESLNG6RQ33X3FUFE"
}
```

Create a Proposal to Remove a Network Member

To create a proposal to remove a member using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. From the navigation pane, choose **Networks**, and then choose the network.
3. Choose **Proposals** and then choose **Propose removal**.
4. For **Submit proposal as**, choose the member in your account that submits the proposal.

   Note
   The member who submits the proposal must also vote on it. A Yes vote is not automatically assumed.
5. Enter an optional **Description** . The description appears to other members. It's a good way to communicate key points or a reminder about the proposal before they vote.
6. For each member that you want to remove, enter the member ID in the space provided. Choose **Add** to enter additional members.

To create a removal proposal using the AWS CLI

- Type a command similar to the following. Replace the value of `Principal` with the AWS account ID that you want to invite. Replace the value of `--member-id` with the value for the member in your account that submits the proposal.

```
[ec2-user@ip-192-0-2-17 ~]$ aws managedblockchain create-proposal \
--actions Removals=[{MemberID=m-K46ICRRXJRCGRNNS4ES4XUUS5A}] \
--network-id n-MWY63ZJZU5HGNCMBQER7IN6OIU \
--member-id m-J46DNSFRTVCCLONS9DT5TTLS2A
```

The command returns the proposal ID, as shown in the following example:

```
{
    "ProposalId": "p-ZR7KUD2YYNESLNG6RQ33X3FUFE"
```

```
}
```

Automating Managed Blockchain Proposal Notifications with CloudWatch Events

Amazon CloudWatch Events enables you to automate your AWS services and respond automatically to system events. Events from AWS services are delivered to CloudWatch Events in near real time on a best-effort basis. You can write simple rules to indicate which events are of interest to you, and what automated actions to take when an event matches a rule. With Managed Blockchain, you can monitor CloudWatch Events events to respond to proposals, including invitations sent to your AWS account to join a network, and notification that proposals are `APPROVED` or `REJECTED`. Some examples include notifying an Amazon SNS topic or an AWS SMS queue when an invitation is sent or when a proposal made by a member in your account changes status.

For more information, see the Amazon CloudWatch Events User Guide.

Example Managed Blockchain Events

AWS Account Received an Invitation Event

The `detail-type` of these messages is `Managed Blockchain Invitation State Change`.

```
{
    "version": "0",
    "id": "abcd1234-eeee-4321-a1a2-123456789012",
    "detail-type": "Managed Blockchain Invitation State Change",
    "source": "aws.managedblockchain",
    "account": "123456789012",
    "time": "2019-04-08T23:40:20.628Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "invitationId": "i-XL9MDD6LVWWDNA9FF94Y4TFTE",
        "networkId": "n-MWY63ZJZU5HGNCMBQER7IN6OIU",
        "networkName": "ExampleCorpNetwork",
        "status": "PENDING",
        "expirationDate": "2019-04-09T23:40:20.628Z",
        "message": "You have received invitation i-XL9MDD6LVWWDNA9FF94Y4TFTE for Amazon
 Managed Blockchain Network n-MWY63ZJZU5HGNCMBQER7IN6OIU and it will expire at 2016-12-16
 20:42 UTC."
    }
}
```

Proposal State Change Event

The `detail-type` of these messages is `Managed Blockchain Proposal State Change`. The following example shows an event for a proposal that changed state to `APPROVED`.

```
{
    "version": "0",
    "id": "abcd1234-eeee-4321-a1a2-123456789012",
    "detail-type": "Managed Blockchain Proposal State Change",
    "source": "aws.managedblockchain",
    "account": "123456789012",
    "time": "2019-04-08T23:40:20.628Z",
```

```
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "proposalId": "p-ZR7KUD2YYNESLNG6RQ33X3FUFE",
        "networkId": "n-MWY63ZJZU5HGNCMBQER7IN6OIU",
        "status": "APPROVED",
        "proposedByMemberId": "m-K46ICRRXJRCGRNNS4ES4XUUS5A",
        "proposedByMemberName": "NetworkMember1",
        "expirationDate": "2019-04-09T23:40:20.628Z",
        "description": "Proposal to remove AnyCompany from supply chain blockchain
 network.",
        "message": "Voting on proposal p-ZR7KUD2YYNESLNG6RQ33X3FUFE in Amazon Managed
 Blockchain Network n-MWY63ZJZU5HGNCMBQER7IN6OIU completed at 2016-19-16T20:10:50Z UTC and
 the proposal was approved."
    }
}
```

Work with Hyperledger Fabric Components and Chaincode

You access services and applications in Hyperledger Fabric on Managed Blockchain from a framework client machine. The client runs tools and applications that you install for the version of Hyperledger Fabric on the network.

The client accesses Managed Blockchain endpoints for components such as the CA, the orderer, and peer nodes through an interface VPC endpoint that you set up in your AWS account. The client machine must have access to the interface VPC endpoint to access resource endpoints. For more information, see Create an Interface VPC Endpoint for Hyperledger Fabric on Amazon Managed Blockchain (p. 43).

You can get the endpoints that networks, members, and peer nodes make available using the AWS Management Console, or using `get` commands and actions with the Managed Blockchain AWS CLI or SDK.

An AWS CloudFormation template to create a Hyperledger Fabric client is available in amazon-managed-blockchain-client-templates repository on Github. For more information, see the readme.md in that repository. For more information about using AWS CloudFormation, see Getting Started in the *AWS CloudFormation User Guide*.

Topics

Register and Enroll a Hyperledger Fabric Admin

Only identities who are admins within a Hyperledger Fabric member can install, instantiate, and query chaincode.

Creating an admin in Hyperledger Fabric is a three-step process.

1. You *register* the identity with the Hyperledger Fabric CA. Registering stores the user name and password in the CA database as an admin.
2. After you register, you *enroll* the identity. This sends the CA a Certificate Signing Request (CSR). The CA validates that the identity is registered and otherwise valid, and returns a signed certificate. The certificate is stored in the Hyperledger Fabric client machine's local Membership Service Provider (MSP).
3. You then copy the certificate to the `admincerts` subdirectory, and the certificate validates the role of the identity as an admin. Similarly, the CA updates the local MSP for the member's peer nodes and the ordering service so that the admin is recognized.

For more information, see Fabric CA User's Guide and Membership in Hyperledger Fabric documentation.

When you first create a member in a Hyperledger Fabric network on Managed Blockchain, you specify the member's first user. Managed Blockchain registers the identity of this user automatically with the Hyperledger Fabric CA. This is called a *bootstrap identity*. Even though the identity is registered, the remaining steps need to be performed. The identity must then enroll itself as an admin and certificates

must be updated. After the steps are complete, the identity can install and instantiate chaincode and can be used to enroll additional identities as admins.

After you enroll an identity as an admin, it may take a minute or two until the identity is able to use the admin certificate to perform tasks.

Important
Managed Blockchain does not support revoking user certificates. After an admin is created, the admin persists for the life of the member.

To register and enroll an identity as an admin, you must have the following:

- The member CA endpoint
- The user name and password either of the bootstrap identity or of an admin with permissions to register and enroll identities
- A valid certificate file and the path to the MSP directory of your identity, which will register the new administrator

Registering an Admin

The following example uses a Fabric-CA Client CLI `register` command to register an admin with these options:

- `--url` specifies the endpoint of the CA along with an existing user name of an admin with permissions to register, such as the bootstrap identity. The example uses a user name of `AdminUser` with password `Password123`.
- `--id.name` and `--id.secret` parameters establish the user name and password for the new admin.
- `--id.type` is set to `user` and `--id.affiliation` is set to the member name to which the admins belong. The example member name is `org1`.
- `--id.attrs` is set to `'hf.admin=true'`. This is a property specific to Managed Blockchain that registers the identity as an admin.
- The `--tls.certfiles` option specifies the location and file name of the Managed Blockchain TLS certificate that you copied from Amazon S3 (see Step 5.1: Create the Certificate File (p. 18)).
- `--mspdir` specifies the MSP directory on the local machine where certificates are saved. The example uses `/home/ec2-user/admin-msp`.

```
fabric-ca-client register \
--url https://AdminUser:Password123@ca.m-K46ICRRXJRCGRNNS4ES4XUUS5A.n-
MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.us-east-1.amazonaws.com:30002 \
--id.name AdminUser2 --id.secret Password456 \
--id.type user --id.affiliation org1 \
--id.attrs 'hf.Admin=true' --tls.certfiles /home/ec2-user/managedblockchain-tls-chain.pem \
--mspdir /home/ec2-user/admin-msp
```

Enrolling an Admin

After registering an identity as an admin, or creating a member along with the bootstrap identity, you can use the Fabric-CA Client CLI `enroll` command to enroll that identity as an admin. This is shown in the following example using these options:

- `-u` (an alternative for `--url`) specifies the endpoint of the CA along with the user name and password of the identity that you are enrolling.
- `tls.certfiles` specifies the location and file name of the Managed Blockchain TLS certificate that you copied from Amazon S3 (see Step 5.1: Create the Certificate File (p. 18)).

- `-M` (an alternative for `--mspdir`) specifies the MSP directory on the local machine where certificates are saved. The example uses `/home/ec2-user/admin-msp`.

```
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
```

Copying the Admin Certificate

After you enroll the admin, copy the certificates from the `signcerts` directory to the `admincerts` directory as shown in the following example. The MSP directory `/home/ec2-user/admin-msp` is used in the example, and the example assumes that you are running the command in the `/home/ec2-user` directory.

```
cp -r admin-msp/signcerts admin-msp/admincerts
```

Work with Channels

A *channel* is a private communication pathway between two or more members of a Hyperledger Fabric network on Amazon Managed Blockchain. Each transaction on a Hyperledger Fabric network occurs on a channel. A network must contain at least one channel and a network on Managed Blockchain can have up to eight channels.

Each channel has its own transaction ledger. The ordering service on the network manages transactions according to a channel configuration that a member creates. The channel configuration specifies which members participate in the channel, the peer nodes that can transact on the channel, and the policies that govern channel updates.

To participate on a channel, a member must join at least one peer node to the channel. A peer node can belong to multiple channels, but ledger data from one channel can't pass to ledger data in another channel. A member shares their admin certificates and root CA with the channel creator when they join the channel. These artifacts allow the ordering service to authenticate and authorize members of the channel when they join their peer nodes to the channel.

Optionally, a channel also can contain private data collections. A private data collection allows a subset of channel members to share ledger data independently of the ordering service. Private data collections are supported only with Hyperledger Fabric 1.4 or later. Private data collections require that at least one peer node for a member is set up as an anchor peer. For more information, see Add an Anchor Peer to a Channel (p. 67) and Work with Private Data Collections in Hyperledger Fabric on Managed Blockchain (p. 71).

Create a Channel

A channel creator is a member that uses their Hyperledger Fabric client to create a channel configuration file. The channel creator then runs a command that outputs a channel transaction file based on that configuration file. Finally, the channel creator uses the transaction file to run a command that creates the channel with the Hyperledger Fabric ordering service on Managed Blockchain. The channel creator must get security artifacts from channel members to be able to add the members to the channel.

After the channel is created, channel members, including the channel creator, download the channel genesis block to their respective Hyperledger Fabric client machines and join peer nodes to the channel.

To create and join a Hyperledger Fabric channel on Managed Blockchain

The following steps specify file locations in terms of both the Hyperledger Fabric client machine's file system and the Hyperledger Fabric CLI Docker container file system, as appropriate. When the CLI is installed, the Docker Compose configuration file maps these locations to one another. For more information, see step 4.4 (p. 16) in the Getting Started (p. 6) tutorial of this guide.

In the following examples, the client machine directory `/home/ec2-user` is mapped to the container directory `/opt/home`. In steps where you save artifacts, the steps use the client machine file system. In steps where you reference artifacts in CLI commands and configuration files, the steps use the container file system.

1. Each member that joins a channel shares its security artifacts with the channel creator. The channel creator saves the artifacts on the Hyperledger Fabric client machine. The channel creator then references the location of these artifacts in the next step. For more information about sharing artifacts, see Step 8.4 (p. 27) and 8.5 in the getting started tutorial of this guide. Sharing artifacts is a prerequisite for the steps below.
2. On your Hyperledger Fabric client machine, create a configuration file in yaml format named `configtx.yaml`. Save this file to a file location on your Hyperledger Fabric client machine that is mapped to the CLI container file system. The following steps use `/home/ec2-user`, which is mapped to `/opt/home`.

   Use the examples below as a starting point. For more information about configuration files, see Channel configuration (configtx) in the Hyperledger Fabric documentation. Replace the following configuration parameters as appropriate for your channel, and add sections as required for additional members.

   - *`Org<n>MemberID`* is the respective member ID for each channel participant, for example, `m-K46ICRRXJRCGRNNS4ES4XUUS5A`.
   - *`Org<n>AdminMSPDir`* is the Docker container file system directory where security artifacts for each corresponding member are saved. For example, *`opt/home/Org2AdminMSP`* references artifacts saved to `/home/ec2-user/Org2AdminMSP` on the channel creator's client machine.

   The configtx file for a Hyperledger Fabric 1.4 network includes application settings to enable expanded features. These settings are not supported in Hyperledger Fabric 1.2 channels. Examples for version 1.2 and 1.4 networks are provided below.

   Version 1.4

```
################################################################################
#
# 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: Org1MemberID
        # ID to load the MSP definition as
        ID: Org1MemberID
        #msp dir of org1 in the docker container
        MSPDir: /opt/home/Org1AdminMSP
        # 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: Org2MemberID
        ID: Org2MemberID
        #msp dir of org2 in the docker container
        MSPDir: /opt/home/Org2AdminMSP
        AnchorPeers:
            - Host:
              Port:
################################################################################
#
#   CAPABILITIES
#
#   This section defines the capabilities of fabric network. This is a new
#   concept as of v1.1.0 and should not be utilized in mixed networks with
#   v1.0.x peers and orderers.  Capabilities define features which must be
#   present in a fabric binary for that binary to safely participate in the
#   fabric network.  For instance, if a new MSP type is added, newer binaries
#   might recognize and validate the signatures from this type, while older
#   binaries without this support would be unable to validate those
#   transactions.  This could lead to different versions of the fabric binaries
#   having different world states.  Instead, defining a capability for a channel
#   informs those binaries without this capability that they must cease
#   processing transactions until they have been upgraded.  For v1.0.x if any
#   capabilities are defined (including a map with all capabilities turned off)
#   then the v1.0.x peer will deliberately crash.
#
################################################################################
Capabilities:
    # Channel capabilities apply to both the orderers and the peers and must be
    # supported by both.
    # Set the value of the capability to true to require it.
    # Note that setting a later Channel version capability to true will also
    # implicitly set prior Channel version capabilities to true. There is no need
    # to set each version capability to true (prior version capabilities remain
    # in this sample only to provide the list of valid values).
    Channel: &ChannelCapabilities
        # V1.4.3 for Channel is a catchall flag for behavior which has been
        # determined to be desired for all orderers and peers running at the v1.4.3
        # level, but which would be incompatible with orderers and peers from
        # prior releases.
        # Prior to enabling V1.4.3 channel capabilities, ensure that all
        # orderers and peers on a channel are at v1.4.3 or later.
        V1_4_3: true
        # V1.3 for Channel enables the new non-backwards compatible
        # features and fixes of fabric v1.3
        V1_3: false
        # V1.1 for Channel enables the new non-backwards compatible
        # features and fixes of fabric v1.1
        V1_1: false
    # Application capabilities apply only to the peer network, and may be safely
    # used with prior release orderers.
    # Set the value of the capability to true to require it.
    # Note that setting a later Application version capability to true will also
    # implicitly set prior Application version capabilities to true. There is no
 need
    # to set each version capability to true (prior version capabilities remain
    # in this sample only to provide the list of valid values).
    Application: &ApplicationCapabilities
        # V1.4.2 for Application enables the new non-backwards compatible
        # features and fixes of fabric v1.4.2
        V1_4_2: true
        # V1.3 for Application enables the new non-backwards compatible
        # features and fixes of fabric v1.3.
        V1_3: false
```

```
        # V1.2 for Application enables the new non-backwards compatible
        # features and fixes of fabric v1.2 (note, this need not be set if
        # later version capabilities are set)
        V1_2: false
        # V1.1 for Application enables the new non-backwards compatible
        # features and fixes of fabric v1.1 (note, this need not be set if
        # later version capabilities are set).
        V1_1: false
################################################################################
#
# 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:
    Capabilities:
        <<: *ApplicationCapabilities
################################################################################
#
# 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
```

Version 1.2

```
################################################################################
#
#   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/Org1AdminMSP
            # 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/Org2AdminMSP
        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
```

3. The channel creator uses the Hyperledger Fabric CLI configtxgen command as shown in the following example to write a channel creation transaction.

   Replace the following configuration parameters as appropriate for your channel.

   - */opt/home/ourchannel.pb* is the channel creation transaction file that the command creates. Specify the file location using the container file system.
   - *TwoOrgChannel* is the profile specified in the `configtx.yaml` from the previous step.
   - *ourchannel* is the channel ID that will be used for the channel.
   - */opt/home/* for the `--configPath` option is the container location where the `configtx.yaml` file is saved.

