Amazon Managed Blockchain

Ethereum Developer Guide

Amazon Managed Blockchain: Ethereum 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 Ethereum blockchain resources using Managed Blockchain. For information about working with Hyperledger Fabric on Managed Blockchain, see Hyperledger Fabric on Amazon Managed Blockchain Developer Guide.

Key concepts: Ethereum on Amazon Managed Blockchain

You can use Ethereum on Amazon Managed Blockchain to quickly provision Ethereum nodes and join them to the public Ethereum *mainnet* or popular public *testnets*. Ethereum nodes on a network collectively store the state of the Ethereum blockchain, verify transactions, and participate in consensus to change the state of the blockchain.

An Ethereum node allows you to develop and use decentralized applications (dapps) that interact with an Ethereum blockchain. The "back-end" of a dapp is a *smart contract* that runs in a decentralized way across all the nodes joined to an Ethereum network. Anyone joined to the network can develop and deploy a smart contract that adds functionality.

The "front end" of a dapp uses methods in the Ethereum JSON-RPC API to interact with the Ethereum network through your Ethereum node in Managed Blockchain, reading data and writing transactions. With Ethereum on Managed Blockchain, your front-end app can use an HTTP or WebSockets connection to make API calls. Only users in the AWS account that owns the node can make API calls. Calls over HTTP and WebSockets are authenticated using the Signature Version 4 signing process.

> **Important**
> Amazon Managed Blockchain helps customers provision Ethereum nodes. You are responsible for the creation, maintenance, and use of your Ethereum Accounts. You are also responsible for the contents of your Ethereum Accounts, which includes, but is not limited to, Ether (ETH) and smart contracts. AWS is not responsible for any of your smart contracts tested, compiled, deployed or called using Ethereum nodes with Amazon Managed Blockchain.

This guide assumes that you're familiar with the concepts essential to Ethereum, such as nodes, dapps, transactions, gas, Ether, and others. Before you deploy a node using Ethereum on Managed Blockchain and develop dapps, we recommend that you review the Ethereum Development Documentation and Mastering Ethereum.

Considerations and limitations for Ethereum on Managed Blockchain

When using Ethereum on Managed Blockchain to host a node on an Ethereum network, consider the following.

- **Supported networks**

  Ethereum has a public *mainnet* and several public *testnets* used for development, testing, and proof of concept. Managed Blockchain supports the following public networks. Private networks are not supported.

  - **Mainnet** – This is the primary public Ethereum production blockchain proof-of-work network. Transactions on mainnet have actual value (they have real costs) and occur on the distributed ledger.
  - **Rinkeby** – This is a public proof-of-authority testnet for Go Ethereum (Geth) clients. Ether on this network has no real monetary value.
  - **Ropsten** – This is a public proof-of-work testnet, so operations on this network closely resemble those on mainnet. Ether on this network has no real monetary value.
- **Mining not supported**

Ethereum nodes created using Managed Blockchain do not support mining.

- **Different endpoints for WebSockets and HTTP**

  Ethereum on Managed Blockchain supports the Ethereum JSON-RPC API over HTTP and WebSockets. Each Ethereum node in Managed Blockchain hosts different endpoints for HTTP and WebSockets.

- **JSON-RPC batch requests not supported**

  Ethereum nodes created using Managed Blockchain do not support JSON-RPC batch requests.

- **Payload limit for API calls**

  WebSockets calls have a 512 KB payload limit. Some calls may exceed this limit and cause a "message response is too large" error. Use HTTP for these requests instead of WebSockets. HTTP requests have a 6 MB payload limit. Responses that exceed the HTTP also result in a "message response is too large" error.

- **Signature Version 4 signing of API calls**

  All Ethereum JSON-RPC API calls to an Ethereum node on Managed Blockchain are authenticated using the Signature Version 4 signing process. This means that only authorized IAM principals in the AWS account that created the node can interact with it using the API. AWS credentials (an access key ID and secret access key) must be provided with the call. IAM policies do not apply to API calls and can not be used to allow or deny Ethereum API access to a node. IAM permissions policies only apply to operations related to node creation, deletion, and management.

  Client credentials should never be embedded in user-facing applications. To expose an Ethereum node on Managed Blockchain to anonymous users visiting from trusted web domains, you can set up a separate endpoint in Amazon API Gateway backed by a Lambda function that forwards requests to your node using the proper IAM credentials.

- **Only raw transactions are supported**

  Managed Blockchain only supports the use of the `eth_sendRawTransaction` method to submit transactions that update the Ethereum blockchain state. These transactions must be created and signed using Ethereum private keys outside Managed Blockchain before they are sent. In other words, Managed Blockchain cannot be used as an Ethereum wallet. Ethereum transactions and private keys must be generated and stored externally.

- **Node limit per account**

  Managed Blockchain supports a maximum of 50 Ethereum nodes per account.

Setting up for Ethereum on Managed Blockchain

Sign up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all AWS services, including Amazon Managed Blockchain. You are charged only for the services that you use.

With Ethereum on Managed Blockchain, you pay for the node, the storage you use, and the number of requests between the node and the network.

If you have an AWS account already, move on to the next step. If you don't have an AWS account, use the following procedure to create one.

To create 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.

Create an IAM user with appropriate permissions

To create and work with Ethereum resources in Managed Blockchain, you need an AWS Identity and Access Management (IAM) principal (user or group) with permissions that allow necessary Managed Blockchain actions on those resources, such as creating or deleting nodes.

An IAM principal is also required to make Ethereum JSON-RPC API calls. All Ethereum JSON-RPC API calls to an Ethereum node on Managed Blockchain are authenticated using the Signature Version 4 signing process. This means that only authorized IAM principals in the AWS account that created the node can interact with it using the API. AWS credentials (an access key ID and secret access key) must be provided with the call. IAM policies do not apply to API calls and can not be used to allow or deny Ethereum API access to a node. IAM permissions policies only apply to operations related to node creation, deletion, and management.

For information about creating an IAM user, see Creating an IAM user in your AWS account. For more information about attaching a permissions policy to a user, see Changing permissions for an IAM user. For an example of a permissions policy that you can use to give a user permission to work with Ethereum on Managed Blockchain resources, see Performing all available actions for Ethereum on Managed Blockchain (p. 33).

