

# Contents

## [Azure Blockchain Service \(Preview\)](#)

### [Overview](#)

#### [About Azure Blockchain Service](#)

### [Quickstarts](#)

#### [Create member - Portal](#)

#### [Create member - CLI](#)

#### [Connect to transaction node](#)

##### [Using MetaMask](#)

##### [Using Geth](#)

##### [Using Truffle](#)

### [Tutorials](#)

#### [Send transactions](#)

### [Concepts](#)

#### [Consortium](#)

#### [Data access and security](#)

#### [Develop](#)

#### [Limits](#)

#### [Patching, updating, versioning](#)

### [How-to guides](#)

#### [Configure Azure AD access](#)

#### [Configure transaction nodes](#)

#### [Manage using Azure CLI](#)

#### [Manage consortium using PowerShell](#)

### [Resources](#)

#### [Blockchain REST API](#)

#### [Blog](#)

#### [Azure Blockchain Service Visual Studio Code extension](#)

#### [Code samples](#)

#### [Community](#)

[Forum](#)

[Product feedback](#)

[Pricing](#)

[Region availability](#)

[Stack Overflow](#)

# What is Azure Blockchain Service?

5/10/2019 • 5 minutes to read • [Edit Online](#)

Azure Blockchain Service is a fully managed ledger service that enables users the ability to grow and operate blockchain networks at scale in Azure. By providing unified control for both infrastructure management as well as blockchain network governance, Azure Blockchain Service provides:

- Simple network deployment and operations
- Built-in consortium management
- Develop smart contracts with familiar development tools

Azure Blockchain Service is designed to support multiple ledger protocols. Currently, it provides support for the Ethereum [Quorum](#) ledger using the [IBFT](#) consensus mechanism.


These capabilities require almost no administration and all are provided at no additional cost. You can focus on app development and business logic rather than allocating time and resources to managing virtual machines and infrastructure. In addition, you can continue to develop your application with the open-source tools and platform of your choice to deliver your solutions without having to learn new skills.


## Network deployment and operations

Deploying Azure Blockchain Service can be done through the Azure portal, Azure CLI as well as through Visual Studio code using the Azure Blockchain extension. Deployment is simplified, including provisioning both transaction and validator nodes, Azure Virtual Networks for security isolation as well as service-managed storage. In addition, when deploying a new blockchain member, users also create, or join, a consortium. Consortia enable multiple parties in different Azure subscriptions to be able to securely communicate with one another on a shared blockchain. This simplified deployment reduces blockchain network deployment from days to minutes.

### Performance and service tiers

The Azure Blockchain Service offers two service tiers: *Basic* and *Standard*. Each tier offers different performance and capabilities to support lightweight development and test workloads up to massively scaled production blockchain deployments. Both tiers include at least one transaction node, and one validator node (Basic) or two validator nodes (Standard).


**Select a pricing tier**  
 PREVIEW


 Changing the pricing tier after member creation is not currently supported.

	<input type="radio"/> <b>Basic</b> Environment for dev/test	<input checked="" type="radio"/> <b>Standard</b> Run production workloads
Compute	1 vCore	2 vCores
Storage <sup>1</sup>	5 GB	5 GB
Number of validator nodes	1	2
Number of transaction nodes <sup>2</sup>	1	1
Hybrid deployment support	N/A	Coming soon
High availability	N/A	99.9%
<b>Estimated cost per month <sup>3</sup></b>	<cost> <cost> per node	<cost> <cost> per node

In addition to offering two validator nodes, the *Standard* tier provides 2 vCores for each transaction and validator node whereas the *Basic* tier offers a 1 vCore configuration. By offering 2 vCores for transaction and validator nodes, 1 vCore can be dedicated to the Quorum ledger while the remaining 1 vCore can be used for other infrastructure-related services, ensuring optimal performance for production blockchain workloads. For more information on pricing details, see [Azure Blockchain Service pricing](#).