```
docker exec cli configtxgen \
-outputCreateChannelTx /opt/home/ourchannel.pb \
-profile TwoOrgChannel -channelID ourchannel \
--configPath /opt/home/
```

4. The channel creator uses the Hyperledger Fabric CLI channel create command to create a channel and write the genesis block to the orderer. The `-f` option specifies the transaction file that you created in the previous step, and the `$ORDERER` environment variable in the example has been set to the endpoint of the network orderer for simplicity—for example, `orderer.n-MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.amazonaws.com:30001`.

```
docker exec cli peer channel create -c ourchannel \
```

```
-f /opt/home/ourchannel.pb -o $ORDERER \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

5. To join a peer node to the channel, each member must fetch the channel genesis block from the orderer, write it to a file, and then reference that genesis block when they join their peer or peers.

   a. To get the channel genesis block, each member uses the Hyperledger Fabric CLI peer channel fetch oldest command. In the following example, the genesis block is written to a file in the container file system, */opt/home/ourchannel.block*. The `$ORDERER` variable is used in the same ways the previous step, and the `ourchannel` channel ID is specified.

   ```
   docker exec cli peer channel fetch oldest /opt/home/ourchannel.block \
   -c ourchannel -o $ORDERER \
   --cafile /opt/home/managedblockchain-tls-chain.pem --tls
   ```

   b. Using the channel genesis block file created above, each member uses the Hyperledger Fabric peer channel join command to join their member's peer node to the channel.

   ```
   docker exec cli peer channel join -b /opt/home/ourchannel.block \
   -o $ORDERER --cafile /opt/home/managedblockchain-tls-chain.pem --tls
   ```

Add an Anchor Peer to a Channel

Anchor peers within an organization enable cross-organization communication among peer nodes using the Hyperledger Fabric gossip protocol. In general, the gossip protocol identifies other available member peers, disseminates ledger data across peers on a channel, and brings new peers up to date quickly.

Although anchor peers are not strictly required for an organization on a channel, at least one anchor peer is required for an organization to participate in features that use the gossip protocol, such as private data collections and service discovery. In addition, we recommend that an organization creates redundant anchor peers on a channel for resiliency. For more information, see Anchor Peer and Gossip data dissemination protocol in the Hyperledger Fabric documentation.

The steps in this section demonstrate how you can update a channel configuration to add a peer as an anchor peer for your organization in Amazon Managed Blockchain. The peer that you want to designate as an anchor peer for your organization must first be joined to the channel.

Prerequisites and Assumptions

The steps in this section are based on the assumption that you are using a network, client machine, member, peer, and joint channel similar to those that you create during the getting started tutorial (p. 6) in this guide.

The example commands shown presume that the environment variables, the directory locations on the client machine, the locations in the Docker container, the properties of members and peer nodes, and the properties of the joint channel from that tutorial are being used. You might find it useful to review that information before running these commands.

For more information, see the following resources:

- Create a client (p. 12)
- Enroll an administrator (p. 17)
- Create the peer node (p. 11)
- Create a multi-member channel (p. 24)
- Join the peer node to the channel (p. 31)

Adding a Peer as an Anchor Peer

A peer node must already be joined to a channel to be set up as an anchor peer. To update a peer to be an anchor peer, you download a configuration file from the orderer on the Managed Blockchain network into a directory that you create on the Hyperledger Fabric client. Next, you use the configtxlator command and the jq tool to manipulate the original protobuf configuration file between protobuf and JSON. Then you use your final protobuf file to update the channel configuration in the orderer.

1. Create a directory on your Hyperledger Fabric client machine, *channel-artifacts*, that you can use to save channel configuration artifacts that you create.

```
cd /home/ec2-user
```

```
mkdir channel-artifacts
```

2. Use the Hyperledger Fabric peer channel `fetch` sub-command to get the channel configuration and save it locally, as shown in the following command. In this example, *ourchannel* is the Channel ID. This Channel ID is the same one established in the step for creating the multi-member channel (p. 31) in the Getting Started (p. 6) tutorial of this guide. The command saves the configuration file to the directory you created in a protobuf file named *ourchannelCfg.pb*.

```
docker exec cli peer channel fetch config \
/opt/home/channel-artifacts/ourchannelCfg.pb \
-c ourchannel -o $ORDERER \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

3. Decode the `ourchannelCfg.pb` configuration from protobuf into a JSON object as shown in the following command.

```
docker exec cli configtxlator proto_decode \
--input /opt/home/channel-artifacts/ourchannelCfg.pb --type common.Block \
--output /opt/home/channel-artifacts/config_block.json
```

4. Switch directories to the `channel-artifacts` directory.

```
cd channel-artifacts
```

5. Convert the JSON object that you created from the protobuf file into a streamlined JSON file named `config.json`, and then copy it to `config_copy.json`, as shown in the following commands.

```
sudo jq .data.data[0].payload.data.config config_block.json > config.json
```

```
cp config.json config_copy.json
```

6. Use the `jq` tool to modify the `config_copy.json` configuration file by adding an anchor peer and to save the modifications to a new JSON configuration file named `modified_config.json` as shown in the following command. Replace the items below as appropriate for your Managed Blockchain member, network, and peer configuration.

   - *OrgMemberID* is the ID of a Managed Blockchain member that belongs to the channel and owns the peer node that will be an anchor peer. For example, `m-K46ICRRXJRCGRNNS4ES4XUUS5A`.
   - *PeerNodeDNS* is the endpoint, minus the port identifier, of the peer node. For example, `nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y.m-K46ICRRXJRCGRNNS4ES4XUUS5A.n-MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.us-east-1.amazonaws.com`.

- *PortNumber* is the port number included in the **peer node endpoint** in the console or the `PeerEndpoint` in the Managed Blockchain API for the peer. For example `30003`.

```
jq '.channel_group.groups.Application.groups["OrgMemberID"].values +=
 {"AnchorPeers":{"mod_policy": "Admins","value":{"anchor_peers": [{"host":
 "PeerNodeDNS","port": PortNumber}]},"version": "0"}}' config_copy.json >
 modified_config.json
```

7. Return the configuration JSON files to protobuf format, calculate the difference between them, and then generate a new `config_update.json` configuration file as shown in the following commands.

```
docker exec cli configtxlator proto_encode \
--input /opt/home/channel-artifacts/config.json \
--type common.Config \
--output /opt/home/channel-artifacts/config.pb
```

```
docker exec cli configtxlator proto_encode \
--input /opt/home/channel-artifacts/modified_config.json \
--type common.Config \
--output /opt/home/channel-artifacts/modified_config.pb
```

```
docker exec cli configtxlator compute_update \
--channel_id ourchannel \
--original /opt/home/channel-artifacts/config.pb \
--updated /opt/home/channel-artifacts/modified_config.pb \
--output /opt/home/channel-artifacts/config_update.pb
```

```
docker exec cli configtxlator proto_decode \
--input /opt/home/channel-artifacts/config_update.pb \
--type common.ConfigUpdate \
--output /opt/home/channel-artifacts/config_update.json
```

8. Change permissions for the `config_update.json` file you created above so that you can pipe its contents along with transaction envelope parameters to the `jq` tool and create the `config_update_in_envelope.json` file, as shown in the following commands.

```
sudo chmod 755 config_update.json
```

```
echo '{"payload":{"header":{"channel_header":{"channel_id":"ourchannel",
 "type":2}},"data":{"config_update":'$(cat config_update.json)'}}}' | jq . >
 config_update_in_envelope.json
```

9. Convert the `config_update_in_envelope.json` file to a `config_update_in_envelope.pb` configuration file in protobuf format as shown in the following command.

```
docker exec cli configtxlator proto_encode --input /opt/home/channel-artifacts/
config_update_in_envelope.json --type common.Envelope --output /opt/home/channel-
artifacts/config_update_in_envelope.pb
```

10. Specify the `config_update_in_envelope.pb` configuration file to update the channel configuration on the orderer for the Managed Blockchain network as shown in the following command.

```
docker exec cli peer channel update \
```

```
-f /opt/home/channel-artifacts/config_update_in_envelope.pb \
-c ourchannel -o $ORDERER \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

Develop Hyperledger Fabric Chaincode

Smart contracts in Hyperledger Fabric are known as *chaincode*.

- For a conceptual overview of chaincode, see Smart Contracts and Developing Applications in the Hyperledger Fabric documentation.
- For links to Hyperledger Fabric SDKs, see Getting Started in the Hyperledger Fabric documentation.

Considerations and Limitations When Developing Chaincode for Managed Blockchain

- All Hyperledger Fabric networks on Managed Blockchain support a maximum of 8 channels per network, regardless of network edition.
- **Hyperledger Fabric Shims for Node.js**

  To simplify chaincode development, Managed Blockchain includes versions of the fabric-shim library. This library provides a low-level chaincode interface between applications, peers, and the Hyperledger Fabric system for chaincode applications developed with Node.js. The library version is specified using the `dependencies` object in the `package.json` file bundled with your chaincode. You can specify a version explicitly or use the semantic versioner (semver) for NPM to specify a version range. The following library versions are available without bundling.

  - **Hyperledger Fabric 1.4**
    - 1.4.0
    - 1.4.1
    - 1.4.2
    - 1.4.4
    - 1.4.5
  - **Hyperledger Fabric 1.2**
    - 1.2.0
    - 1.2.2
    - 1.2.3
    - 1.2.4

  Dependencies on other versions of fabric-shim or other library packages require that you bundle them with your chaincode because peer nodes do not have internet access to the NPM repository.
- The default limit for the size of a transaction payload is 1MB. To request a limit increase, create a case using the AWS Support Center.

Topics

Work with Private Data Collections in Hyperledger Fabric on Managed Blockchain

Network members that participate on a Hyperledger Fabric channel on Managed Blockchain may want to keep some data private from other members on the same channel. Hyperledger Fabric *private data collections* allow you to accomplish this. Private data collections allow a subset of members on a channel to endorse, commit, and query private data without having to create a separate channel to exclude other members. Private data reduces the overhead associated with channel creation and maintenance. In addition, private data is disseminated only between authorized peers using the Hyperledger Fabric *gossip protocol*, so the data is kept confidential from the ordering service.

To create a private data collection, a peer creates a *private data definition* during chaincode instantiation. The definition establishes the members who have access to the private data collection. It also defines a policy that governs how transactions are endorsed and committed.

Private data is stored on each peer node in a way that is accessible to the peer chaincode and logically distinct from other channel transaction data. This helps ensure that channel members who are not specified in the private data definition cannot access the private data.

For more information about private data collections, see Private Data and the Private Data architecture topic in Hyperledger Fabric documentation.

Private data collections with Managed Blockchainlong require the following:

- A network created using Hyperledger Fabric version 1.4 or later.
- A channel configuration that supports the latest features of your version. The multi-member channel configuration (p. 28) featured in the Getting Started (p. 6) tutorial of this guide enables the latest features.
- At least one anchor peer for each member that participates in the private data collection. This is because Hyperledger Fabric uses the gossip protocol to distribute private data to authorized peers. For more information, see Add an Anchor Peer to a Channel (p. 67).

Develop Hyperledger Fabric Chaincode Using Java on Amazon Managed Blockchain

Hyperledger Fabric 1.4 networks on Amazon Managed Blockchain are based on the following components:

- Hyperledger Fabric v1.4.7
- Hyperledger Fabric CA v1.4.7
- fabric-chaincode-java

Managed Blockchain Dependency on Javaenv Docker Image

Managed Blockchain uses the `fabric-chaincode-java` package to build the `hyperledger/fabric-javaenv` Docker image. This image is the base layer of the Java chaincode container. It has built-in support to build the Java chaincode using Gradle or Maven at the following versions:

- gradle@4.6
- maven@3.5.0

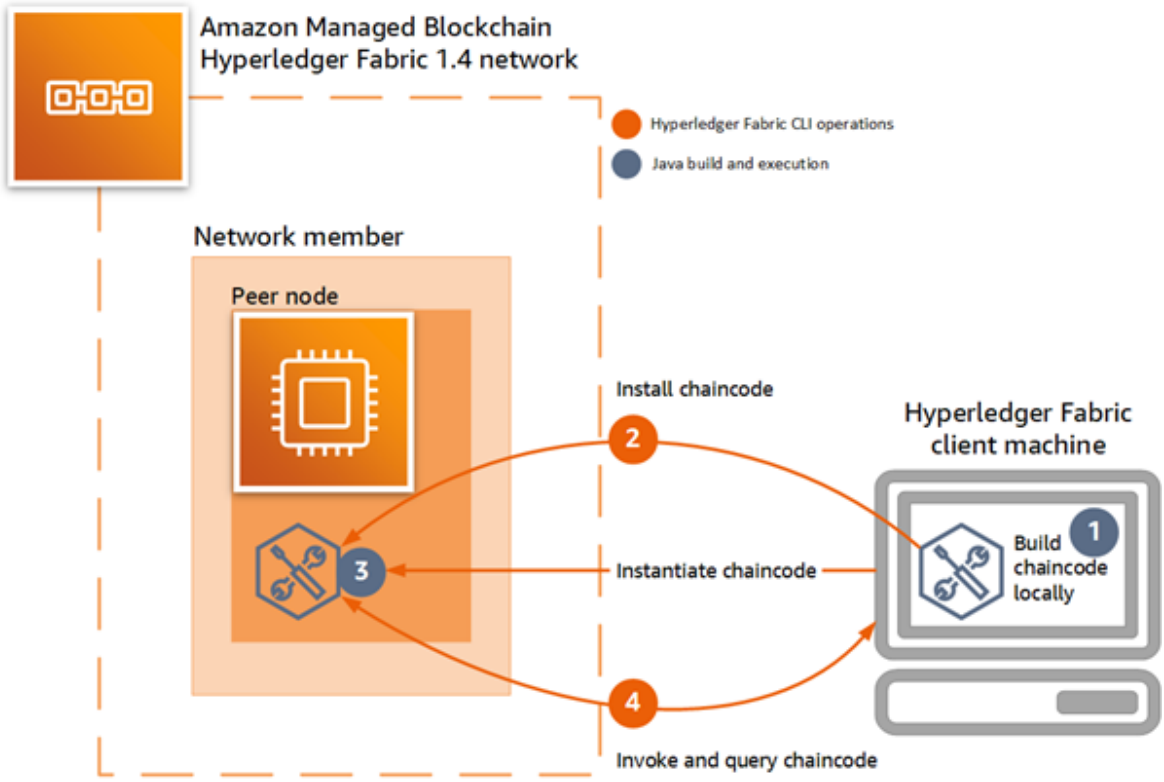
You can verify this by examining the Docker file in the fabric-chaincode-java package.