Working with Ethereum nodes using Managed Blockchain

Creating a node

When you create an Ethereum node, you select the network that the node joins and the configuration details such as the instance type and the Ethereum node type. Creating an Ethereum node in Amazon Managed Blockchain creates a full Geth node on the selected Ethereum network. The IAM principal (user or group) that you use must have permissions to create nodes and view node information. For more information, see Performing all available actions for Ethereum on Managed Blockchain (p. 33).

Select the following characteristics when you create an Ethereum node:

- **Blockchain network** – Managed Blockchain supports the following public Ethereum networks:
  - **Mainnet** – This is the primary public Ethereum production blockchain proof-of-work network. Transactions on mainnet have actual value (they have real costs) and occur on the distributed ledger.
  - **Rinkeby** – This is a public proof-of-authority testnet for Go Ethereum (Geth) clients. Ether on this network has no real monetary value.
  - **Ropsten** – This is a public proof-of-work testnet, so operations on this network closely resemble those on mainnet. Ether on this network has no real monetary value.
- **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.
- **Ethereum node type** – The only node type that is currently supported is **Full node (Geth)**. For more information about node types and Go Ethereum (Geth), see Nodes and clients in the Ethereum developer documentation.
- **Availability Zone** – You can select the Availability Zone to launch the Ethereum node in. The ability to distribute nodes across different Availability Zones lets you design your blockchain application for resiliency. For more information, see Regions and Availability Zones in the *Amazon EC2 User Guide for Linux Instances*.

After you create the node, the **Node** details page displays the endpoints that you use to make Ethereum JSON-RPC API calls from code on a client. There are separate endpoints for HTTP connections and WebSockets connections. For more information about sending API calls to an Ethereum node on Managed Blockchain to interact with smart contracts, see Using the Ethereum JSON-RPC API with Amazon Managed Blockchain (p. 10).

To create an Ethereum node using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. Choose **Join public network**.
3. Choose the **Blockchain network** for the node to join according to the preceding guidelines.

4. Choose a **Blockchain instance type** suitable for your application. In general, choose an instance type with more CPU and RAM if your nodes need to process a higher rate of transactions more efficiently.
5. For **Ethereum node type**, choose **Full node (Geth)**.
6. Choose **Create node**.

   Managed Blockchain provisions and configures the node for you. The length of this process depends on many variables. It may take a few minutes for nodes on testnets, and up to an hour or more for nodes on mainnet.

To create an Ethereum node using the AWS CLI

Use the `create-node` command, as shown in the following example. Replace the value of `--network-id`, `InstanceType`, and `AvailabilityZone` as appropriate.

```
aws managedblockchain create-node \
  --node-configuration '{"InstanceType":"bc.t3.large","AvailabilityZone":"us-east-1a"}' \
  --network-id n-ethereum-rinkeby
```

Ethereum public networks have the following network IDs:

- `n-ethereum-mainnet`
- `n-ethereum-rinkeby`
- `n-ethereum-ropsten`

The command returns the node ID, as shown in the following example.

```
{
    "NodeId": "nd-RG3GM4U7HFFHHHGJHHU7UNPCLU"
}
```

Viewing node details

After you create a node, you can view administrative properties for each node that your AWS account owns, such as the endpoints to use for Ethereum JSON-RPC API calls for WebSockets and HTTP, the node status, and important performance metrics for the node. The IAM principal (user or group) that you use must have permissions to list and get node information. For more information, see Identity-based policy examples (p. 31).

Basic information such as the Managed Blockchain instance type, Availability Zone, and creation date, are available for the node, along with the following important properties:

- **Status**
  - **Creating**

    Managed Blockchain is provisioning and configuring the Managed Blockchain instance for the node. Creation time depends on many factors. Nodes on testnets typically take a few minutes. Nodes on mainnet may take an hour or more.
  - **Available**

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

Managed Blockchain detected a problem and is automatically replacing the blockchain instance on which the node runs. Nodes in an unhealthy state typically return to an available state in approximately five minutes.

- **Failed**

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

- **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.

- **Deleted**

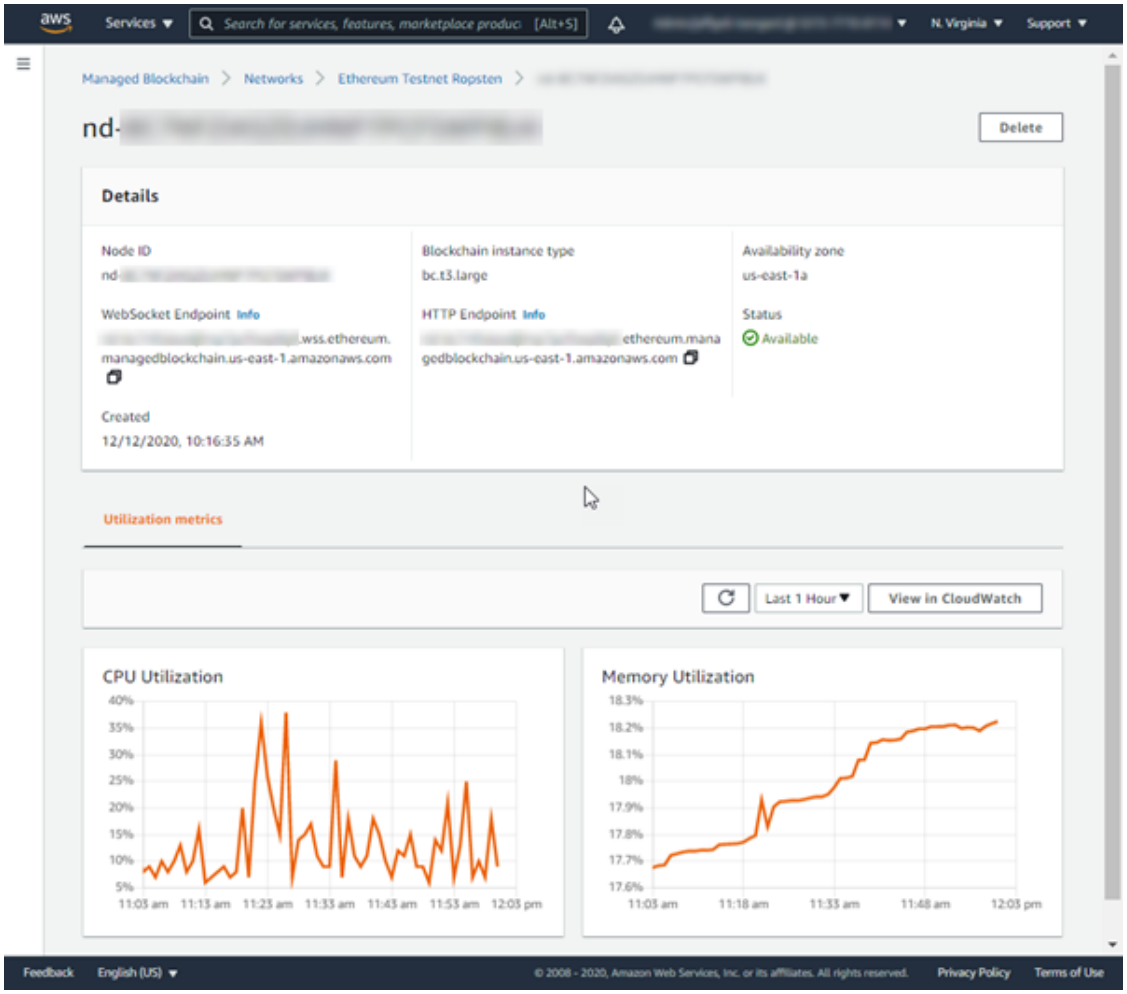
  The node has been deleted. See the previous item for possible reasons.