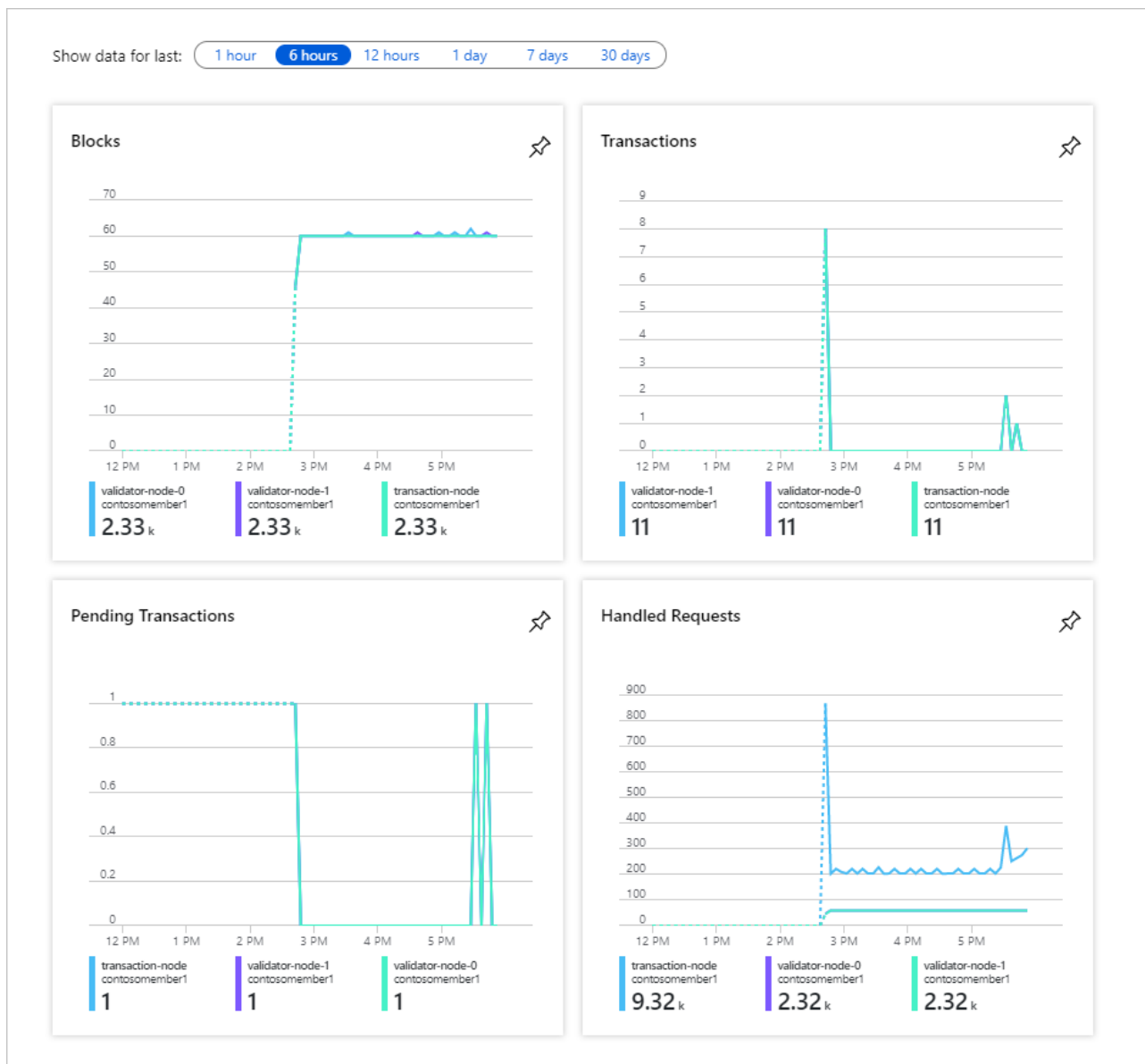
### Security and maintenance

After provisioning your first blockchain member, you have the ability to add additional transaction nodes to your member. By default, transaction nodes are secured through firewall rules and will need to be configured for access. Additionally, all transaction nodes encrypt data in motion via TLS. Multiple options exist for securing transaction node access, including firewall rules, basic authentication, access keys as well as Azure Active Directory integration. For more information, see [configure transaction nodes](#) and [configure Azure Active Directory access](#).