The default settings support Gradle or Maven for building Java chaincode. Once the `javaenv` Docker image is deployed, it is used to provision all new peer nodes and instantiate Java chaincode in those peer nodes. You cannot modify these settings or update versions on the `javaenv` Docker image because Managed Blockchain manages these.

Installing and Running Java Chaincode

The following is a high-level summary of the steps to install and run Java Chaincode using Managed Blockchain with Hyperledger Fabric 1.4

1. **Marshall dependencies locally** – This step uses Gradle and the Gradle shadow plugin to download dependencies to the client machine and package them for installation to the peer node.
2. **Install the chaincode** – This step packages the Java chaincode project and copies it to the Hyperledger Fabric peer node.
3. **Instantiate the chaincode** – This step uses Gradle to resolve the dependencies, access the dependency packages, and build the Java chaincode. This step also builds the Docker image of this Java chaincode, starts the chaincode container, and instantiates the first object in the ledger.
4. **Invoke and query the chaincode** – After the chaincode instantiates successfully, you can send query transactions or invoke transactions to modify the ledger.



Building Java Chaincode with Managed Blockchain: Step-by-step

Java chaincode can depend on other third-party packages during the **Instantiate chaincode** step. In a traditional installation of Hyperledger Fabric chaincode built with Java, Gradle automatically resolves dependencies by loading them locally in addition to downloading them from remote repositories. Managed Blockchain does not allow downloaded dependencies to help ensure the security and reliability of the network. For this reason, to run Java chaincode, or any chaincode, in a Hyperledger Fabric network

on Managed Blockchain, you manually download dependencies to the client machine and add them into the chaincode project.

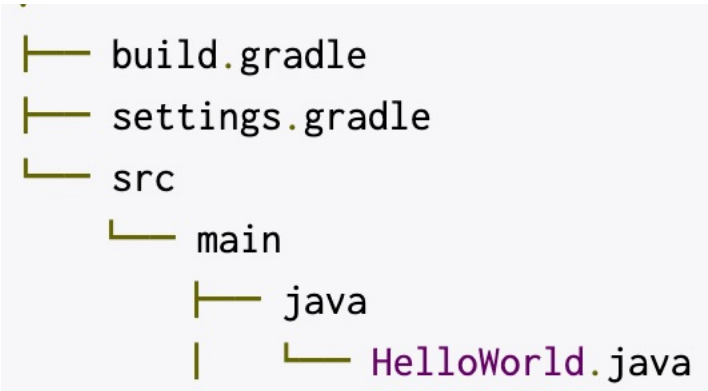
The steps in this section demonstrate how to use Gradle to build and deploy a sample Java chaincode project with fabric-shim@1.4.6 to a Hyperledger Fabric network on Managed Blockchain. The Gradle Shadow plugin is needed to combine all jar files into a single jar file.

Prerequisites and Assumptions

The steps below require that you have the following:

- A Hyperledger Fabric client machine with access to the internet and access to a member's peer node, ordering service, and CA on an Managed Blockchain network running Hyperledger Fabric v1.4 or later. For more information, see the following:
  - Create a Hyperledger Fabric Blockchain Network on Amazon Managed Blockchain (p. 33)
  - Create an Interface VPC Endpoint for Hyperledger Fabric on Amazon Managed Blockchain (p. 43)
  - Work with Hyperledger Fabric Peer Nodes on Managed Blockchain (p. 45)
  - Step 4: Create an Amazon EC2 Instance and Set Up the Hyperledger Fabric Client (p. 12). Optionally, modify step 4.4 (p. 16) when launching the Docker container to establish CLI variables demonstrated in the install and instantiate steps below.
- You must be a Hyperledger Fabric admin to install and instantiate chaincode. For more information, see Register and Enroll a Hyperledger Fabric Admin (p. 60).

The working directory of a typical Java chaincode project using Gradle on a Hyperledger Fabric client machine looks similar to the directory structure shown in the following diagram. The `src` directory includes the chaincode source. The `build.gradle` file declares all the dependencies and defines the scripted tasks. The steps below assume this directory structure.



Step 1: Download the Gradle Shadow plugin and add it to the chaincode project

Because Managed Blockchain peer nodes don't have internet access, the Gradle Shadow plugin needs to be downloaded manually to the client and added to the project. The plugin is downloaded from the Gradle plugin repository.

1. Create a sub-directory in the Java chaincode working directory on the Hyperledger Fabric client machine where you want to download the plugin, and then switch to that directory. The example below uses a directory named `plugin`.

```
mkdir plugin
cd plugin
```

2. Download the plugin.

```
wget https://plugins.gradle.org/m2/com/github/jengelman/gradle/plugins/shadow/2.0.4/
shadow-2.0.4.jar
```

3. Verify that the plugin was downloaded as expected.

```
ls

shadow-2.0.4.jar
```

4. After downloading the shadow plugin jar file, use an editor to modify the `build.gradle` file so that it references the local directory and plugin.

   Make sure to remove the `id 'com.github.johnrengelman.shadow' version '2.0.3'` entry from the `plugins` section.

   The following example shows an excerpt from an edited `build.gradle` file, with *relevant sections emphasized*:

```
buildscript {
    dependencies {
        classpath fileTree(dir: 'plugin', include:['*.jar'])
    }
}

plugins {
    id 'java'
}

group 'org.hyperledger.fabric-chaincode-java'
version '1.0-SNAPSHOT'

sourceCompatibility = 1.8


repositories {
    mavenLocal()
    mavenCentral()
}

dependencies {
    compile group: 'org.hyperledger.fabric-chaincode-java', name: 'fabric-chaincode-
shim', version: '1.4.6'
    testCompile group: 'junit', name: 'junit', version: '4.12'
}

apply plugin: 'com.github.johnrengelman.shadow'

shadowJar {
    baseName = 'chaincode'
    version = null
    classifier = null

    manifest {
        attributes 'Main-Class': 'org.hyperledger.fabric.example.SimpleChaincode'
    }
}
```

Step 2: Add fabric-chaincode-shim and the getDeps task to the chaincode project

Update `build.gradle` to add fabric-chaincode-shim as a dependency in the `dependencies` section. In addition, reference the task `getDeps` to download dependencies to the `libs` sub-directory on the client machine. The following example shows an excerpt from a `build.gradle` file with *relevant sections emphasized*.

```
buildscript {
    dependencies {
        classpath fileTree(dir: 'plugin', include:['*.jar'])
    }
}

plugins {
    id 'java'
}

group 'org.hyperledger.fabric-chaincode-java'
version '1.0-SNAPSHOT'

sourceCompatibility = 1.8

repositories {
    mavenLocal()
    mavenCentral()
}

dependencies {
    compile group: 'org.hyperledger.fabric-chaincode-java', name: 'fabric-chaincode-shim',
 version: '1.4.6'
    testCompile group: 'junit', name: 'junit', version: '4.12'
}

apply plugin: 'com.github.johnrengelman.shadow'

shadowJar {
    baseName = 'chaincode'
    version = null
    classifier = null

    manifest {
        attributes 'Main-Class': 'org.hyperledger.fabric.example.SimpleChaincode'
    }
}

task getDeps(type: Copy) {
  from sourceSets.main.compileClasspath
  into 'libs/'
}
```

Step 3: Build the project locally and download dependencies

Run `gradle build` and `gradle getDeps` as shown in the following examples to build the chaincode locally and download dependencies. When the `gradle getDeps` task runs, Gradle creates the `lib` sub-directory on the client machine and downloads dependencies to it.

1. Build the project using Gradle.

```
gradle build
```

The command creates output similar to the following example.

```
> Task :compileJava
Note: /home/ec2-user/fabric-samples/chaincode/chaincode_example02/java/src/main/java/org/
hyperledger/fabric/example/SimpleChaincode.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

Deprecated Gradle features were used in this build, making it incompatible with Gradle
 7.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/6.5/userguide/
command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 1s
2 actionable tasks: 1 executed, 1 up-to-date
```

2. Download chaincode project dependencies locally.

```
gradle getDeps
```

The command creates output similar to the following example.

```
Deprecated Gradle features were used in this build, making it incompatible with Gradle
 7.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/6.5/userguide/
command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 1s
1 actionable task: 1 executed
```

3. Verify that dependencies were downloaded to the libs sub-directory.

```
ls libs

animal-sniffer-annotations-1.17.jar  grpc-api-1.23.0.jar             jsr305-3.0.2.jar
                              netty-transport-4.1.38.Final.jar
annotations-4.1.1.4.jar              grpc-context-1.23.0.jar         netty-
buffer-4.1.38.Final.jar                     opencensus-api-0.21.0.jar
bcpkix-jdk15on-1.62.jar              grpc-core-1.23.0.jar            netty-
codec-4.1.38.Final.jar                      opencensus-contrib-grpc-metrics-0.21.0.jar
bcprov-jdk15on-1.62.jar              grpc-netty-1.23.0.jar           netty-codec-
http2-4.1.38.Final.jar                perfmark-api-0.17.0.jar
checker-compat-qual-2.5.2.jar        grpc-protobuf-1.23.0.jar        netty-codec-
http-4.1.38.Final.jar                 protobuf-java-3.9.0.jar
classgraph-4.8.47.jar                grpc-protobuf-lite-1.23.0.jar   netty-codec-
socks-4.1.38.Final.jar                protobuf-java-util-3.5.1.jar
commons-cli-1.4.jar                  grpc-stub-1.23.0.jar            netty-
common-4.1.38.Final.jar                     proto-google-common-protos-1.12.0.jar
commons-logging-1.2.jar              gson-2.7.jar                    netty-
handler-4.1.38.Final.jar                    swagger-annotations-2.0.0.jar
error_prone_annotations-2.3.2.jar    guava-26.0-android.jar          netty-handler-
proxy-4.1.38.Final.jar
fabric-chaincode-protos-1.4.6.jar    j2objc-annotations-1.1.jar      netty-
resolver-4.1.38.Final.jar
fabric-chaincode-shim-1.4.6.jar      javax
```

Step 4: Add dependencies downloaded in the previous step to the project

Use a text editor to update the `build.gradle` file again so that it reference the dependencies in the `libs` sub-directory. This adds the dependencies that you downloaded in the previous step to the project.

The following example shows an excerpt from a `build.gradle` file with *relevant sections emphasized*.

```
buildscript {
    dependencies {
        classpath fileTree(dir: 'plugin', include:['*.jar'])
    }
}

plugins {
    id 'java'
}

group 'org.hyperledger.fabric-chaincode-java'
version '1.0-SNAPSHOT'

sourceCompatibility = 1.8

repositories {
    mavenLocal()
    mavenCentral()
}

dependencies {
    compile fileTree(dir:'libs',includes:['*.jar'])
    testCompile group: 'junit', name: 'junit', version: '4.12'
}

apply plugin: 'com.github.johnrengelman.shadow'

shadowJar {
    baseName = 'chaincode'
    version = null
    classifier = null

    manifest {
        attributes 'Main-Class': 'org.hyperledger.fabric.example.SimpleChaincode'
    }
}

task getDeps(type: Copy) {
  from sourceSets.main.compileClasspath
  into 'libs/'
}
```

Step 5: Use the Hyperledger Fabric CLI to install the chaincode on the peer node

Use the Hyperledger Fabric CLI to package the Java chaincode project and copy it to the peer node. The `-l java` option is required to specify that the chaincode is Java-based. The path to the Java chaincode working directory is an absolute path, which is different from chaincode written in golang.

1. Before running the `instantiate`command, establish variables for the Hyperledger Fabric CLI Docker container as shown in the following example. Alternatively, you can configure a Docker compose file to establish these variables. For more information, see Step 4.4: Configure and Run Docker Compose to Start the Hyperledger Fabric CLI (p. 16) in the getting started tutorial. Replace the values with those appropriate for your network. Set `CORE_PEER_ADDRESS` to the endpoint of the peer node for which you want to instantiate chaincode, set `CORE_PEER_LOCALMSPID` to the ID of the member that

owns the peer node, and set `CORE_PEER_MSPCONFIGPATH` as shown, which is the location of the membership service provider directory on the peer node.

```
docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem
" \
-e "CORE_PEER_ADDRESS=nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y.m-K46ICRRXJRCGRNNS4ES4XUUS5A.n-
MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.us-east-1.amazonaws.com:30003" \
-e "CORE_PEER_LOCALMSPID=m-K46ICRRXJRCGRNNS4ES4XUUS5A" \
-e "CORE_PEER_MSPCONFIGPATH=/opt/home/admin-msp"
```

2. Running the Hyperledger Fabric `install` command on the client installs the chaincode on the peer node. The following example demonstrates an `install` command using the following flags.
   - The `-o` flag specifies the ordering service endpoint for the member.
   - The `-l` flag specifies that the chaincode language is `java`.
   - The `--cafile` flag specifies the location of the certificate for the ordering service that you copied when you set up the Hyperledger Fabric admin. For more information, see step 5.1 (p. 18) in the Getting Started (p. 6) tutorial.
   - The `--tls` flag specifies that communication with the ordering service uses TLS.
   - The `-p` flag specifies the location of the chaincode on the client machine. When installing Java chaincode, this must be an absolute path.
   - The `-n` and `-v` options establish the name and version of the chaincode. You'll reference this name and version when you instantiate the chaincode in the next step.