- **Endpoints**

  Endpoints are used to make Ethereum JSON-RPC API calls to the node. Managed Blockchain assigns unique endpoints when it creates the node. Nodes support connection over HTTP and WebSockets. You use a different endpoint for each connection. For more information, see API call examples (p. 17).

To view Ethereum node information using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. If the console doesn't open to the **Networks** list, choose **Networks** from the navigation pane.
3. Choose the **Name** of the Ethereum network that the node belongs to from the list.
4. On the network details page, under **Nodes**, choose the **Node ID**.
5. The node details page displays key properties and metrics for the node, as shown in the example that follows.



To view Ethereum node information using the AWS CLI

Use the `get-node` command to view Ethereum node information, as shown in the following example. Replace the value of `--network-id` and `--node-id` as appropriate.

```
aws managedblockchain get-node \
  --network-id n-ethereum-rinkeby \
  --node-id nd-RG3GM4U7HFFHHHGJHHU7UNPCLU
```

The command returns output that includes the node's `HttpEndpoint`, `WebSocketEndpoint`, and other key properties, as shown in the following example.

```
{
    "Node": {
        "NetworkId": "n-ethereum-rinkeby",
        "Id": "nd-RG3GM4U7HFFHHHGJHHU7UNPCLU",
        "InstanceType": "bc.t3.large",
        "AvailabilityZone": "us-east-1a",
        "FrameworkAttributes": {
            "Ethereum": {
                "HttpEndpoint": "nd-
rg3gm4u7hffhhhgjhhu7unpclu.ethereum.managedblockchain.us-east-1.amazonaws.com",
```

```
                "WebSocketEndpoint": "nd-
rg3gm4u7hffhhhgjhhu7unpclu.wss.ethereum.managedblockchain.us-east-1.amazonaws.com"
            }
        },
        "Status": "CREATING",
        "CreationDate": "2021-06-25T20:10:18.555000+00:00",
        "Tags": {},
        "Arn": "arn:aws:managedblockchain:us-east-1:111122223333:nodes/nd-
RG3GM4U7HFFHHHGJHHU7UNPCLU"
    }
}
```

Deleting a node

When you delete an Ethereum node from Managed Blockchain, all resources stored on that node are immediately deleted. The IAM principal (user or group) that you use must have permissions to delete nodes. For more information, see Performing all available actions for Ethereum on Managed Blockchain (p. 33).

To delete an Ethereum node using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. If the console doesn't open to the **Networks** list, choose **Networks** from the navigation pane.
3. Choose the **Name** of the Ethereum network that the node belongs to from the list.
4. On the network details page, under **Nodes**, select the **Node ID** and then choose **Delete**.

To delete an Ethereum node using the AWS CLI

Use the `delete-node` command to delete an Ethereum node, as shown in the following example. Replace the value of `--network-id` and `--node-id` as appropriate.

```
aws managedblockchain delete-node \
  --network-id n-ethereum-rinkeby \
  --node-id nd-RG3GM4U7HFFHHHGJHHU7UNPCLU
```

Using the Ethereum JSON-RPC API with Amazon Managed Blockchain

This topic provides a listing of Ethereum JSON-RPC methods that Managed Blockchain supports, followed by code examples that implement JSON-RPC API calls from clients using either WebSockets or HTTP.

You use the Ethereum JSON-RPC API from a client to query smart contract data and submit transactions to an Ethereum node in Managed Blockchain. The client uses either an HTTP or WebSockets endpoint hosted by the Ethereum node in Managed Blockchain to connect and send the API calls. You can get the node endpoints using the node details page in the Managed Blockchain console. For more information, see Viewing node details (p. 6).

Ethereum on Managed Blockchain only supports the `eth_sendRawTransaction` method, which requires that you create and sign the transaction before sending it to the node. Managed Blockchain does not have any way to sign transactions similar to an Ethereum wallet application.

All Ethereum JSON-RPC API calls to an Ethereum node on Managed Blockchain are authenticated using the Signature Version 4 signing process. This means that only authorized IAM principals in the AWS account that created the node can interact with it using the API. AWS credentials (an access key ID and secret access key) must be provided with the call. IAM policies do not apply to API calls and can not be used to allow or deny Ethereum API access to a node. IAM permissions policies only apply to operations related to node creation, deletion, and management.

Supported JSON-RPC methods

Ethereum on Managed Blockchain supports the following Ethereum JSON-RPC API methods. Methods not listed are not supported. JSON-RPC batch requests are also not supported. Each supported API call has a brief description of its utility. Unique considerations for using the JSON-RPC method with an Ethereum node in Managed Blockchain are indicated where applicable.