As a managed service, Azure Blockchain Service ensures that your blockchain member's nodes are patched with the latest host operating system and ledger software stack updates, configured for high-availability (Standard tier only), eliminating much of the DevOps required for traditional IaaS blockchain nodes. For more information on patching and updates, see [supported Azure Blockchain Service ledger versions](#).

### Monitoring and logging

In addition, Azure Blockchain Service provides rich metrics through Azure Monitor Service providing insights into nodes' CPU, memory and storage usage, as well as helpful insights into blockchain network activity such as transactions and blocks mined, transaction queue depth, as well as active connections. Metrics can be customized to provide views into the insights that are important to your blockchain application. In addition, thresholds can be defined through alerts enabling users to trigger actions such as sending an email or text message, running a Logic App, Azure Function or sending to a custom-defined webhook.



Through Azure Log Analytics, users can view logs related to the Quorum ledger, or other important information such as attempted connections to the transaction nodes.

## Built-in consortium management

When deploying your first blockchain member, you either join or create a new consortium. A consortium is a logical group used to manage the governance and connectivity between blockchain members who transact in a multi-party process. Azure Blockchain Service provides built-in governance controls through pre-defined smart contracts, which determine what actions members in the consortium can take. These governance controls can be customized as necessary by the administrator of the consortium. When you create a new consortium, your blockchain member is the default administrator of the consortium, enabling the ability to invite other parties to join your consortium. You can join a consortium only if you have been previously invited. When joining a consortium, your blockchain member is subject to the governance controls put in place by the consortium's administrator.

Home > consortium1member1 > consortiumdemo1

**consortiumdemo1**  
PREVIEW

Invite Refresh Remove Edit

⚠ Functionality related to consortium management can be done through PowerShell.

NAME	DISPLAY NAME	SUBSCRIPTION ID	ROLE	STATUS	JOIN DATE	DATE MODIFIED
consortium1member1	consortium1member1	<Subscription ID>	admin	Active	4/26/2019	4/26/2019
consortium1member2	consortium1member2	<Subscription ID>	admin	Active	4/26/2019	4/26/2019
consortium1member3	consortium1member3	<Subscription ID>	user	Active	4/26/2019	4/26/2019
consortium1member4		<Subscription ID>		Active	4/26/2019	4/26/2019

Consortium management actions such as adding and removing members from a consortium can be accessed through PowerShell and a REST API. You can programmatically manage a consortium using common interfaces rather than modifying and submitting solidity-based smart contracts. For more information, see [consortium management](#).

## Develop using familiar development tools

Based on the open-sourced Quorum Ethereum ledger, you can develop applications for Azure Blockchain Service the same way as you do for existing Ethereum applications. Working with leading industry partners, the Azure Blockchain Development Kit Visual Studio Code extension allows developers to leverage familiar tools like Truffle Suite to build smart contracts. Using the Azure Blockchain Development Kit extension, developers can create, or connect to an existing consortium so that you can build and deploy your smart contracts all from one IDE. Using the Azure Blockchain Visual Studio Code extension, you can create or connect to an existing consortium so that you can build and deploy your smart contracts all from one IDE. For more information, see [Azure Blockchain Development Kit in the VS Code marketplace](#) and the [Azure Blockchain Development Kit user guide](#).

## Support and feedback

Need help or have feedback?

- Visit the [Azure Blockchain blog](#), [Microsoft Tech Community](#), and [Azure Blockchain forum](#).
- To provide feedback or to request new features, create an entry via [UserVoice](#).

## Next steps

To get started, try a quickstart or find out more details from these resources.

- [Create a blockchain member using the Azure portal](#) or [create a blockchain member using Azure CLI](#)
- For cost comparisons and calculators, see the [pricing page](#).
- Build your first app using the [Azure Blockchain Development Kit](#)
- Azure Blockchain VSCode Extension [user guide](#)

# Quickstart: Create an Azure Blockchain Service using the Azure portal

5/30/2019 • 2 minutes to read • [Edit Online](#)

Azure Blockchain Service is a blockchain platform that you can execute your business logic within a smart contract. This quickstart shows you how to get started by creating a managed ledger using the Azure portal.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Create a managed ledger

Azure Blockchain Service is created with a defined set of compute and storage resources.