   For more information about options, see peer chaincode install in Hyperledger Fabric documentation.

```
cli peer chaincode install \
-o orderer.n-MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.amazonaws.com:30001 \
-l java \
--cafile /opt/home/managedblockchain-tls-chain.pem \
--tls -p  /opt/home/fabric-samples/chaincode/chaincode_example02/java -v MyCCVerNumber -
n MyJavaChaincodeName
```

Step 6:Use the Hyperledger Fabric CLI to instantiate the chaincode

Running the Hyperledger Fabric CLI `instantiate` command directs the peer node to build the Java chaincode, create the chaincode image, start the chaincode container, and instantiate object values. Similar to a previous example, the command should have the `-l java` option specified.

1. Before running the `install`command, establish variables for the Hyperledger Fabric CLI Docker container as shown in the following example. Alternatively, you can configure a Docker compose file to establish these variables. For more information, see Step 4.4: Configure and Run Docker Compose to Start the Hyperledger Fabric CLI (p. 16) in the getting started tutorial. Set `CORE_PEER_ADDRESS` to the endpoint of the peer node for which you want to instantiate chaincode, set `CORE_PEER_LOCALMSPID` to the ID of the member that owns the peer node, and set `CORE_PEER_MSPCONFIGPATH` as shown, which is the location of the membership service provider directory on the peer node.

```
docker exec -e "CORE_PEER_TLS_ENABLED=true" \
-e "CORE_PEER_TLS_ROOTCERT_FILE=/opt/home/managedblockchain-tls-chain.pem" \
-e "CORE_PEER_ADDRESS=nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y.m-K46ICRRXJRCGRNNS4ES4XUUS5A.n-
MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.us-east-1.amazonaws.com:30003" \
-e "CORE_PEER_LOCALMSPID=m-K46ICRRXJRCGRNNS4ES4XUUS5A" \
-e "CORE_PEER_MSPCONFIGPATH=/opt/home/admin-msp"
```

2. Running the Hyperledger Fabric CLI `instantiate` command on the client instantiates the chaincode on the peer. The following example demonstrates an `instantiate` command.
   - The `-o` flag specifies the ordering service endpoint for the member.
   - The `-l` flag specifies that the chaincode language is `java`.
   - The `--cafile` flag specifies the location of the certificate for the ordering service that you copied when you set up the Hyperledger Fabric admin. For more information, see step 5.1 (p. 18) in the Getting Started (p. 6) tutorial.
   - The `--tls` flag specifies that communication with the ordering service uses TLS.
   - The `-C` flag specifies the channel on which to instantiate the chaincode.
   - The `-c` flag specifies the constructor message in JSON format that initializes object values for the sample chaincode that you installed on the peer node in the previous step. `a` is set to a value of `100` and `b` is set to a value of `200`.
   - The `-n` and `-v` options specify the name and version of the chaincode to be instantiated, which you established when installing the chaincode in the previous step.

   For more information about options, see peer chaincode instantiate in Hyperledger Fabric documentation.

```
cli peer chaincode instantiate \
-o orderer.n-MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.amazonaws.com:30001 \
-l java \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls \
-C my-hlf-channel-id \
-c '{"Args":["init", "a","100","b","200"]}' -v v0 -n javacc
```

You can verify that Gradle is using local dependencies by examining the build logs. The following example shows that, as expected, Gradle did not download jar files during build.

```
2020-06-26T07:42:37.371-07:00 Gradle build
2020-06-26T07:42:37.371-07:00 Starting a Gradle Daemon, 1 incompatible and 1 stopped
 Daemons could not be reused, use --status for details
2020-06-26T07:42:37.371-07:00 :compileJavaNote: /tmp/tmp.xANvpIqhV0/src/main/java/org/
hyperledger/fabric/example/SimpleChaincode.java uses or overrides a deprecated API.
2020-06-26T07:42:37.371-07:00 Note: Recompile with -Xlint:deprecation for details.
2020-06-26T07:42:37.371-07:00 :processResources NO-SOURCE
2020-06-26T07:42:37.371-07:00 :classes
2020-06-26T07:42:37.371-07:00 :jar
2020-06-26T07:42:37.371-07:00 :assemble
2020-06-26T07:42:37.371-07:00 :check
2020-06-26T07:42:37.371-07:00 :build
2020-06-26T07:42:37.371-07:00 :shadowJar
2020-06-26T07:42:37.371-07:00 Deprecated Gradle features were used in this build, making it
 incompatible with Gradle 5.0.
2020-06-26T07:42:37.371-07:00 See https://docs.gradle.org/4.6/userguide/
command_line_interface.html#sec:command_line_warnings
2020-06-26T07:42:37.371-07:00 BUILD SUCCESSFUL in 26s
2020-06-26T07:42:37.371-07:00 3 actionable tasks: 3 executed
```

If Gradle had downloaded the dependencies, the file would have references to plugin locations as shown in the following example.

```
2020-06-22T12:42:20.417-07:00 Gradle build
2020-06-22T12:42:20.417-07:00 Starting a Gradle Daemon, 1 incompatible and 1 stopped
 Daemons could not be reused, use --status for details
2020-06-22T12:42:20.417-07:00 Download https://plugins.gradle.org/m2/com/
github/johnrengelman/shadow/com.github.johnrengelman.shadow.gradle.plugin/2.0.4/
com.github.johnrengelman.shadow.gradle.plugin-2.0.4.pom
```

```
2020-06-22T12:42:20.417-07:00 Download https://plugins.gradle.org/m2/com/github/jengelman/
gradle/plugins/shadow/2.0.4/shadow-2.0.4.pom
2020-06-22T12:42:20.417-07:00 Download https://plugins.gradle.org/m2/org/codehaus/groovy/
groovy-backports-compat23/2.4.4/groovy-backports-compat23-2.4.4.pom
2020-06-22T12:42:20.417-07:00 Download https://plugins.gradle.org/m2/com/github/jengelman/
gradle/plugins/shadow/2.0.4/shadow-2.0.4.jar
2020-06-22T12:42:20.417-07:00 Download https://plugins.gradle.org/m2/org/codehaus/groovy/
groovy-backports-compat23/2.4.4/groovy-backports-compat23-2.4.4.jar
2020-06-22T12:42:20.417-07:00 Download https://hyperledger.jfrog.io/hyperledger/fabric-
maven/org/hyperledger/fabric-chaincode-java/fabric-chaincode-shim/maven-metadata.xml
2020-06-22T12:42:20.417-07:00 Download https://hyperledger.jfrog.io/hyperledger/fabric-
maven/org/hyperledger/fabric-chaincode-java/fabric-chaincode-shim/1.4.5/fabric-chaincode-
shim-1.4.5.pom
2020-06-22T12:42:20.417-07:00 Download https://jcenter.bintray.com/org/hyperledger/fabric-
chaincode-java/fabric-chaincode-shim/maven-metadata.xml
2020-06-22T12:42:20.417-07:00 Download https://jcenter.bintray.com/org/hyperledger/fabric-
chaincode-java/fabric-chaincode-shim/1.4.6/fabric-chaincode-shim-1.4.6.pom
2020-06-22T12:42:20.417-07:00 Download https://jcenter.bintray.com/com/owlike/genson/1.5/
genson-1.5.pom
2020-06-22T12:42:20.417-07:00 Download https://jcenter.bintray.com/com/owlike/genson-
parent/1.5/genson-parent-1.5.pom
2020-06-22T12:42:20.417-07:00 Download https://jcenter.bintray.com/org/hyperledger/fabric-
chaincode-java/fabric-chaincode-protos/1.4.6/fabric-chaincode-protos-1.4.6.pom
2020-06-22T12:42:20.417-07:00 Download https://jcenter.bintray.com/io/grpc/grpc-core/maven-
metadata.xml
2020-06-22T12:42:20.417-07:00 Download https://jcenter.bintray.com/io/netty/netty-codec-
http2/maven-metadata.xml
2020-06-22T12:42:20.417-07:00 Download https://jcenter.bintray.com/io/grpc/grpc-api/maven-
metadata.xml
2020-06-22T12:42:20.417-07:00 Download https://jcenter.bintray.com/com/google/errorprone/
error_prone_annotations/maven-metadata.xml
2020-06-22T12:42:20.417-07:00 Download https://jcenter.bintray.com/com/owlike/genson/1.5/
genson-1.5.jar
2020-06-22T12:42:20.417-07:00 Download https://jcenter.bintray.com/org/hyperledger/fabric-
chaincode-java/fabric-chaincode-shim/1.4.6/fabric-chaincode-shim-1.4.6.jar
2020-06-22T12:42:20.417-07:00 Download https://jcenter.bintray.com/org/hyperledger/fabric-
chaincode-java/fabric-chaincode-protos/1.4.6/fabric-chaincode-protos-1.4.6.jar
...
2020-06-22T12:42:20.417-07:00 :compileJava
2020-06-22T12:42:20.417-07:00 :processResources NO-SOURCE
2020-06-22T12:42:20.417-07:00 :classes
2020-06-22T12:42:20.417-07:00 :jar
2020-06-22T12:42:20.417-07:00 :assemble
2020-06-22T12:42:20.417-07:00 :check
2020-06-22T12:42:20.417-07:00 :build
2020-06-22T12:42:20.417-07:00 :shadowJar
2020-06-22T12:42:20.417-07:00 BUILD SUCCESSFUL in 41s
2020-06-22T12:42:20.417-07:00 3 actionable tasks: 3 executed
```

Step 7: Invoke the sample chaincode to perform a transaction

Precede the following example command with the variable overrides for the CLI container as shown in the previous example.

The following example invokes the sample chaincode that you instantiated. The parameters are the same as the previous example, except for the JSON constructor. This constructor invokes the the sample chaincode to transfer a value of 10 from `a` to `b`.

```
cli peer chaincode invoke \
-o orderer.n-MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.amazonaws.com:30001 \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls \
-C one-org-channel-java \
-c '{"Args":["invoke","a", "b", "10"]}' -n javacc
```

The command should print output similar to the following.

```
2020-06-26 23:49:43.895 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 003 Chaincode
 invoke successful. result: status:200 message:"invoke finished successfully" payload:"a:
 90 b: 210
```

Step 8: Invoke the sample chaincode to query

Precede the following command with the variable overrides for the CLI container as shown in the previous example.

The following example queries the ledger for the value attributed to `a`. The query should print `90` as output.

```
cli peer chaincode query \
-o orderer.n-MWY63ZJZU5HGNCMBQER7IN6OIU.managedblockchain.amazonaws.com:30001 \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls \
-C one-org-channel-java \
-c '{"Args":["query","a"]}' -n javacc
```

Query Chaincode Data in the State Database

The state database in Hyperledger Fabric peer nodes on Managed Blockchain networks created using Hyperledger Fabric 1.4 and later supports two types of peer state databases:

- **CouchDB** is a state database in Managed Blockchain that models ledger data as JSON. It is the default peer state database for Hyperledger Fabric 1.4 or later network on Managed Blockchain because CouchDB supports rich queries and indexing for more efficient queries over large datasets.
- **LevelDB** is a state database that stores ledger data as simple key-value pairs. It is the only peer state database available for Hyperledger Fabric 1.2 networks on Managed Blockchain. LevelDB supports only key, key range, and composite key queries.

For detailed information about CouchDB implementation and capabilities in Hyperledger Fabric, see CouchDB as the State Database in Hyperledger Fabric documentation.

Specifying and Viewing the State Database Type

You specify the peer database type when you create a peer node for a member using the AWS Management Console, the AWS CLI, or the Managed Blockchain API. For more information, see Create a Hyperledger Fabric Peer Node on Amazon Managed Blockchain (p. 45) and CreateNode in the *Amazon Managed Blockchain API Reference*. You can't change the state database type after a node has been created.

The type of state database that a peer node uses is available on the peer node properties page in the AWS Management Console. It is also returned by the AWS CLI `managedblockchain get-node` command. For more information, see View Hyperledger Fabric Peer Node Properties on Amazon Managed Blockchain (p. 46).

Rich Queries With CouchDB

The following examples demonstrate rich queries against a CouchDB state database using the `peer chaincode query` command of the Hyperledger Fabric CLI in Managed Blockchain. The examples are based on the marbles02 chaincode sample on GitHub. This example includes implementations of

rich query functionality within chaincode, including the code that supports the queryMarbles and the queryMarblesWithPagination functions passed as arguments in the examples.

The marbles02 sample chaincode is available when you set up a Hyperledger Fabric client (p. 12) according to the instructions in the getting started tutorial of this guide. Those instructions include a step to clone the samples repository (p. 16) that contains marbles02.

For examples of basic queries that can be run when using either LevelDB or CouchDB, see the query chaincode (p. 24) example in the getting started tutorial of this guide.

Before querying the chaincode, `install` the chaincode, `instantiate` it, and `invoke` the chaincode to add three ledger records as shown in the following example commands. For more information, see the peer chaincode command reference in Hyperledger Fabric documentation.

```
docker exec cli peer chaincode install \
-n myrichquerycc -v v0 -p github.com/marbles02/go
```

```
docker exec cli peer chaincode instantiate \
-o $ORDERER -C mychannel -n myrichquerycc \
-v v0 -c '{"Args":["init"]}' \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

```
docker exec cli peer chaincode invoke \
-o $ORDERER -C mychannel -n myrichquerycc \
-c '{"Args":["initMarble","marble1","blue","10","tom"]}' \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

```
docker exec cli peer chaincode invoke \
-o $ORDERER -C mychannel -n myrichquerycc \
-c '{"Args":["initMarble","marble2","red","20","tom"]}' \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

```
docker exec cli peer chaincode invoke \
-o $ORDERER -C mychannel -n myrichquerycc \
-c '{"Args":["initMarble","marble3","green","30","tom"]}' \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

Now that the state database is populated with three records, use the rich query function `queryMarbles` defined within the marbles02 chaincode sample as shown in the following example. The query returns only those marbles owned by `tom`.

```
docker exec cli peer chaincode query -o $ORDERER \
-C mychannel -n myrichquerycc \
-c '{"Args":["queryMarbles","{\"selector\":{\"owner\":\"tom\"}}"]}' \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

The query example below uses the rich query function `queryMarblesWithPagination` defined within the marbles02 chaincode sample. The query is similar to the previous rich query. However, this function allows you to specify the number of records to return (the page size) and an optional bookmark. The page size specified in the example is `1` and the bookmark argument is empty.