> **Important**
> WebSockets calls have a 512 KB payload limit. Some calls may exceed this limit and cause a "message response is too large" error. Use HTTP for these requests instead of WebSockets. HTTP requests have a 6 MB payload limit. Responses that exceed the HTTP also result in a "message response is too large" error.

The block identifier parameter

Some methods have an extra block identifier parameter. The following options are possible values for this parameter:

- A hexadecimal string value that represents an integer block number.
- `"earliest"` – String for the genesis block.
- `"latest"` – String for the latest mined block.
- `"pending"` – String for the pending state transactions.

Method	Description	Considerations
eth_blockNumber	Returns the number of the most recent block.	
eth_call	Immediately executes a new message call without creating a transaction on the blockchain.	eth_call consumes 0 gas, but has a gas parameter for messages that require it.
eth_chainId	Returns an integer value for the currently configured `Chain Id` value introduced in EIP-155. Returns `None` if no `Chain Id` is available.	
eth_estimateGas	Estimates and returns the gas required for a transaction without adding the transaction to the blockchain.	
eth_gasPrice	Returns the current price per gas in Wei.	
eth_getBalance	Returns the balance of an account for the specified account address and block identifier.	
eth_getBlockByHash	Returns information about the block specified using the block hash.	
eth_getBlockByNumber	Returns information about the block specified using the block number.	
eth_getBlockTransactionCountByHash	Returns the number of transactions in the block specified	

Method	Description	Considerations
	using the block hash.	
eth_getBlockTransactionCountByNumber	Returns the number of transactions in the block specified using the block number.	
eth_getCode	Returns the code at the specified account address and block identifier.	
eth_getFilterChanges	Polls the specified filter ID, retuning an array of logs that occurred since the last poll.	Filters are ephemeral. If Managed Blockchain needs to manage or maintain node instances for availability and performance, and an instance is replaced, filters may be deleted. We recommend that you write your application code to handle the occasional deletion of filters.
eth_getFilterLogs	Returns an array of all logs for the specified filter ID.	Filters are ephemeral. If Managed Blockchain needs to manage or maintain node instances for availability and performance, and an instance is replaced, filters may be deleted. We recommend that you write your application code to handle the occasional deletion of filters.

Method	Description	Considerations
eth_getLogs	Returns an array of all logs for a specified filter object.	Filters are ephemeral. If Managed Blockchain needs to manage or maintain node instances for availability and performance, and an instance is replaced, filters may be deleted. We recommend that you write your application code to handle the occasional deletion of filters.
eth_getProof	*Experimental* – Returns the account and storage values of the specified account, including the Merkle proof.	
eth_getStorageAt	Returns the value of the specified storage position for the specified account address and block identifier.	
eth_getTransactionByBlockHashAndIndex	Returns information about a transaction using the specified block hash and transaction index position.	
eth_getTransactionByBlockNumberAndIndex	Returns information about a transaction using the specified block number and transaction index position.	
eth_getTransactionByHash	Returns information about the transaction with the specified transaction hash.	

Method	Description	Considerations
eth_getTransactionCount	Returns the number of transactions sent from the specified address and block identifier.	
eth_getTransactionReceipt	Returns the receipt of the transaction using the specified transaction hash.	
eth_getUncleByBlockHashAndIndex	Returns information about the uncle block specified using the block hash and uncle index position.	
eth_getUncleByBlockNumberAndIndex	Returns information about the uncle block specified using the block number and uncle index position.	
eth_getUncleCountByBlockHash	Returns the number of counts in the uncle specified using the uncle hash.	
eth_getUncleCountByBlockNumber	Returns the number of counts in the uncle specified using the uncle number.	
eth_getWork	Returns the hash of the current block, the seedHash, and the boundary condition (also called the "target") to be met.	

Method	Description	Considerations
eth_newBlockFilter	Creates a filter in the node to notify when a new block arrives. Use <code>eth_getFilterChanges</code> to check for state changes.	
eth_newFilter	Creates a filter object with the specified filter options (such as from block, to block, contract address, or topics).	
eth_newPendingTransactionFilter	Creates a filter in the node to notify when new pending transactions arrive. Use <code>eth_getFilterChanges</code> to check for state changes.	
eth_protocolVersion	Returns the current Ethereum protocol version.	
eth_sendRawTransaction	Creates a new message call transaction or a contract creation for signed transactions.	Managed Blockchain supports raw transactions only. You must create and sign transactions before sending them. For more information, see How to create raw transactions in Ethereum.
eth_subscribe	*Experimental for publication subscription* – Creates a subscription for specified events and returns a subscription ID.	Available only when using WebSockets. Subscriptions are coupled to each connection. When the connection closes, the subscription is removed.

Method	Description	Considerations
eth_syncing	Returns an object with sync status data or `false` when not syncing.	
eth_uninstallFilter	Uninstalls the filter with the specified filter ID.	
eth_unsubscribe	*Experimental for publication subscription* – Cancels the subscription with the specified subscription ID.	
net_listening	Returns `true` if the client is actively listening for network connections.	
net_peerCount	Returns the number of peers currently connected to the client.	
net_version	Returns the current network ID.	
txpool_inspect	Lists a textual summary of all transactions currently pending inclusion in the next blocks, and those that are queued (being scheduled for future execution only).	
txpool_status	Provides a count of all transactions currently pending inclusion in the next blocks, and those that are queued (being scheduled for future execution only).	

Method	Description	Considerations
web3_clientVersion	Returns the current client version.	
web3_sha3	Returns Keccak-256 (not the standardized SHA3-256) of the given data.	

Examples – making Ethereum API calls to an Ethereum node in Amazon Managed Blockchain

The following examples demonstrate ways to make Ethereum API calls to an Ethereum node on Amazon Managed Blockchain.

Topics

> **Important**
> The Signature Version 4 signing process requires credentials associated with an AWS account. Some examples in this section export these sensitive credentials to the shell environment of the client. Only use these examples on a client running in a trusted context. Do not use these examples in an untrusted context, such as in a web browser or mobile app. Client credentials should never be embedded in user-facing applications. To expose an Ethereum node on Managed Blockchain to anonymous users visiting from trusted web domains, you can set up a separate endpoint in Amazon API Gateway backed by a Lambda function that forwards requests to your node using the proper IAM credentials.