1. Sign in to the [Azure portal](#).
2. Select **Create a resource** in the upper left-hand corner of the Azure portal.
3. Select **Blockchain > Azure Blockchain Service**.
4. Complete the template.

Create a blockchain member - M x

←

→

↻

https://portal.azure.com/?feature.customportal=false&microsoft\_a...

🔍

☆

🔥

👤

⋮

Microsoft Azure

🔍

>

📄

🔔1

⚙️

?

😊

admin@contoso.com

CONTOSO

»

Home > Create a blockchain member

Create a blockchain member

PREVIEW

×

Basics

Tags

Review + create

Create a blockchain member that runs the Quorum ledger protocol in a new or existing consortium. Complete the Basics tab then Review + create to provision a blockchain member with default parameters or review each tab for full customization. [Learn more about Azure Blockchain Service](#)

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

\* Subscription ⓘ

Visual Studio Enterprise

▼

\* Resource group ⓘ

myResourceGroup

▼

[Create new](#)

\* Region ⓘ

East US

▼

BLOCKCHAIN DETAILS

Select the protocol and consortium that you've been invited to join or create your own consortium to start. You can invite others to join later.

\* Consortium ⓘ

(New) contosofood

▼

[Create new](#)

\* Protocol ⓘ

Quorum

▼

MEMBER DETAILS

\* Name ⓘ

myblockchainmember

✓

\* Member account password ⓘ

.....

✓

\* Pricing ⓘ

Standard - 2 vCores

2 validator nodes, 1 transaction node

Estimated cost 710.03 USD/month

[Change](#)

\* Transaction node password ⓘ

.....

✓

Review + create

Previous

Next: Tags

SETTING	DESCRIPTION
Blockchain Member	Choose a unique name that identifies your Azure Blockchain Service member. The blockchain member name can only contain lowercase letters and numbers. The first character must be a letter. The value must be between 2 and 20 characters long.
Subscription	Select the Azure subscription that you want to use for your service. If you have multiple subscriptions, choose the subscription in which you get billed for the resource.



SETTING	DESCRIPTION
Resource group	A new resource group name or an existing one from your subscription.
Region	Location must be the same for all members of the consortium.
Member account password	The member account password is used to encrypt the private key for the Ethereum account that is created for your member. You use the member account and member account password for consortium management.
Consortium name	For a new consortium, enter a unique name. If joining a consortium through an invite, the value is the consortium you are joining.
Description	Description of the consortium.
Protocol	Preview supports the Quorum protocol.
Pricing	The node configuration for your new service. Select <b>Standard</b> . 2 validator nodes and 1 transaction node are the default settings.
Transaction node password	The password for the member's default transaction node. Use the password for basic authentication when connecting to blockchain member's default transaction node public endpoint.

5. Select **Create** to provision the service. Provisioning takes about 10 minutes.
6. Select **Notifications** on the toolbar to monitor the deployment process.
7. After deployment, navigate to your blockchain member.

Select **Overview**, you can view the basic information about your service including the RootContract address and member account.

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes the 'Microsoft Azure' logo, a search bar, and user information for 'admin@contoso.com'. The left sidebar contains a navigation menu with options like Home, Overview, Activity log, Access control (IAM), Tags, Settings, Properties, Locks, and Export template. The main content area displays the 'Overview' page for a resource named 'myblockchainmember' (Azure Blockchain Service - PREVIEW). The page shows a search bar, a 'Delete' button, and a table of properties:

Resource group	myResourceGroup	Member name	myblockchainmember
Status	Available	Protocol	Quorum
Location	East US	Pricing Tier	Standard (2 vCores, 3 nodes)
Subscription	Visual Studio Enterprise	Consortium	contosofood
Subscription ID	<Subscription ID>	RootContract address	0xb255f55e8d600f09ebc1035dd2118ace5ca1ab1e
		Member account	0x528cc9d5743ef315c1792cf90ad213ac5ca1ab1e

## Clean up resources

You can use the member you created for the next quickstart or tutorial. When no longer needed, you can delete

the resources by deleting the `myResourceGroup` resource group you created by the Azure Blockchain Service.