```
docker exec cli peer chaincode query \
-o $ORDERER -C mychannel -n myrichquerycc \
-c '{"Args":["queryMarblesWithPagination","{\"selector\":{\"owner\":\"tom\"}}","1",""]}' \
--cafile /opt/home/managedblockchain-tls-chain.pem --tls
```

Hyperledger Fabric on Amazon Managed Blockchain Security

Amazon Managed Blockchain utilizes AWS features and the features of the open-source framework running on Managed Blockchain to provide data protection as well as authentication and access control.

This chapter covers security information specific to Hyperledger Fabric on Managed Blockchain. For security information specific to Ethereum on Managed Blockchain, see Ethereum on Managed Blockchain Security in the *Ethereum on Amazon Managed Blockchain Developer Guide*.

Topics

Data Protection for Hyperledger Fabric on Amazon Managed Blockchain

The AWS shared responsibility model applies to data protection in Hyperledger Fabric on Amazon Managed Blockchain. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a **Name** field. This includes when you work with Hyperledger Fabric on Managed Blockchain or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data Encryption for Hyperledger Fabric on Managed Blockchain

Data encryption helps prevent unauthorized users from reading data from a blockchain network and the associated data storage systems. This includes data saved to persistent media, known as *data at rest*, and data that may be intercepted as it travels the network, known as *data in transit*.

Topics

Encryption at Rest for Hyperledger Fabric on Managed Blockchain

Amazon Managed Blockchain offers fully managed encryption at rest. Managed Blockchain *encryption at rest* provides enhanced security by encrypting all data at rest on networks, members, and peer nodes using keys in AWS Key Management Service (AWS KMS). This functionality helps reduce the operational burden and complexity involved in protecting sensitive data. With encryption at rest, you can build security-sensitive blockchain applications that meet strict encryption compliance and regulatory requirements.

Encryption at rest integrates with AWS KMS for managing the encryption key that is used to protect your Hyperledger Fabric data. For more information about AWS KMS, see AWS Key Management Service concepts in the *AWS Key Management Service Developer Guide*.

In Hyperledger Fabric on Managed Blockchain, you can specify the type of AWS KMS key at the member level. This encryption key is then inherited by any peer nodes that the member creates. Data at the network level is encrypted using AWS owned keys by default at no additional cost.

When you create a new member, you can choose one of the following types of KMS keys to protect your member and its peer nodes:

- *AWS owned key* – The default encryption type, if no KMS key ARN is specified. The key is owned by Managed Blockchain (no additional charge, and no configuration required).
- *Customer managed key* – The key is stored in your AWS account and is created, owned, and managed by you. You have full control over the key (AWS KMS charges apply).

When you access networks, members, and peer nodes, Managed Blockchain decrypts the data transparently. You don't have to change any code or applications to use or manage encrypted data. All Managed Blockchain operations work seamlessly on your encrypted data.

You can specify a KMS key when you create a new member by using the AWS Management Console, the Managed Blockchain API, or the AWS Command Line Interface (AWS CLI). For more information, see the MemberConfiguration data type in the *Amazon Managed Blockchain API Reference*. You can pass this data type as an input parameter to the CreateMember or CreateNetwork API operations.

> **Note**
> By default, Amazon Managed Blockchain automatically enables encryption at rest using AWS owned keys at no additional charge. However, AWS KMS charges apply for using a customer managed key. For information about pricing, see AWS Key Management Service pricing.

Topics

Encryption at Rest: How It Works

Hyperledger Fabric on Managed Blockchain encryption at rest encrypts your data using 256-bit Advanced Encryption Standard (AES-256), which helps secure your data from unauthorized access to the underlying storage.

Encryption at rest integrates with AWS Key Management Service (AWS KMS) for managing the encryption key that is used to protect your Hyperledger Fabric data. When creating a new member, you can choose one of the following types of AWS KMS keys:

AWS owned key

AWS owned keys aren't stored in your AWS account. They are part of a collection of KMS keys that AWS owns and manages for use in multiple AWS accounts. AWS services can use AWS owned keys to protect your data.

You don't need to create or manage AWS owned keys. However, you can't view, use, track, or audit them. You aren't charged a monthly fee or a usage fee for AWS owned keys, and they don't count against the AWS KMS quotas for your account.

For more information, see AWS owned keys in the *AWS Key Management Service Developer Guide*.

Customer managed key

Customer managed keys are KMS keys in your AWS account that you create, own, and manage. You have full control over these KMS keys. Managed Blockchain supports symmetric customer managed keys only.

Use a customer managed key to get the following features:

- Setting and maintaining key policies, IAM policies, and grants to control access to the key
- Enabling and disabling the key
- Rotating cryptographic material for the key
- Creating key tags and aliases
- Scheduling the key for deletion
- Importing your own key material or using a custom key store that you own and manage
- Using AWS CloudTrail and Amazon CloudWatch Logs to track the requests that Managed Blockchain sends to AWS KMS on your behalf

For more information, see Customer managed keys in the *AWS Key Management Service Developer Guide*.

Customer managed keys incur a charge for each API call, and AWS KMS quotas apply to these KMS keys. For more information, see AWS KMS resource or request quotas.

When you specify a customer managed key as the member-level KMS key, the member and its peer nodes are protected with the same customer managed key. Network-level data is encrypted using AWS owned keys by default at no additional cost.

Inaccessible Customer Managed Keys

If you disable your customer managed key, schedule the key for deletion, or revoke the grants on the key, the status of your member and its peer nodes becomes `INACCESSIBLE_ENCRYPTION_KEY`. In this state, the member and its peer nodes are impaired and might not function as expected. An inaccessible key prevents all users and the Managed Blockchain service from encrypting or decrypting data—and from performing read and write operations on the nodes. Managed Blockchain must have access to your KMS key to ensure that you can continue to access your nodes and to prevent data loss.

The effect of disabling or deleting a key or of revoking a grant is not immediate. It might take some time for the member and peer node resources to discover that the key is inaccessible. When a resource is in this state, we recommend deleting and recreating the resource, and reconfiguring the KMS key. To check the encryption status of a member or peer node resource, use the GetMember or GetNode API operation respectively.

How Managed Blockchain Uses Grants in AWS KMS

Managed Blockchain requires *grants* to use your customer managed key. When you create a member that is protected with a customer managed key, Managed Blockchain creates grants on your behalf by sending CreateGrant requests to AWS KMS. Grants in AWS KMS are used to give Managed Blockchain access to a KMS key in a customer AWS account. For more information, see Using Grants in the *AWS Key Management Service Developer Guide*.

Managed Blockchain requires the grants to use your customer managed key for the following API operations:

- Decrypt
- DescribeKey
- Encrypt
- GenerateDataKey
- GenerateDataKeyWithoutPlaintext
- ReEncryptFrom
- ReEncryptTo

Managed Blockchain uses Amazon Elastic Compute Cloud (Amazon EC2), Amazon Elastic Block Store (Amazon EBS), and AWS Secrets Manager to call these AWS KMS operations on its behalf.

You can revoke a grant to remove the service's access to the customer managed key at any time. If you do, the key becomes inaccessible, and Managed Blockchain loses access to any of the data protected by the customer managed key. In this state, the member and its peer node resources are impaired and might not function as expected. When a resource is in this state, we recommend deleting and recreating the resource, and reconfiguring the KMS key.

Encryption at Rest Considerations

Consider the following when you are using encryption at rest in Hyperledger Fabric on Managed Blockchain.

- Server-side encryption at rest is enabled on all Managed Blockchain network, member, and peer node data and can't be disabled. You can't encrypt only a subset of data in these resource types.
- Encryption at rest only encrypts data while it is static (at rest) on a persistent storage media. If data security is a concern for data in transit or data in use, you might need to take additional measures as follows:
  - *Data in transit*: All your data in Managed Blockchain is encrypted in transit. By default, communications to and from Managed Blockchain use the HTTPS protocol, which protects network traffic by using Secure Sockets Layer (SSL)/Transport Layer Security (TLS) encryption.

- *Data in use*: Use client-side encryption to protect your data before sending it to Managed Blockchain.
- Managed Blockchain currently doesn't support encryption context for AWS KMS cryptographic operations.

Using Customer Managed Keys in Managed Blockchain

You can use the Managed Blockchain console, the Managed Blockchain API, or the AWS CLI to specify the AWS KMS key for new members in Managed Blockchain. The following topics describe how to manage and monitor the usage of your customer managed keys in Managed Blockchain.

Topics

Prerequisites

Before you can protect Managed Blockchain resources with a customer managed key, you must first create the key in AWS KMS. You must also specify a key policy that allows Managed Blockchain to create grants on that AWS KMS key on your behalf.

Creating a Customer Managed Key

To create a customer managed key, follow the steps in Creating symmetric KMS keys in the *AWS Key Management Service Developer Guide*. Managed Blockchain doesn't support asymmetric keys.

Setting a Key Policy

Key policies are the primary way to control access to customer managed keys in AWS KMS. Every customer managed key must have exactly one key policy. The statements in the key policy document determine who has permission to use the AWS KMS key and how they can use it. For more information, see Using key policies in AWS KMS in the *AWS Key Management Service Developer Guide*.

You can specify a key policy when you create your customer managed key. To change a key policy for an existing customer managed key, see Changing a key policy.

To allow Managed Blockchain to use your customer managed key, the key policy must include permissions for the CreateGrant API operation. This operation adds a grant to a customer managed key. Grants control access to a specified KMS key. Managed Blockchain creates grants that allow access to the grant operations that it requires, which are listed in How Managed Blockchain Uses Grants in AWS KMS (p. 86).

Key Policy Example

The following is a key policy example that you can use for Managed Blockchain. This policy allows principals that are authorized to use Managed Blockchain from the account `111122223333` to call the `CreateGrant` operation on all resources.

To use this policy, replace *`aws-region`* and *`111122223333`* in the example with your own information.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Sid" : "Allow access to principals authorized to use Amazon Managed Blockchain",
      "Effect" : "Allow",
      "Principal" : {
        "AWS" : "*"
      },
      "Action" : [
        "kms:CreateGrant"
      ],
      "Resource" : "*",
      "Condition" : {
        "StringEquals" : {
          "kms:ViaService" : "managedblockchain.aws-region.amazonaws.com",
          "kms:CallerAccount" : "111122223333"
        }
      }
    }
  ]
}
```

Specifying a Customer Managed Key

You can specify the Amazon Resource Name (ARN) of a customer managed KMS key when you create a new member by using the Managed Blockchain console, the Managed Blockchain API, or the AWS CLI. The following is an example of a KMS key ARN.

```
arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

For more information, see the MemberConfiguration data type in the *Amazon Managed Blockchain API Reference*. You can pass this data type as an input parameter to the CreateMember or CreateNetwork API operations.

Monitoring Your Customer Managed Keys

If you use a customer managed key to protect your Amazon Managed Blockchain resources, you can use AWS CloudTrail or Amazon CloudWatch Logs to track the requests that Managed Blockchain sends to AWS KMS on your behalf. For more information, see Monitoring AWS KMS keys in the *AWS Key Management Service Developer Guide*.

The following example is a CloudTrail log entry for the `CreateGrant` API operation. In addition to this event from Managed Blockchain, you can expect AWS KMS calls for your customer managed key from Amazon EC2, Amazon EBS, and AWS Secrets Manager. These AWS services call other AWS KMS operations such as `GenerateDataKey`, `Decrypt`, and `Encrypt` on behalf of Managed Blockchain.

CreateGrant

When you specify a customer managed key to protect your member, Managed Blockchain sends `CreateGrant` requests to AWS KMS on your behalf to allow access to your KMS key.

The grants that Managed Blockchain creates are specific to a member and its peer nodes. The principal in the `CreateGrant` request is the user who created the member or peer node.

The event that records the `CreateGrant` operation is similar to the following example event. The parameters include the Amazon Resource Name (ARN) of the customer managed key, the grantee principal and retiring principal (the Managed Blockchain service), and the operations that the grant covers.

```
{
    "eventVersion": "1.08",
```

```
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/sample-user",
        "accountId": "111122223333",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "sample-user",
        "sessionContext": {
            "sessionIssuer": {},
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-06-08T16:36:03Z"
            }
        },
        "invokedBy": "managedblockchain.amazonaws.com"
    },
    "eventTime": "2021-06-08T16:36:04Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "managedblockchain.amazonaws.com",
    "userAgent": "managedblockchain.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "granteePrincipal": "managedblockchain.us-east-1.amazonaws.com",
        "retiringPrincipal": "managedblockchain.us-east-1.amazonaws.com",
        "operations": [
            "Decrypt"
        ],
        "constraints": {
            "encryptionContextSubset": {}
        }
    },
    "responseElements": {
        "grantId": "64690d7d6ad23d39edd0c8fdea6400f297fae673ad84ce3563b6ca3a1e685ba2"
    },
    "requestID": "55b9f158-f537-43c2-acec-f336f41866b1",
    "eventID": "944aa59c-3bf9-4092-9ca2-435f585b41f6",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333"
}
```


Encryption in Transit for Hyperledger Fabric on Managed Blockchain

The Hyperledger Fabric certificate authority (CA) in each membership provides a TLS certificate authority to secure Hyperledger Fabric communication channels in the Amazon Managed Blockchain network. For more information, see the Fabric CA's User Guide in Hyperledger Fabric documentation.

Authentication and Access Control for Hyperledger Fabric on Managed Blockchain

AWS Identity and Access Management (IAM) permissions policies, VPC endpoint services powered by AWS PrivateLink, and Amazon EC2 security groups provide the primary means for you to control access to Amazon Managed Blockchain. In addition to these AWS services, open-source frameworks that run on Managed Blockchain have authentication and access control features that you can configure.

IAM permissions policies are associated with AWS users in your account and determine who has access to what. Permissions policies specify the actions that each user can perform using Managed Blockchain and other AWS services. VPC endpoint services allow each Managed Blockchain network member to connect privately to Managed Blockchain resources. Amazon EC2 security groups act as virtual firewalls and determine the inbound and outbound network traffic that is allowed between Managed Blockchain resources and other Amazon EC2 resources. In Managed Blockchain, these security groups are associated with the VPC endpoint in your account and with any framework clients that run on AWS, such as a Hyperledger Fabric client running on an Amazon EC2 instance.

Before you configure authentication and access control using AWS services and open-source features, we recommend that you review the following resources:

- For more information about IAM and IAM permissions policies, see Identity and Access Management for Hyperledger Fabric on Amazon Managed Blockchain (p. 90). We also recommend What is IAM? and IAM JSON Policy Reference in the *IAM User Guide*.
- For more information about VPC endpoints, see Create an Interface VPC Endpoint for Hyperledger Fabric on Amazon Managed Blockchain (p. 43) and VPC Endpoints in the *Amazon VPC User Guide*.
- For more information about Amazon EC2 security groups, see Configuring Security Groups for Hyperledger Fabric on Amazon Managed Blockchain (p. 111) and Amazon EC2 Security Groups for Linux Instances in the *Amazon EC2 User Guide for Linux Instances*.
- For more information about the Hyperledger Fabric Certificate Authority (CA), see Certificate Authority (CA) Setup in the Hyperledger Fabric documentation.
- For more information about Hyperledger Fabric application access control lists, see Application Access Control Lists in the Hyperledger Fabric documentation.

Identity and Access Management for Hyperledger Fabric on Amazon Managed Blockchain

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Managed Blockchain resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Managed Blockchain.

Service user – If you use the Managed Blockchain service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Managed Blockchain features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Managed Blockchain, see Troubleshooting Hyperledger Fabric on Amazon Managed Blockchain identity and access (p. 110).

Service administrator – If you're in charge of Managed Blockchain resources at your company, you probably have full access to Managed Blockchain. It's your job to determine which Managed Blockchain features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Managed Blockchain, see How Hyperledger Fabric on Amazon Managed Blockchain works with IAM (p. 94).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Managed Blockchain. To view example Managed Blockchain identity-based policies that you can use in IAM, see Hyperledger Fabric on Amazon Managed Blockchain Identity-Based Policy Examples (p. 97).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for Amazon Managed Blockchain in the *Service Authorization Reference*.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear

in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access Control Lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Hyperledger Fabric on Amazon Managed Blockchain works with IAM

Before you use IAM to manage access to Hyperledger Fabric on Managed Blockchain, you should understand what IAM features are available to use with Hyperledger Fabric on Managed Blockchain. To get a high-level view of how Hyperledger Fabric on Managed Blockchain and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

Topics

Hyperledger Fabric on Managed Blockchain identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Hyperledger Fabric on Managed Blockchain supports specific actions and resources. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in Hyperledger Fabric on Managed Blockchain use the following prefix before the action: `managedblockchain:`. For example, to grant someone permission to create a node with the Managed Blockchain `CreateNode` API operation, you include the `managedblockchain:CreateNode` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Hyperledger Fabric on Managed Blockchain defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows.