Endpoint format for WebSockets and HTTP

An Ethereum node created using Ethereum on Managed Blockchain hosts one endpoint for WebSockets connections and another for HTTP connections. These endpoints conform to the following patterns.

> **Note**
> The node ID is case-sensitive and must be lowercase where indicated, or a signature mismatch error occurs.

WebSockets endpoint format

```
wss://my-node-id-lowercase.wss.ethereum.managedblockchain.us-east-1.amazonaws.com/
```

For example, `wss://nd-6eaj5va43jggnpxouzp7y47e4y.wss.ethereum.managedblockchain.us-east-1.amazonaws.com/`

HTTP endpoint format

```
https://my-node-id-lowercase.ethereum.managedblockchain.us-east-1.amazonaws.com/
```

For example, `https://nd-6eaj5va43jggnpxouzp7y47e4y.ethereum.managedblockchain.us-east-1.amazonaws.com/`

Using web3.js

Web3.js is a popular collection of JavaScript libraries available using the Node package manager (npm). You can run the following examples to send a JSON-RPC API call to Ethereum using a Javascript file for Node.js. The examples demonstrate an HTTP connection and a WebSockets connection to an Ethereum node.

Both HTTP and WebSockets connection types rely on a local connection provider library to open the Signature Version 4 authenticated connection to the Ethereum node. You install the provider for the connection locally by copying the source code to a file on your client. You then reference the library files in the script that makes the Ethereum API call.

Prerequisites

Running the example scripts requires the following prerequisites. Prerequisites for both HTTP and WebSockets connections are included.

1. You must have node version manager (nvm) and Node.js installed on your machine. If you are using an Amazon EC2 instance as your Ethereum client, see Tutorial: Setting Up Node.js on an Amazon EC2 Instance for more information.
2. Type `node --version` and verify that you are using Node version 14 or later. If necessary, you can use the `nvm install 14` command followed by the `nvm use 14` command to install version 14.
3. Use node package manager (npm) to install the `aws-sdk`, `web3`, and `xhr2` packages as shown in the following examples.

```
npm install aws-sdk
```

```
npm install web3
```

```
npm install xhr2
```

4. The example scripts use ES modules. To enable ECMAScript (ES) module support, add the `"type": "module"` line to your `package.json` file. The example that follows shows the contents of a simple `package.json` file.

```
{
  "type": "module",
  "dependencies": {
    "aws-sdk": "^2.809.0",
    "web3": "^1.3.0",
    "xhr2": "^0.2.0"
  }
}
```

5. The environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` must contain credentials associated with the same AWS account that created the node. The environment variables `AMB_HTTP_ENDPOINT` and `AMB_WS_ENDPOINT` must contain your Ethereum node's HTTP and WebSockets endpoints respectively.

Export these variables as strings on your client as shown in the examples that follow. Replace the values with appropriate values from your IAM user account.

```
export AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
```

```
export AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
```

```
export
 AMB_HTTP_ENDPOINT="https://nd-6eaj5va43jggnpxouzp7y47e4y.ethereum.managedblockchain.us-
east-1.amazonaws.com/"
```

```
export
 AMB_WS_ENDPOINT="wss://nd-6eaj5va43jggnpxouzp7y47e4y.wss.ethereum.managedblockchain.us-
east-1.amazonaws.com/"
```

To make an Ethereum API call using web3.js over HTTP to your Ethereum node on Managed Blockchain

1. Copy the contents of the example that follows, and then use your preferred text editor to save it to a file named `aws-http-provider.js` on your client machine in the same directory where you run your script.

   Contents of aws-http-provider.js

```
////////////////////////////////////////////////////
// Authored by Carl Youngblood
// Senior Blockchain Solutions Architect, AWS
// Adapted from web3 npm package v1.3.0
// licensed under GNU Lesser General Public License
// https://github.com/ethereum/web3.js
////////////////////////////////////////////////////

import AWS from 'aws-sdk';
import HttpProvider from 'web3-providers-http';
import XHR2 from 'xhr2';

export default class AWSHttpProvider extends HttpProvider {
  send(payload, callback) {
    const self = this;
    const request = new XHR2(); // eslint-disable-line

    request.timeout = self.timeout;
    request.open('POST', self.host, true);
    request.setRequestHeader('Content-Type', 'application/json');

    request.onreadystatechange = () => {
      if (request.readyState === 4 && request.timeout !== 1) {
        let result = request.responseText; // eslint-disable-line
        let error = null; // eslint-disable-line

        try {
          result = JSON.parse(result);
        } catch (jsonError) {
          let message;
          if (!!result && !!result.error && !!result.error.message) {
            message = `[aws-ethjs-provider-http] ${result.error.message}`;
          } else  {
```

```
            message = `[aws-ethjs-provider-http] Invalid JSON RPC response from host
 provider ${self.host}: ` +
               `${JSON.stringify(result, null, 2)}`;
          }
          error = new Error(message);
        }

        callback(error, result);
      }
    };

    request.ontimeout = () => {
      callback(`[aws-ethjs-provider-http] CONNECTION TIMEOUT: http request timeout
 after ${self.timeout} ` +
        `ms. (i.e. your connect has timed out for whatever reason, check your
 provider).`, null);
    };

    try {
      const strPayload = JSON.stringify(payload);
      const region = process.env.AWS_DEFAULT_REGION || 'us-east-1';
      const credentials = new AWS.EnvironmentCredentials('AWS');
      const endpoint = new AWS.Endpoint(self.host);
      const req = new AWS.HttpRequest(endpoint, region);
      req.method = request._method;
      req.body = strPayload;
      req.headers['host'] = request._url.host;
      const signer = new AWS.Signers.V4(req, 'managedblockchain');
      signer.addAuthorization(credentials, new Date());
      request.setRequestHeader('Authorization', req.headers['Authorization']);
      request.setRequestHeader('X-Amz-Date', req.headers['X-Amz-Date']);
      request.send(strPayload);
    } catch (error) {
      callback(`[aws-ethjs-provider-http] CONNECTION ERROR: Couldn't connect to node
 '${self.host}': ` +
        `${JSON.stringify(error, null, 2)}`, null);
    }
  }
}
```

2. Copy the contents of the example that follows, and then use a text editor of your choosing to save it to a file named `web3-example-http.js` in the same directory where you saved the provider from the previous step. The example script runs the *`getNodeInfo`* Ethereum method. You can modify the script to include other methods and their parameters.