```
"Action": [
      "managedblockchain:action1",
      "managedblockchain:action2"
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `List`, include the following action.

```
"Action": "managedblockchain:List*"
```

To see a list of Managed Blockchain actions, see Actions Defined by Amazon Managed Blockchain in the *IAM User Guide*.

Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

Managed Blockchain resource types that can be used in IAM permissions policy statements for resources on Ethereum networks include the following:

- network
- member
- node
- proposal
- invitation

Members, nodes, and invitations are associated with your account. Networks and proposals, on the other hand, are scoped to the entire blockchain network and are not associated with a particular account.

For example a network resource on Managed Blockchain has the following ARN.

```
arn:${Partition}:managedblockchain:${Region}::networks/${NetworkId}
```

For example, to specify the `n-MWY63ZJZU5HGNCMBQER7IN6OIU` network in your statement, use the following ARN.

```
"Resource": "arn:aws:managedblockchain:us-east-1::networks/n-MWY63ZJZU5HGNCMBQER7IN6OIU"
```

To specify any network that is visible to your account, use the wildcard (*).

```
"Resource": "arn:aws:managedblockchain:us-east-1::networks/*"
```

Some Managed Blockchain actions, such as `CreateNetwork`, `ListInvitations`, and `ListNetworks` cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

To see a list of Hyperledger Fabric on Managed Blockchain resource types and their ARNs, see Resources Defined by Amazon Managed Blockchain in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon Managed Blockchain.

Condition keys

Hyperledger Fabric on Managed Blockchain does not provide any service-specific condition keys, but it does support using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*.

Examples

To view examples of Hyperledger Fabric on Managed Blockchain identity-based policies, see Hyperledger Fabric on Amazon Managed Blockchain Identity-Based Policy Examples (p. 97).

Hyperledger Fabric on Managed Blockchain Resource-Based Policies

Hyperledger Fabric on Managed Blockchain does not support resource-based policies.

Authorization based on Hyperledger Fabric on Managed Blockchain tags

You can attach tags to Hyperledger Fabric on Managed Blockchain resources or pass tags in a request to Managed Blockchain. To control access based on tags, you provide tag information in the condition element of a policy using the `managedblockchain:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, or `aws:TagKeys` condition keys. For more information about tagging Hyperledger Fabric on Managed Blockchain resources, see Tagging Amazon Managed Blockchain resources (p. 113).

To view example identity-based policies for allowing or denying access to resources and actions based on tags, see Controlling access using tags (p. 99).

Hyperledger Fabric on Amazon Managed Blockchain Identity-Based Policy Examples

By default, IAM users and roles don't have permission to create or modify Managed Blockchain resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the *IAM User Guide*.

Topics

Policy Best Practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Managed Blockchain resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started using AWS managed policies** – To start using Managed Blockchain quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Allow Users to View Their Own Permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Using the Managed Blockchain Console

To access the Amazon Managed Blockchain console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Managed Blockchain resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the Managed Blockchain console, also attach the following AWS managed policy to the entities.

```
AmazonManagedBlockchainConsoleFullAccess
```

For more information, see Adding Permissions to a User in the *IAM User Guide*.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Performing All Managed Blockchain Actions on All Accessible Networks for an AWS Account

This example grants an IAM user in your AWS account access to all network and member resources in your account in the `us-east-1` Region for the AWS account `123456789012`. This includes the ability to create new networks, reject invitations to join other networks, and join other networks by creating a member.

```
{
 "Version": "2012-10-17",
 "Statement": [{
   "Sid": "ManageNetworkResources",
```

```
   "Effect": "Allow",
   "Action": [
    "managedblockchain:CreateProposal",
    "managedblockchain:GetProposal",
    "managedblockchain:DeleteMember",
    "managedblockchain:VoteOnProposal",
    "managedblockchain:ListProposals",
    "managedblockchain:GetNetwork",
    "managedblockchain:ListMembers",
    "managedblockchain:ListProposalVotes",
    "managedblockchain:RejectInvitation",
    "managedblockchain:GetNode",
    "managedblockchain:GetMember",
    "managedblockchain:DeleteNode",
    "managedblockchain:CreateNode",
    "managedblockchain:CreateMember",
    "managedblockchain:ListNodes"
   ],
   "Resource": [
    "arn:aws:managedblockchain:us-east-1::networks/*",
    "arn:aws:managedblockchain:us-east-1::proposals/*",
    "arn:aws:managedblockchain:us-east-1:123456789012:members/*",
    "arn:aws:managedblockchain:us-east-1:123456789012:invitations/*",
    "arn:aws:managedblockchain:us-east-1:123456789012:nodes/*"
   ]
  },
  {
   "Sid": "WorkWithNetworksForAcct",
   "Effect": "Allow",
   "Action": [
    "managedblockchain:ListNetworks",
    "managedblockchain:ListInvitations",
    "managedblockchain:CreateNetwork"
   ],
   "Resource": "*"
  }
 ]
}
```

Controlling access using tags

The following example policy statements demonstrate how you can use tags to limit access to Hyperledger Fabric on Managed Blockchain resources and actions performed on those resources.

To tag resources during creation, policy statements that allow the create action for the resource as well as the `TagResource` action must be attached to the IAM principal performing the operation.

> **Note**
> This topic includes examples of policy statements with a `Deny` effect. Each policy statement assumes that a statement with a broader `Allow` effect for the same actions exists; the `Deny` policy statement restricts that otherwise overly-permissive allow statement.

Example – Require a tag key with either of two values to be added during network creation

The following identity-based policy statements allow the IAM principal to create a network only if the network is created with the tags specified.

The statement with the `Sid` set to `RequireTag` specifies that network creation is allowed only if the network is created with a tag that has a tag key of `department` and a value of either `sales` or `marketing`; otherwise, the create network operation fails.

The statement with `Sid` set to `AllowTagging` allows the IAM principal to tag networks, which are not associated with an AWS account ID. It also allows tagging of members in the specified AWS account, `111122223333`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RequireTag",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:CreateNetwork"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "aws:RequestTag/department": [
                        "sales",
                        "marketing"
                    ]
                }
            }
        },
        {
            "Sid": "AllowTagging",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:TagResource"
            ],
            "Resource": [
                "arn:aws:managedblockchain:us-east-1::networks/*",
                "arn:aws:managedblockchain:us-east-1:111122223333:members/*"
            ]
        }
    ]
}
```

Example – Deny access to networks that have a specific tag key

The following identity-based policy statement denies the IAM principal the ability to retrieve or view information for networks that have a tag with a tag key of `restricted`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyRestrictedNetwork",
            "Effect": "Deny",
            "Action": [
                "managedblockchain:GetNetwork"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/restricted": [
                        "*"
                    ]
                }
            }
        }
    ]
}
```

Example – Deny member creation for networks with a specific tag key and value

The following identity-based policy statement denies the IAM principal from creating a member on any network that has a tag with a tag key of `department` with the value of `accounting`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyMemberCreation",
            "Effect": "Deny",
            "Action": [
                "managedblockchain:CreateMember"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/department": "accounting"
                }
            }
        }
    ]
}
```

Example – Allow member creation only if a specific tag key and value are added

The following identity-based policy statements allow the IAM principal to create members in the account `111122223333` only if the member is created with a tag that has the tag key of `department` and a tag value of `accounting`; otherwise, the create member operation fails.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RestrictMemberCreation",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:CreateMember"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/department": "accounting"
                }
            }
        },
        {
            "Sid": "AllowTaggingOfMembers",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:TagResource"
            ],
            "Resource": [
                "arn:aws:managedblockchain:us-east-1:111122223333:members/*"
            ]
        }
    ]
}
```

Example – Allow access only to members that have a specified tag key

The following identity-based policy statement allows the IAM principal to retrieve or view information about members only if the member has a tag with a tag key of `unrestricted`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:GetMember"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/unrestricted": [
                        "*"
                    ]
                }
            }
        }
    ]
}
```

Example – Deny node creation if a specific tag key and value are added during creation

The following identity-based policy statement denies the IAM principal the ability to create a node if a tag with the tag key of `department` and a value of `analytics` is added during creation.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyCreateNodeWithTag",
            "Effect": "Deny",
            "Action": [
                "managedblockchain:CreateNode"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/department": [
                        "analytics"
                    ]
                }
            }
        }
    ]
}
```

Example – Deny node creation for members with a specific tag key and value

The following identity-based policy statement denies the IAM principal the ability to create a node if the member to which the node will belong has a tag with a tag key of `department` and a tag value of `analytics`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyCreateNodeForTaggedMember",
            "Effect": "Deny",
            "Action": [
                "managedblockchain:CreateNode"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/department": [
                        "analytics"
                    ]
                }
            }
        }
    ]
}
```

Example – Allow access only to nodes with a specific tag key and value

The following identity-based policy statement allows the IAM principal to retrieve or view information about nodes only if the node has a tag with a tag key of `department` and a tag value of `sales`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowTaggedNodeAccess",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:GetNode"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/department": [
                        "sales"
                    ]
                }
            }
        }
    ]
}
```

Example – Deny access to nodes with a specific tag key

The following identity-based policy statement denies the IAM principal the ability to retrieve or view information about a node if the node has a tag with a tag key of `restricted`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyAccessToTaggedNodes",
            "Effect": "Deny",
```

```
            "Action": [
                "managedblockchain:GetNode"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/restricted": [
                        "*"
                    ]
                }
            }
        }
    ]
}
```

Example – Allow proposal creation only for networks with a specific tag key and value

The following identity-based policy statement allows the IAM principal to create a proposal only if the network for which the proposal is made has a tag with a tag key of `department` and a value of `accounting`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreateProposalOnlyForTaggedNetwork",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:CreateProposal"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/department": [
                        "accounting"
                    ]
                }
            }
        }
    ]
}
```

Example – Allow proposal creation only if a specific tag key and value are added

The following identity-based policy statements allow the IAM principal to create a proposal for inviting the specified account `123456789012` to the network only if a tag with a tag key of `consortium` and a tag value of `exampleconsortium` is added during proposal creation.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RequireTagWithProposal",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:CreateProposal"
            ],
            "Resource": [
```

```
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/consortium": [
                        "exampleconsortium"
                    ]
                }
            }
        },
        {
            "Sid": "AllowProposalAndInvitationTagging",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:TagResource"
            ],
            "Resource": [
                "arn:aws:managedblockchain:us-east-1::proposals/*",
                "arn:aws:managedblockchain:us-east-1:123456789012:invitations/*"
            ]
        }
    ]
}
```

Example – Allow access only to proposals with a specific tag key and value

The following identity-based policy statement allows the IAM principal to retrieve and view proposals only if the proposal has a tag with a tag key of `consortium` and a tag value of `exampleconsortium`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowOnlyTaggedProposalAccess",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:GetProposal"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/consortium": [
                        "exampleconsortium"
                    ]
                }
            }
        }
    ]
}
```

Example – Deny the ability to vote on proposals based on a specific tag key and value

The following identity-based policy statement denies the IAM principal the ability to vote on proposals that have a tag with the tag key of `consortium` and a tag value of `exampleconsortium`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyVoteOnTaggedProposal",
            "Effect": "Deny",
```

```
            "Action": [
                "managedblockchain:VoteOnProposal"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/consortium": [
                        "exampleconsortium"
                    ]
                }
            }
        }
    ]
}
```

Example – Deny the ability to reject an invitation with a specific tag and value

The following identity-based policy statement denies the IAM principal the ability to reject on invitation that has a tag with the tag key of `consortium` and a tag value of `exampleconsortium`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "1",
            "Effect": "Deny",
            "Action": [
                "managedblockchain:RejectInvitation"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/consortium": [
                        "exampleconsortium"
                    ]
                }
            }
        }
    ]
}
```

Example IAM Role Permissions Policy for Hyperledger Fabric Client EC2 Instance

When you administer, develop, and deploy chaincode using an EC2 instance as a Hyperledger Fabric client, the permissions policies attached to the AWS Identity and Access Management *instance profile* and *instance role* associated with the instance determine its permissions to interact with other AWS resources, including Managed Blockchain. The permissions policy shown in the following procedure provides sufficient privileges when it is attached to the IAM role of the instance.

The procedure demonstrates how to create a role with only this permissions policy attached and then attach that role to an EC2 instance. If you have an existing service role and instance profile attached to your EC2 instance, you can create an additional policy using the following example and then attach it to the existing role. Only one role can be attached to an EC2 instance.

For more information, see IAM roles for Amazon EC2 in the *Amazon EC2 User Guide for Linux Instances*.

To create a permissions policy, attach it to an IAM role, and attach the role to a Hyperledger Fabric client EC2 instance

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**, and then **Create policy**.
3. Choose the `JSON` tab, and then copy and paste the following policy statement.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ListNetworkMembers",
      "Effect": "Allow",
      "Action": [
          "managedblockchain:GetNetwork",
          "managedblockchain:ListMembers"
      ],
      "Resource": [
          "arn:aws:managedblockchain:*:123456789012:networks/*"
      ]
    },
    {
      "Sid": "AccessManagedBlockchainBucket",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::us-east-1.managedblockchain/*"
    },
    {
      "Sid": "ManageNetworkResources",
      "Effect": "Allow",
      "Action": [
        "managedblockchain:CreateProposal",
        "managedblockchain:GetProposal",
        "managedblockchain:DeleteMember",
        "managedblockchain:VoteOnProposal",
        "managedblockchain:ListProposals",
        "managedblockchain:GetNetwork",
        "managedblockchain:ListMembers",
        "managedblockchain:ListProposalVotes",
        "managedblockchain:RejectInvitation",
        "managedblockchain:GetNode",
        "managedblockchain:GetMember",
        "managedblockchain:DeleteNode",
        "managedblockchain:CreateNode",
        "managedblockchain:CreateMember",
        "managedblockchain:ListNodes"
      ],
      "Resource": [
        "arn:aws:managedblockchain:*::networks/*",
        "arn:aws:managedblockchain:*::proposals/*",
        "arn:aws:managedblockchain:*:123456789012:members/*",
        "arn:aws:managedblockchain:*:123456789012:invitations/*",
        "arn:aws:managedblockchain:*:123456789012:nodes/*"
      ]
    },
    {
      "Sid": "WorkWithNetworksForAcct",
      "Effect": "Allow",
      "Action": [
        "managedblockchain:ListNetworks",
        "managedblockchain:ListInvitations",
        "managedblockchain:CreateNetwork"
```

```
      ],
      "Resource": "*"
    }
  ]
}
```

4. Replace *123456789012* with your AWS Account ID. If you are working in a different Region, replace *us-east-1* with the appropriate Region, and then choose **Review policy**.
5. Enter a **Name** for the policy, for example, `HyperledgerFabricClientAccess`. Enter an optional **Description**, and then choose **Create policy**.

   You now have a permissions policy that you can attach to an IAM role for an EC2 instance.
6. From the navigation pane, choose **Roles**, **Create role**.
7. Under **Select type of trusted entity**, leave **AWS service** selected. Under **Common use cases**, choose **EC2 - Allows EC2 instances to call AWS services on your behalf**, and then choose **Next: Permissions**.
8. Under **Attach permissions policies**, start typing the name of the permissions policy you created in the previous steps. Select that policy from the list, and then choose **Next: Tags**.
9. Leave the key-value fields blank or type key-value pairs for any tags that you want to apply to this role, and then choose **Next: Review**.
10. Enter a **Role name** that helps you identify this role, for example, **`ServiceRoleForHyperledgerFabricClient`**. Enter an optional **Description**, and then choose **Create role**.

    You now have a service role with appropriate permissions that you can associate with your Hyperledger Fabric EC2 instance.
11. Open the EC2 console at https://console.aws.amazon.com/ec2/.
12. From the navigation pane, choose **Instances**.
13. From the list of instances, select the instance that you are using as a Hyperledger Fabric client.
14. Choose **Actions**, **Security**, **Modify IAM role**.
15. For **IAM role** begin typing the name of the role you created, for example, **`ServiceRoleForHyperledgerFabricClient`**. Select it from the list, and then choose **Apply**.

Using Service-Linked Roles for Managed Blockchain

Amazon Managed Blockchain uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Managed Blockchain. Service-linked roles are predefined by Managed Blockchain and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role can make setting up Managed Blockchain easier because you don't have to manually add the necessary permissions. Managed Blockchain defines the permissions of its service-linked roles, and, unless defined otherwise, only Managed Blockchain can assume its roles. The defined permissions include the trust policy and the permissions policy. The permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting its related resources. This protects your Managed Blockchain resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-Linked Role Permissions for Managed Blockchain

Managed Blockchain uses the service-linked role named **AWSServiceRoleForAmazonManagedBlockchain**. This role enables access to AWS Services and Resources used or managed by Amazon Managed Blockchain.

The AWSServiceRoleForAmazonManagedBlockchain service-linked role trusts the following services to assume the role:

- `managedblockchain.amazonaws.com`

The role permissions policy allows Managed Blockchain to complete actions on the specified resources shown in the following example policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "logs:CreateLogGroup"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:logs:*:*:log-group:/aws/managedblockchain/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:/aws/managedblockchain/*:log-stream:*"
            ]
        }
    ]
}
```

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

Creating a Service-Linked Role for Managed Blockchain

You don't need to manually create a service-linked role. When you create a network, a member, or a peer node, Managed Blockchain creates the service-linked role for you. It doesn't matter if you use the AWS Management Console, the AWS CLI, or the AWS API. The IAM entity performing the action must have permissions to create the service-linked role. After the role is created in your account, Managed Blockchain can use it for all networks and members.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a network, member, or node, Managed Blockchain creates the service-linked role for you again.

Editing a Service-Linked Role for Managed Blockchain

Managed Blockchain does not allow you to edit the AWSServiceRoleForAmazonManagedBlockchain service-linked role. After you create a service-linked role, you cannot change the name of the role

because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide*.

Deleting a Service-Linked Role for Managed Blockchain

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

> **Note**
> If the Managed Blockchain service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To manually delete the service-linked role

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAmazonManagedBlockchain service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported Regions for Managed Blockchain Service-Linked Roles

Managed Blockchain supports using service-linked roles in all of the Regions where the service is available. For more information, see AWS Regions and Endpoints.

Troubleshooting Hyperledger Fabric on Amazon Managed Blockchain identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Hyperledger Fabric on Managed Blockchain and IAM.

Topics

I Am Not Authorized to Perform an Action in Hyperledger Fabric on Managed Blockchain

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to create a member but does not have `managedblockchain:`*`CreateMember`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 managedblockchain:CreateMember on resource: n-MWY63ZJZU5HGNCMBQER7IN6OIU
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *n-MWY63ZJZU5HGNCMBQER7IN6OIU* resource using the `managedblockchain:`*`CreateMember`* action.

I'm an administrator and want to allow others to access Hyperledger Fabric on Managed Blockchain

To allow others to access Managed Blockchain, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Managed Blockchain.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

Configuring Security Groups for Hyperledger Fabric on Amazon Managed Blockchain

Security groups act as virtual firewalls. They control inbound and outbound traffic between your Hyperledger Fabric client and network resources on Managed Blockchain through the VPC endpoint in your account. By default, security group rules are restrictive, so you must add rules that allow traffic for any resources, such as client computers, that must access the network. The following tables list the minimum required security group rules that must be associated with the VPC endpoint and the Hyperledger Fabric client.

VPC Endpoint Security Group Minimum Rules

Inbound/Outbound	Type	Source/Destination	Purpose
Outbound	All traffic	0.0.0/0 (Anywhere)	Default. Allows unrestricted outbound traffic from the interface VPC endpoint to all recipients.
Inbound	Custom TCP, Port for Ordering Service (ranging between 30000 and 34000)—for example, 30001. The port is available within the **Ordering service endpoint** on the network details page using the console and returned within the `OrderingServiceEndpoint` property using the `get-network` command from the AWS CLI or using the `GetNetwork` API action.	The IPv4 address, an address range, or a security group that includes all members' Hyperledger Fabric clients.	Allows the Hyperledger Fabric ordering service to receive traffic from Hyperledger Fabric clients.
Inbound	Custom TCP, Port for the CA Service for a member (ranging between 30000 and 34000)—for example, 30002. This is unique to each member, and	The IPv4 address, an address range, or a security group that includes all members' Hyperledger Fabric clients.	Allows the Hyperledger Fabric certificate authority (CA) for each member to receive traffic from respective Hyperledger Fabric clients.

Inbound/Outbound	Type	Source/Destination	Purpose
	each member only needs access to their own CA. The port is available within the **Fabric certificate authority endpoint** on the member details page using the console and returned within the `CaEndpoint` property using the `get-member` command from the AWS CLI or using the `GetMember` API action.		
Inbound	Custom TCP, Ports, or Range of Ports for Peer Event Services on Peer Nodes (ranging between 30000 and 34000). The port is available within the **Peer node endpoints** on the member details page using the console and returned as the `PeerEventPort` property using the `get-node` command from the AWS CLI or using the `GetNode` API action.	The IPv4 address, an address range, or a security group that includes all members' Hyperledger Fabric clients.	Allows the network to receive traffic from peer nodes as required. Each node in each membership has a unique port associated with its peer event service. Any node that might be a participant in an endorsement policy, regardless of membership, must be allowed communications in order to endorse transactions.

Hyperledger Fabric Client Security Group Minimum Rules

Inbound/Outbound	Type	Source/Destination	Purpose
Outbound	All traffic	0.0.0/0 (Anywhere)	Default. Allows unrestricted outbound traffic from the Hyperledger Fabric client to all recipients. If necessary, you can limit the destination to the interface VPC endpoint.
Inbound	SSH (Port 22)	The IP address, address range, or security group that includes trusted SSH clients that connect to the Hyperledger Fabric client.	Allows trusted clients to use SSH to connect to the Hyperledger Fabric client to interact—for example, to query and run chaincode.

Tagging Amazon Managed Blockchain resources

A *tag* is a custom attribute label that you assign or that AWS assigns to an AWS resource. Each tag has two parts:

- A *tag key*, such as `CostCenter`, `Environment`, or `Project`. Tag keys are case-sensitive.
- An optional field known as a *tag value*, such as `111122223333` or `Production`. Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case-sensitive.

Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you could assign the same tag to an Amazon Managed Blockchain node and an EC2 instance that you use as a client for the Managed Blockchain framework.
- Track your AWS costs. You activate these tags on the AWS Billing and Cost Management dashboard. AWS uses the tags to categorize your costs and deliver a monthly cost allocation report to you. For more information, see Using cost allocation tags in the AWS Billing and Cost Management User Guide.
- Control access to your AWS resources with AWS Identity and Access Management (IAM). For information, see Controlling access using tags (p. 99) in this developer guide and Control access using IAM tags in the *IAM User Guide*.

For more information about tags, see the Tagging Best Practices guide.

The following sections provide more information about tags for Managed Blockchain.

Create and add tags for Hyperledger Fabric on Managed Blockchain resources

You can tag the following resources:

- Networks
- Members
- Nodes
- Proposals and invitations

  Members can assign tags to proposals when they create them. If a proposal is approved and an invitation is sent, the invitation inherits the tags added during proposal creation. Members who receive invitations or vote on proposals can subsequently add or remove tags for proposals and invitations. These subsequent tag edits are scoped only to the AWS account that made the edits.

Tags that you create—including those for network-wide resources like networks, proposals, and invitations—are scoped only to the account in which you create them. Tags created during proposal creation are the exception. Other AWS accounts participating on the network cannot access the tags.

Tag naming and usage conventions

The following basic naming and usage conventions apply to tags used with Managed Blockchain resources:

- Each resource can have a maximum of 50 tags.
- For each resource, each tag key must be unique, and each tag key can have only one value.
- The maximum tag key length is 128 Unicode characters in UTF-8.
- The maximum tag value length is 256 Unicode characters in UTF-8.
- Allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: `. : + = @ _ / -` (hyphen).
- Tag keys and values are case-sensitive. As a best practice, decide on a strategy for capitalizing tags, and consistently implement that strategy across all resource types. For example, decide whether to use `Costcenter`, `costcenter`, or `CostCenter`, and use the same convention for all tags. Avoid using similar tags with inconsistent case treatment.
- The `aws:` prefix is reserved for AWS use. You can't edit or delete a tag's key or value when the tag has a tag key with the `aws:` prefix. Tags with this prefix do not count against your limit of tags per resource.

Working with tags

You can use the Managed Blockchain console, the AWS CLI, or the Managed Blockchain API to add, edit, or delete tag keys and tag values. You can assign tags when you create a resource, or you can apply tags after the resource is created.

For more information about Managed Blockchain API actions for tagging, see the following topics in the *Amazon Managed Blockchain API Reference*:

- ListTagsForResource
- TagResource
- UntagResource

Add or remove tags

You can add a tag to Hyperledger Fabric on Managed Blockchain resources when you create or work with them.

Topics

Add or remove tags for networks

For information about adding a tag when you create a network, see Create a Hyperledger Fabric Blockchain Network on Amazon Managed Blockchain (p. 33).

To add or remove a tag for a Hyperledger Fabric network on Managed Blockchain using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. Choose **Networks** and then choose a Hyperledger Fabric network from the list.
3. Under **Tags**, choose **Edit tags**, and then do one of the following:
   - To add a tag, choose **Add new tag**, enter a **Key** and optional **Value**, and then choose **Save**.
   - To remove a tag, choose **Remove** next to the **Tag** you want to remove, and then choose **Save**.

Add or remove tags for members

For information about adding a tag when you create a member, see Create a Member and Join a Network (p. 41).

To add or remove a tag for a member

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. Choose **Networks** and then choose a Hyperledger Fabric network from the list.
3. Choose **Members**.
4. Under **Members owned by you**, choose a member from the list.
5. Under **Tags**, choose **Edit tags**, and then do one of the following:
   - To add a tag, choose **Add new tag**, enter a **Key** and optional **Value**, and then choose **Save**.
   - To remove a tag, choose **Remove** next to the **Tag** you want to remove, and then choose **Save**.

Add or remove tags for nodes

For information about adding a tag when you create a node, see Create a Hyperledger Fabric Peer Node on Amazon Managed Blockchain (p. 45).

To add or remove a tag for a node

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. Choose **Networks** and then choose a Hyperledger Fabric network from the list.
3. Choose **Members**.
4. Under **Members owned by you**, choose a member from the list.
5. Under peer nodes, choose a **Node ID** from the list.
6. Choose **Tags**, choose **Edit tags**, and then do one of the following:
   - To add a tag, choose **Add new tag**, enter a **Key** and optional **Value**, and then choose **Save**.
   - To remove a tag, choose **Remove** next to the **Tag** you want to remove, and then choose **Save**.

Add or remove tags for proposals

For information about adding a tag when you create a proposal, see Create an Invitation Proposal (p. 56) and Create a Proposal to Remove a Network Member (p. 57).

You can add and remove tags from active and completed proposals.

To add or remove a tag for a proposal

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. Choose **Networks** and then choose a network from the list.
3. Choose **Proposals**.
4. Under **Active** or **Completed**, choose a **Proposal ID** from the list.
5. Under **Tags**, choose **Edit tags**, and then do one of the following:

   - To add a tag, choose **Add new tag**, enter a **Key** and optional **Value**, and then choose **Save**.
   - To remove a tag, choose **Remove** next to the **Tag** you want to remove, and then choose **Save**.

Add or remove tags for invitations

If the member who created a proposal for an invitation tagged the proposal, the invitation inherits the tag key and value from the proposal.

To add or remove a tag for an invitation

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. Choose **Invitations** and then choose a **Network name** from the list.
3. Under **Tags**, choose **Edit tags**, and then do one of the following:

   - To add a tag, choose **Add new tag**, enter a **Key** and optional **Value**, and then choose **Save**.
   - To remove a tag, choose **Remove** next to the **Tag** you want to remove, and then choose **Save**.

Monitoring Hyperledger Fabric on Managed Blockchain Using CloudWatch Logs

Hyperledger Fabric on Amazon Managed Blockchain supports publishing peer node, chaincode, and Certificate Authority (CA) logs to Amazon CloudWatch Logs. You can use these logs to troubleshoot during chaincode development and to monitor network activity and errors.

You can enable and view logs in the Managed Blockchain management console, in the CloudWatch Logs console, and using AWS CLI commands for CloudWatch Logs. In addition, you can set up *metric filters* in CloudWatch Logs to turn log data into numerical CloudWatch metrics that you can graph and set an alarm on. For each member that has logging enabled, Managed Blockchain creates a log group in CloudWatch Logs. For more information about CloudWatch Logs, see the Amazon CloudWatch Logs User Guide. For more information about creating metric filters, see Searching and Filtering Log Data in the *Amazon CloudWatch Logs User Guide*.

- **Peer node logs** help you debug timeout errors associated with proposals and identify rejected proposals that do not meet the endorsement policies. Peer node logs contain messages generated when your client submits transaction proposals to peer nodes, requests to join channels, enrolls an admin peer, and lists the chaincode instances on a peer node. Peer node logs also contain the results of chaincode installation. You can enable and disable logs on individual peer nodes.
- **Chaincode logs** help you analyze and debug the business logic and execution of chaincode on a peer node. They contain the results of instantiating, invoking, and querying the chaincode. A peer can run multiple instances of chaincode. When you enable chaincode logging, individual log streams are created for each and every chaincode on the peer.
- **CA logs** help you determine when a member in your account joins the network, or when new peers register with a member CA. You can use CA logs to debug problems related to certificates and enrollment. CA logging can be enabled and disabled for each member. A single log stream for the CA exists for each member.