   Contents of web3-example-http.js

```
import Web3 from 'web3';
import AWSHttpProvider from './aws-http-provider.js';
const endpoint = process.env.AMB_HTTP_ENDPOINT
const web3 = new Web3(new AWSHttpProvider(endpoint));
web3.eth.getNodeInfo().then(console.log);
```

3. Run the script to call the Ethereum API method over HTTP on your Ethereum node.

```
node web3-example-http.js
```

   You should see output similar to the following.

```
Geth/v1.9.24-stable-cc05b050/linux-amd64/go1.15.5
```

To make an Ethereum API call using web3.js over WebSockets to your Ethereum node on Managed Blockchain

1. Copy the contents of the example that follows, and then use a text editor of your choosing to save it to a file named `aws-websocket-provider.js` in the same directory on your client where you run your script.

   Contents of aws-websocket-provider.js

```
////////////////////////////////////////////////////
// Authored by Carl Youngblood
// Senior Blockchain Solutions Architect, AWS
// Adapted from web3 npm package v1.3.0
// licensed under GNU Lesser General Public License
// https://github.com/ethereum/web3.js
////////////////////////////////////////////////////

import AWS from 'aws-sdk';
import WebsocketProvider  from 'web3-providers-ws';
import pkg from 'websocket';
const { w3cwebsocket } = pkg;
const Ws = w3cwebsocket;

export default class AWSWebsocketProvider extends WebsocketProvider {
  connect() {
    const region = process.env.AWS_DEFAULT_REGION || 'us-east-1';
    const credentials = new AWS.EnvironmentCredentials('AWS');
    const host = new URL(this.url).hostname;
    const endpoint = new AWS.Endpoint(`https://${host}/`);
    const req = new AWS.HttpRequest(endpoint, region);
    req.method = 'GET';
    req.body = '';
    req.headers['host'] = host;
    const signer = new AWS.Signers.V4(req, 'managedblockchain');
    signer.addAuthorization(credentials, new Date());
    const headers = {
      'Authorization': req.headers['Authorization'],
      'X-Amz-Date': req.headers['X-Amz-Date'],
      ...this.headers
    }
    this.connection = new Ws(this.url, this.protocol, undefined, headers,
 this.requestOptions, this.clientConfig);
    this._addSocketListeners();
  }
}
```

2. Copy the contents of the following, and then use a text editor of your choosing to save it to a file named `web3-example-ws.js` in the same directory where you saved the provider from the previous step. The example script runs the *`getNodeInfo`* Ethereum method and then closes the connection. You can modify the script to include other methods and their parameters.

   Contents of web3-example-ws.js

```
import Web3 from 'web3';
import AWSWebsocketProvider from "./aws-websocket-provider.js";
const endpoint = process.env.AMB_WS_ENDPOINT
const web3 = new Web3(new AWSWebsocketProvider(endpoint));
web3.eth.getNodeInfo().then(console.log).then(() => {
  web3.currentProvider.connection.close();
});
```

3. Run the script to call the Ethereum API method over WebSockets on your Ethereum node.

```
node web3-example-ws.js
```

You should see output similar to the example that follows.

```
Geth/v1.9.24-stable-cc05b050/linux-amd64/go1.15.5
```

Using the AWS SDK for JavaScript with WebSockets

The example that follows uses a JavaScript file for Node.js to open a WebSockets connection to the Ethereum node endpoint in Managed Blockchain, and then send an Ethereum JSON-RPC API call.

Running the example script requires the following:

- You must have Node.js installed on your machine. If you are using an Amazon EC2 instance, see Tutorial: Setting Up Node.js on an Amazon EC2 Instance.
- You must use the Node package manager (npm) to install the AWS SDK for JavaScript, websocket-client, and ws packages as shown in the example commands that follow. The script uses classes from these packages.

```
npm install websocket-client
```

```
npm install ws
```

```
npm install aws-sdk
```

- The environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` must contain credentials associated with the same account that created the node.

Note
If you added the `"type": "module"` line in your `package.json`, which is necessary for the previous web3 example, the script that follows will fail with the error `require is not defined`. Modify `package.json` to remove or comment out this line before running the script.

To make an Ethereum API call over WebSockets to your Ethereum node on Managed Blockchain

1. Copy the contents of the script that follows and save it to a file on your machine, for example, `ws-ethereum-example.js`. Use a text editor to replace *`my-node-id-lowercase`* with the ID of a node in your account, such as `nd-6eaj5va43jggnpxouzp7y47e4y`, and replace `us-east-1` with the AWS Region in which you created your node.

   The example calls the Ethereum JSON-RPC method `eth_subscribe` along with the `newHeads` parameter. You can replace this method and its parameters with any method listed in Supported JSON-RPC methods (p. 10).

   Contents of ws-ethereum-example.js

```
const AWS = require('aws-sdk');
const WebSocket = require('ws')
const region = 'us-east-1';
const host = 'my-node-id-lowercase.wss.ethereum.managedblockchain.us-
east-1.amazonaws.com';
```

```
const payload = {
  jsonrpc: '2.0',
  method: 'eth_subscribe',
  params: ["newHeads"],
  id: 67
}
const credentials = new AWS.EnvironmentCredentials('AWS');
const endpoint = new AWS.Endpoint(`https://${host}`);
const request = new AWS.HttpRequest(endpoint, region);
request.method = 'GET';
request.body = '';
request.headers['host'] = host;
const signer = new AWS.Signers.V4(request, 'managedblockchain');
signer.addAuthorization(credentials, new Date());
const ws = new WebSocket(`wss://${host}`, {headers: request.headers});
ws.onopen = async () => {
  ws.send(JSON.stringify(payload));
  console.log('Sent request');
}
ws.onerror = (error) => {
  console.error(`WebSocket error: ${error.message}`)
}
ws.onmessage = (e) => {
  console.log(e.data)
}
```

2. Run the script to call the Ethereum API method over WebSockets on your Ethereum node.

```
node ws-ethereum-example.js
```

The `eth_subscribe` method with the `newHeads` parameter generates a notification each time a new header is appended to the chain. You should see output similar to the example that follows, with additional subsequent responses. The WebSockets connection remains open and additional notifications appear until you cancel the command.

```
sent request
{"id":67,"jsonrpc":"2.0","result":"0xabcd123456789efg0h123ijk45l6m7n8"}
```

Using awscurl to make JSON-RPC calls over HTTP

The example that follows uses awscurl, which sends a signed HTTP request based on the current credentials you have set for the AWS CLI. If you construct your own HTTP requests, see Signing AWS requests with Signature Version 4 in the *AWS General Reference*.