> **Note**
> Managed Blockchain gathers CloudWatch metrics for peer nodes automatically and separately from CloudWatch Logs for CAs, peer nodes, and chaincode. For more information, see Use Hyperledger Fabric Peer Node Metrics on Amazon Managed Blockchain (p. 48).

Considerations and Limitations

Consider the following before you enable and view CloudWatch Logs for Hyperledger Fabric on Managed Blockchain.

- CA logs can be enabled only for members created after April 6, 2020. Peer node logs and chaincode logs can be enabled only for peer nodes created after April 6, 2020.
- Log entries are updated every five seconds.
- Logging requires the service-linked role for Managed Blockchain. The role is created automatically when an IAM principal (user or role) in your account with permissions to create the service-linked role creates a network, member, or peer. For more information, see Using Service-Linked Roles for Managed Blockchain (p. 108).

- Logging currently does not support CloudWatch Logs encryption.
- Logging currently does not support CloudWatch Logs Insights.
- To log chaincode events, you must first configure Go chaincode for logging. For more information, see Logging Control for Go Chaincodes in the Hyperledger Fabric documentation.

Enabling and Disabling Logs

You can enable logs using the Managed Blockchain management console when you create a member or node, or at any time after a member or node is created. You can disable CA logging, peer node logging, and chaincode logging at any time. When a log is disabled, log entries are not published to CloudWatch Logs. Previous log entries are still viewable and available in CloudWatch Logs.

Enabling and Disabling Peer Node and Chaincode Logs

You can enable peer node logs, chaincode logs, or both when you create a peer node or when viewing information about a peer node. You can also disable logs while viewing information about a peer node.

To enable peer node or chaincode logs when you create a node

1. On the **Members** tab of the network that you are working with, under **Members owned by you**, choose the name of the member from the list, and then choose **Create peer node**.
2. Under **Logging configuration**, choose **Enable peer node logs**, **Enable chaincode logs**, or both, and then choose **Create peer node**.

To enable or disable peer node and chaincode logs for an existing member

1. On the **Members** tab of the network that you are working with, under **Members owned by you**, choose the name of the member from the list.
2. Under **Peer nodes**, choose the **Node ID** of the peer.
3. Under monitoring, choose **Peer node logs** or **Chaincode logs**.
4. Choose **Enable logging**, or choose **Actions** and then **Disable Logging**.

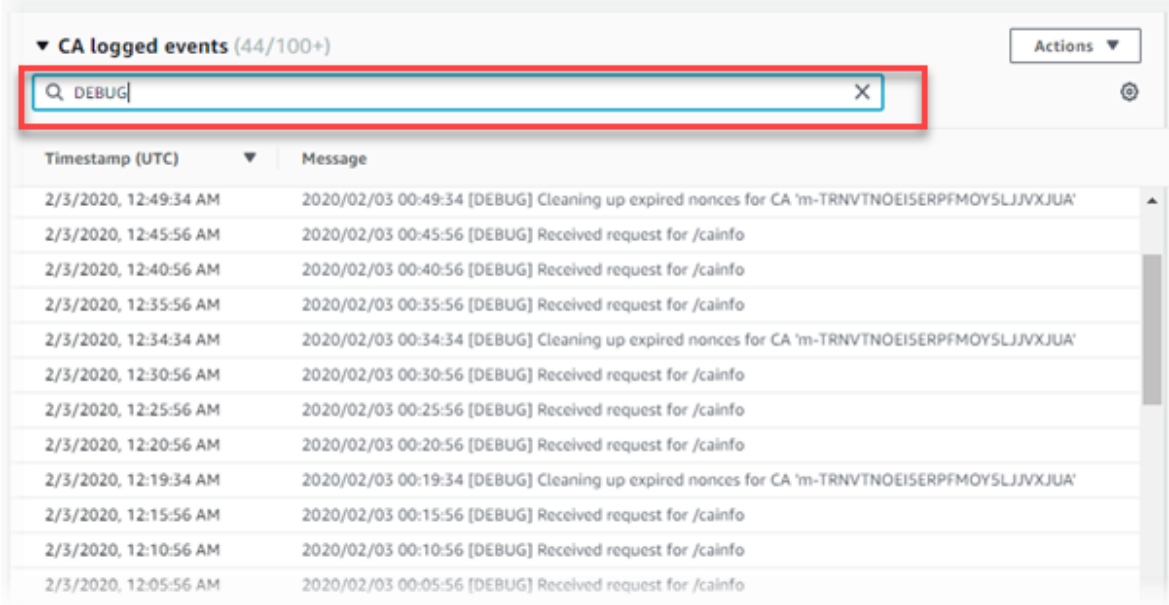
Working with Logged Events in the Managed Blockchain Console

Managed Blockchain publishes logged events to CloudWatch Logs every five seconds. By default, logged events are updated every five seconds in the Managed Blockchain console under **Logged events** on member and node information pages. When viewing logged events in Managed Blockchain, you can choose **Actions** and then view **View in CloudWatch** to open the CloudWatch Logs management console focused on the log stream that you are viewing. Choose the gear icon to configure the logging interval and other details.

Searching (Filtering) Logged Events

While viewing events for any log in Managed Blockchain, you can enter a keyword or phrase in the **Search events** box to show only those events that contain the search term. For example, you can enter a

date, a date and time, or a log level such as `CRITICAL`, `DEBUG`, or `WARNING`. When you download a log after searching, only the events filtered by your search term are downloaded.



Downloading Logged Events

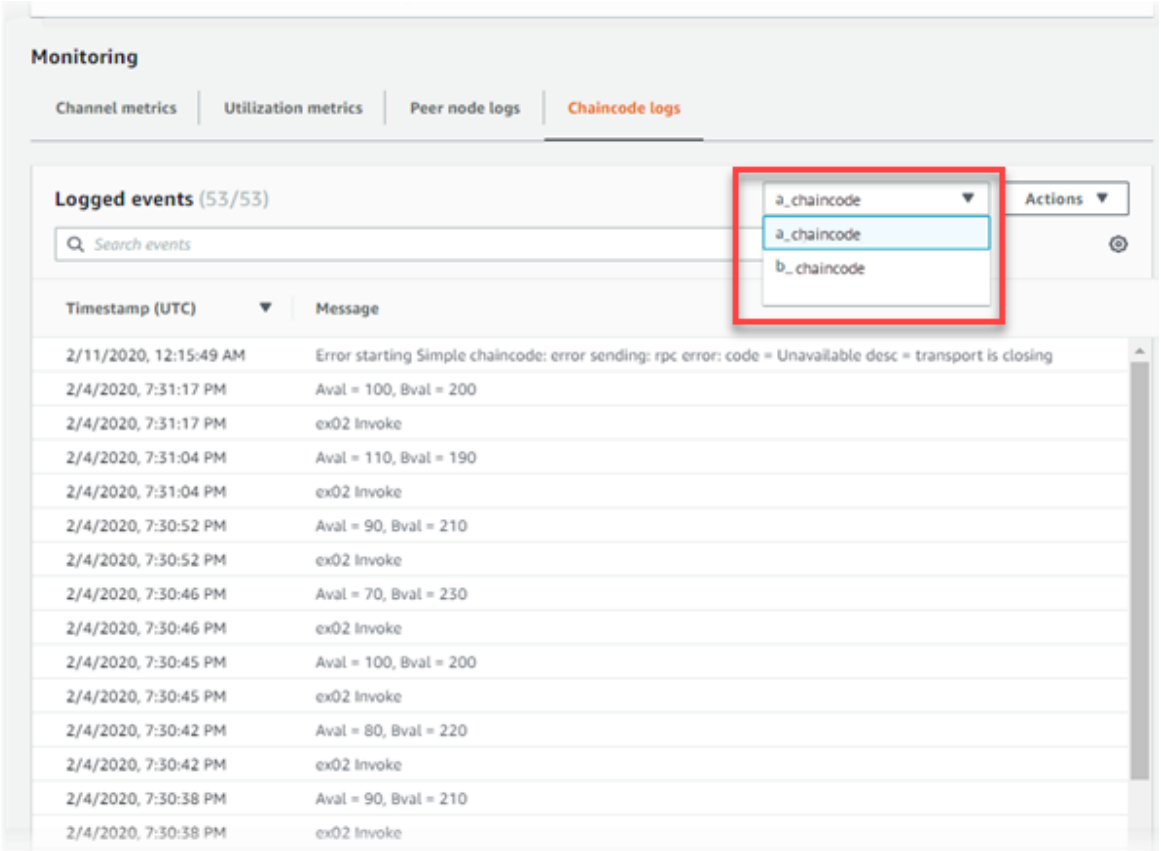
Choose **Actions** and then **Download** while viewing any log in Managed Blockchain to save the events that are loaded to the default download directory on your local machine with a `.log` extension. Logged events are listed along with a header that contains log information as shown in the following example.

```
Event Count: 100
Filtered: Yes
Filtered Event Count: 44
Filter: "DEBUG"

2020-02-03T01:04:34.548Z 2020/02/03 01:04:34 [DEBUG] Cleaning up expired nonces for CA 'm-
J46DNSFRTVCCLONS9DT5TTLS2A'
2020-02-03T01:00:56.382Z 2020/02/03 01:00:56 [DEBUG] Received request for /cainfo
2020-02-03T00:55:56.308Z 2020/02/03 00:55:56 [DEBUG] Received request for /cainfo
2020-02-03T00:50:56.208Z 2020/02/03 00:50:56 [DEBUG] Received request for /cainfo
2020-02-03T00:49:34.544Z 2020/02/03 00:49:34 [DEBUG] Cleaning up expired nonces for CA 'm-
J46DNSFRTVCCLONS9DT5TTLS2A'
2020-02-03T00:45:56.282Z 2020/02/03 00:45:56 [DEBUG] Received request for /cainfo
2020-02-03T00:40:56.111Z 2020/02/03 00:40:56 [DEBUG] Received request for /cainfo
2020-02-03T00:35:56.026Z 2020/02/03 00:35:56 [DEBUG] Received request for /cainfo
2020-02-03T00:34:34.539Z 2020/02/03 00:34:34 [DEBUG] Cleaning up expired nonces for CA 'm-
J46DNSFRTVCCLONS9DT5TTLS2A'
2020-02-03T00:30:56.081Z 2020/02/03 00:30:56 [DEBUG] Received request for /cainfo
2020-02-03T00:25:56.123Z 2020/02/03 00:25:56 [DEBUG] Received request for /cainfo
2020-02-03T00:20:56.197Z 2020/02/03 00:20:56 [DEBUG] Received request for /cainfo
```

Viewing Different Chaincode Logs

When viewing chaincode logs for a peer node with multiple chaincodes, you can choose the chaincode to view by choosing the chaincode name from the list next to **Logged events**. When you download a log, only the logged events for the chaincode that you are viewing are downloaded.



Identifying Logs in CloudWatch Logs

Each set of log events in Managed Blockchain corresponds to a *log stream* in CloudWatch Logs. The easiest way to access a log stream in the CloudWatch Logs management console is to choose **Actions** and then **View in CloudWatch** while viewing a log in the Managed Blockchain management console.

All log streams associated with a member and peer nodes that a member owns are in a log group with the naming pattern shown below—for example, **aws/managedblockchain/n-MWY63ZJZU5HGNCMBQER7IN6OIU/m-J46DNSFRTVCCLONS9DT5TTLS2A**.

```
aws/managedblockchain/NetworkID/MemberID
```

Log streams in the log group are named according to the patterns in the table below.

Type of log	Stream name
Peer node logs	*PeerNodeID*, for example, **nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y**.
Chaincode logs	*MemberID*-*PeerNodeID*-*ChaincodeName*-*ChaincodeVersion*, for example, **m-J46DNSFRTVCCLONS9DT5TTLS2A-nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y-MyChaincode-v0**.
CA logs	**ca**

Logging Amazon Managed Blockchain API calls using AWS CloudTrail

Amazon Managed Blockchain is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Managed Blockchain. CloudTrail captures all API calls for Managed Blockchain as events. The calls captured include calls from the Managed Blockchain console and code calls to the Managed Blockchain API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Managed Blockchain. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Managed Blockchain, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Managed Blockchain information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Managed Blockchain, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail Event history.

For an ongoing record of events in your AWS account, including events for Managed Blockchain, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple Regions and Receiving CloudTrail log files from multiple accounts

All Managed Blockchain actions are logged by CloudTrail and are documented in the Amazon Managed Blockchain API Reference. For example, calls to the `CreateNode`, `GetNode` and `DeleteNetwork` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.

- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding Managed Blockchain log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `GetNode` action.


```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "ABCD1EF23G4EXAMPLE56:carlossalazar",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/carlossalazar",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "ABCD1EF23G4EXAMPLE56",
        "arn": "arn:aws:iam::111122223333:role/Admin",
        "accountId": "111122223333",
        "userName": "Admin"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-12-10T05:36:38Z"
      }
    }
  },
  "eventTime": "2020-12-10T05:50:48Z",
  "eventSource": "managedblockchain.amazonaws.com",
  "eventName": "GetNode",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "198.51.100.1",
  "userAgent": "aws-cli/2.0.7 Python/3.7.3 Linux/5.4.58-37.125.amzn2int.x86_64
 botocore/2.0.0dev11",
  "requestParameters": {
    "networkId": "n-MWY63ZJZU5HGNCMBQER7IN6OIU",
    "nodeId": "nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y"
  },
  "responseElements": null,
  "requestID": "1e2xa3m4-56p7-8l9e-0ex1-23456a78m90p",
  "eventID": "ex12345a-m678-901p-23e4-567ex8a9mple",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

Document History for Hyperledger Fabric on Amazon Managed Blockchain Developer 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
Guide updated to be specific to Hyperledger Fabric on Managed Blockchain.	Updated title from Amazon Managed Blockchain Management Guide to Hyperledger Fabric on Amazon Managed Blockchain Developer Guide in conjunction with release of Ethereum framework support. Slight changes to content throughout, clarifying applicability to Hyperledger Fabric networks, members, peer nodes, and chaincode.	November 15, 2020
Updated for Hyperledger Fabric v1.4, including anchor peers, private data collections, Java chaincode.	Added sections on setting up anchor peers, setting up channels, using private data collections, and running Java chaincode. Updated getting started tutorial for Hyperledger Fabric v1.4. Added new peer node metrics.	September 14, 2020
Updated for support of Amazon CloudWatch Logs and streamlined Getting Started experience.	Added Monitoring chapter for publishing peer node, chaincode, and CA logs to CloudWatch Logs. Miscellaneous improvements to getting started tutorial to streamline and simplify.	April 6, 2020
Major updates for new proposal and voting work flow for member invitations and removals	Updated Getting Started tutorial, conceptual information, and procedures for new voting proposal design.	April 8, 2019
Added security group configuration guidance	Added prescriptive guidance for configuring security groups for the tutorial. Added references for minimum inbound and outbound security group rules required for Hyperledger Fabric client and interface VPC	February 28, 2019

	endpoint for reference and customization.	
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

AWS glossary

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