Replace *my-node-id-lowercase* with the ID of a node in your account such as `nd-6eaj5va43jggnpxouzp7y47e4y`. The example calls the `web3_clientVersion` method, which takes an empty parameter block. You can replace this method and its parameters with any method listed in Supported JSON-RPC methods (p. 10).

```
awscurl --service managedblockchain \
-X POST -d '{"jsonrpc": "2.0", "method": "web3_clientVersion", "params": [], "id": 67}' \
https://my-node-id-lowercase.ethereum.managedblockchain.us-east-1.amazonaws.com
```

The command returns output similar to the following.

```
{"jsonrpc":"2.0","id":67,"result":"Geth/v1.9.22-stable-c71a7e26/linux-amd64/go1.15.5"}
```

Ethereum on Amazon Managed Blockchain Security

To provide data protection as well as authentication and access control, Amazon Managed Blockchain benefits from AWS features and the features of the open-source framework running on Managed Blockchain.

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

Topics

Data protection for Ethereum on Amazon 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*.

Encryption at rest

Amazon Managed Blockchain offers fully managed encryption at rest. Managed Blockchain encryption at rest provides enhanced security by encrypting all data at rest on Ethereum nodes using Managed Blockchain owned encryption 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 encrypt your tables. A Managed Blockchain owned key is used to encrypt data at rest by default at no additional cost. No configuration is required. Using an AWS managed encryption key is not supported. For more information, see AWS owned keys in the *AWS Key Management Service Developer Guide*.

Encryption in transit

By default, Managed Blockchain uses an HTTPS/TLS connection to encrypt all data transmitted from a client computer running the AWS CLI to AWS service endpoints.

You don't need to do anything to enable the use of HTTPS/TLS. It is always enabled unless you explicitly disable it for an individual AWS CLI command by using the --no-verify-ssl command line option.

Authentication and access control for Ethereum on Amazon Managed Blockchain

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.

Before you configure IAM permissions, see Identity and Access Management for Ethereum on Amazon Managed Blockchain (p. 25). We also recommend What is IAM? and IAM JSON Policy Reference in the *IAM User Guide*.

Identity and Access Management for Ethereum 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 Ethereum on Amazon Managed Blockchain identity and access (p. 38).

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 Ethereum on Amazon Managed Blockchain works with IAM (p. 29).

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 Ethereum on Amazon Managed Blockchain identity-based policy examples (p. 31).

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 Ethereum on Amazon Managed Blockchain works with IAM

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

> **Important**
> All Ethereum JSON-RPC API calls to an Ethereum node on Managed Blockchain are authenticated using the Signature Version 4 signing process. This means that only authorized IAM principals in the AWS account that created the node can interact with it using the API. AWS credentials (an access key ID and secret access key) must be provided with the call. IAM policies do not apply to API calls and can not be used to allow or deny Ethereum API access to a node. IAM permissions policies only apply to operations related to node creation, deletion, and management.

Topics

Ethereum 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. Ethereum 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 Ethereum 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. Ethereum 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
- node

Nodes are associated with your account. Networks are associated with Ethereum public networks and are not associated with AWS Regions.

For example an Ethereum public network resource on Managed Blockchain has one of the following ARNs.

```
arn:aws:managedblockchain:::networks/n-ethereum-mainnet
```

```
arn:aws:managedblockchain:::networks/n-ethereum-ropsten
```

```
arn:aws:managedblockchain:::networks/n-ethereum-ropsten
```

To see a list of Ethereum 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

Ethereum 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 Ethereum on Managed Blockchain identity-based policies, see Ethereum on Amazon Managed Blockchain identity-based policy examples (p. 31).

Ethereum on Managed Blockchain Resource-Based Policies

Ethereum on Managed Blockchain does not support resource-based policies.

Authorization based on Ethereum on Managed Blockchain tags

You can attach tags to Ethereum 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 Ethereum on Managed Blockchain resources, see Tagging Amazon Managed Blockchain resources (p. 40).

To view example identity-based policies for allowing or denying access to resources and actions based on tags, see Controlling access using tags (p. 34).

Ethereum on Amazon Managed Blockchain identity-based policy examples

By default, IAM users and roles don't have permission to create or modify Ethereum on 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 Ethereum on Managed Blockchain Console

To access the Ethereum on Amazon Managed Blockchain console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Ethereum on 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 Ethereum on 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 available actions for Ethereum on Managed Blockchain

This example grants an IAM user in your AWS account access to list all Ethereum networks, to create and list nodes on all those networks, and to get and delete nodes in `111122223333`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "WorkWithEthereumNetworks",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:ListNetworks",
                "managedblockchain:GetNetwork"
            ],
            "Resource": [
                "arn:aws:managedblockchain:::networks/n-ethereum-mainnet",
                "arn:aws:managedblockchain:::networks/n-ethereum-ropsten",
                "arn:aws:managedblockchain:::networks/n-ethereum-rinkeby"
            ]
        },
        {
            "Sid": "CreateAndListEthereumNodes",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:CreateNode",
                "managedblockchain:ListNodes"
            ],
            "Resource": [
                "arn:aws:managedblockchain:::networks/*"
            ]
        },
        {
            "Sid": "ManageEthereumNodes",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:GetNode",
                "managedblockchain:DeleteNode"
```

```
            ],
            "Resource": [
                "arn:aws:managedblockchain:*:111122223333:nodes/*"
            ]
        }
    ]
}
```

Controlling access using tags

The following example policies demonstrate how you can use tags to limit access to Ethereum on Managed Blockchain resources and actions performed on those resources.

> **Note**
> This topic includes examples of policy statements with a `Deny` effect. These policies assume that other policies with `Allow` effect for those actions exist with broader applicability. The `Deny` policy statement is being used to restrict that otherwise overly-permissive allow statement.

Example – Deny access to networks with a specific tag key

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

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

Example – Deny node creation on networks that have a specific tag and value

The following identity-based policy statement denies the IAM principal the ability to create a node on an Ethereum public network tagged in the AWS account with the tag key of `department` and the value `accounting`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyCreateNodeForNetworkWithTag",
            "Effect": "Deny",
            "Action": [
                "managedblockchain:CreateNode"
            ],
```

```
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/department": [
                        "accounting"
                    ]
                }
            }
        }
    ]
}
```

Example – Require a specific tag key and value to be added when a node is created

The following identity-based policy statements allow an IAM principal to create a node for the AWS account 111122223333 only if a key with the tag key of `department` and a value of `accounting` is added during creation.

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

Example – Deny listing nodes for networks that have a specific tag key and value

The following identity-based policy statement denies the IAM principal the ability to list nodes on an Ethereum public network tagged in the AWS account with the tag key of `department` and the value `accounting`.

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

```
        {
            "Sid": "DenyListNodesForNetworkWithTag",
            "Effect": "Deny",
            "Action": [
                "managedblockchain:ListNodes"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/department": [
                        "accounting"
                    ]
                }
            }
        }
    ]
}
```

Example – Deny retrieving and viewing node information for nodes with a specific tag key and value

The following identity-based policy statement denies the IAM principal the ability to view node information for nodes that have a tag with the tag key of `department` and the value `accounting`.

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

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 Ethereum 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 Ethereum on Managed Blockchain and IAM.

Topics

I Am Not Authorized to Perform an Action in Ethereum 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 an Ethereum node on the Ropsten testnet, but does not have `managedblockchain:CreateNode` permissions.

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

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

I'm an administrator and want to allow others to access Ethereum 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*.

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. 34) 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 Ethereum on Managed Blockchain resources

You can tag the following resources:

- Networks
- Nodes

Tags that you create for Ethereum public networks 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

Using the Managed Blockchain console, you can add a tag to an Ethereum node when you create it or when viewing node details. You can remove a tag when viewing node details. For more information, see Working with Ethereum nodes using Managed Blockchain (p. 5).

Managed Blockchain allows you to tag public Ethereum networks after you create a node on the network using Managed Blockchain.

To add or remove a tag for an Ethereum network using the AWS Management Console

1. Open the Managed Blockchain console at https://console.aws.amazon.com/managedblockchain/.
2. If the console doesn't open to the **Networks** list, choose **Networks** from the navigation pane.
3. Choose the network from the list.
4. 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**.

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 an Ethereum network from the list.
3. Under **Nodes**, choose a **Node ID** from the list.
4. 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**.

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 as management events 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. Managed Blockchain supports logging management events. For more information, see Logging management events for trails in the *AWS CloudTrail User Guide*. Managed Blockchain also supports logging data events for Ethereum JSON-RPC calls over HTTP or WebSockets. For more information, see Using CloudTrail to track Ethereum JSON-RPC calls (p. 44).

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.

Example – Management event log entry

The following example shows a CloudTrail management event 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",
      "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-ethereum-ropsten",
    "nodeId": "nd-6EAJ5VA43JGGNPXOUZP7Y47E4Y"
  },
  "responseElements": null,
  "requestID": "1e2xa3m4-56p7-8l9e-0ex1-23456a78m90p",
  "eventID": "ex12345a-m678-901p-23e4-567ex8a9mple",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```


Using CloudTrail to track Ethereum JSON-RPC calls

You can track JSON-RPC as *data events* using CloudTrail. Data events are not logged by default when you create a trail. To record Ethereum JSON-RPC calls as CloudTrail data events, you must explicitly

add the supported resources or resource types for which you want to collect activity to a trail. Managed Blockchain supports adding data events using the AWS CLI. For more information, see Log events by using advanced selectors in the *AWS CloudTrail User Guide*.

To log data events for a trail, run the put-event-selectors command after you create the trail. Use the `--advanced-event-selectors` option to specify the data events to log. The following example demonstrates a `put-event-selectors` command that logs all Ethereum JSON-RPC calls for a trail named *`my-ethereum-trail`* in the *`us-east-1`* Region.

```
aws cloudtrail put-event-selectors \
--region us-east-1 \
--trail-name my-ethereum-trail \
--advanced-event-selectors '[{
    "Name": "MyDataEventSelectorForEtherumJsonRpcCalls",
    "FieldSelectors": [
      { "Field": "eventCategory", "Equals": ["Data"] },
      { "Field": "resources.type", "Equals": ["AWS::ManagedBlockchain::Node"] } ]}]'
```

Example – Data event log entry

The following example demonstrates a CloudTrail data event log entry for an Ethereum JSON-RPC call, `web3_clientVersion`, from a client to a node in Managed Blockchain.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "ABCD1EF23G4EXAMPLE56:carlossalazar",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/carlossalazar",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-12-11T16:51:12Z"
      }
    }
  },
  "eventTime": "2020-12-11T19:56:36Z",
  "eventSource": "managedblockchain.amazonaws.com",
  "eventName": "web3_clientVersion",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "198.51.100.1",
  "userAgent": "python-requests/2.23.0",
  "requestParameters": {
    "id": 67,
    "jsonrpc": "2.0",
    "method": "web3_clientVersion",
    "params": []
  },
  "responseElements": {
    "result": "Geth/v1.9.24-stable-cc05b050/linux-amd64/go1.15.5",
    "id": 67,
    "jsonrpc": "2.0"
  },
  "requestID": "1e2xa3m4-56p7-8l9e-0ex1-23456a78m90p",
  "eventID": "ex12345a-m678-901p-23e4-567ex8a9mple",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

Document history

The following table describes important additions to the *Ethereum on Amazon Managed Blockchain Developer Guide*.

update-history-change	update-history-description	update-history-date
Initial Release	Initial release.	December 15, 2020