To delete the resource group:

1. In the Azure portal, navigate to **Resource group** in the left navigation pane and select the resource group you want to delete.
2. Select **Delete resource group**. Verify deletion by entering the resource group name and select **Delete**.

## Next steps

[Use MetaMask to connect and deploy a smart contract](#)

# Quickstart: Create an Azure Blockchain Service blockchain member using Azure CLI

5/30/2019 • 2 minutes to read • [Edit Online](#)

Azure Blockchain Service is a blockchain platform you can use to execute your business logic within a smart contract. This quickstart shows you how to get started by creating a blockchain member using Azure CLI.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to <https://shell.azure.com/bash>. Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press enter to run it.

If you prefer to install and use the CLI locally, this quickstart requires Azure CLI version 2.0.51 or later. Run `az --version` to find the version. If you need to install or upgrade, see [install Azure CLI](#).

## Create a resource group

Create a resource group with the `az group create` command. An Azure resource group is a logical container into which Azure resources are deployed and managed. The following example creates a resource group named *myResourceGroup* in the *eastus* location:

```
az group create --name myResourceGroup --location eastus
```

## Create a blockchain member

Create a blockchain member in Azure Blockchain Service that runs the Quorum ledger protocol in a new consortium. There are several parameters and properties you need to pass. Replace the example parameters with your values.

```
az resource create --resource-group myResourceGroup --name myblockchainmember --resource-type Microsoft.Blockchain/blockchainMembers --is-full-object --properties "{ \"location\": \"eastus\", \"properties\": {\"password\": \"strongMemberAccountPassword@1\", \"protocol\": \"Quorum\", \"consortium\": \"myConsortiumName\", \"consortiumManagementAccountPassword\": \"strongConsortiumManagementPassword@1\", \"sku\": { \"name\": \"S0\" } } }
```

PARAMETER	DESCRIPTION
<b>resource-group</b>	Resource group name where Azure Blockchain Service resources are created. Use the resource group you created in the previous section.

PARAMETER	DESCRIPTION
<b>name</b>	A unique name that identifies your Azure Blockchain Service blockchain member. The name is used for the public endpoint address. For example, <code>myblockchainmember.blockchain.azure.com</code> .
<b>location</b>	Azure region where the blockchain member is created. For example, <code>eastus</code> . Choose the location that is closest to your users or your other Azure applications.
<b>password</b>	The password for the member's default transaction node. Use the password for basic authentication when connecting to blockchain member's default transaction node public endpoint.
<b>consortium</b>	Name of the consortium to join or create.
<b>consortiumAccountPassword</b>	The consortium account password is also known as the member account password. The member account password is used to encrypt the private key for the Ethereum account that is created for your member. You use the member account and member account password for consortium management.
<b>skuName</b>	Tier type. Use S0 for Standard and B0 for Basic.

It takes about 10 minutes to create the blockchain member and supporting resources.

## Clean up resources

You can use the blockchain member you created for the next quickstart or tutorial. When no longer needed, you can delete the resources by deleting the `myResourceGroup` resource group you created by the Azure Blockchain Service.

Run the following command to remove the resource group and all related resources.

```
az group delete --name myResourceGroup --yes
```

## Next steps

Now that you have created a blockchain member, try one of the connection quickstarts for [Geth](#), [MetaMask](#), or [Truffle](#).

[Use Truffle to connect to a an Azure Blockchain Service network](#)

# Quickstart: Use MetaMask to connect and deploy a smart contract

5/2/2019 • 3 minutes to read • [Edit Online](#)

In this quickstart, you'll use MetaMask to connect to an Azure Blockchain Service network and use Remix to deploy a smart contract. Metamask is a browser extension to manage an Ether wallet and perform smart contract actions.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

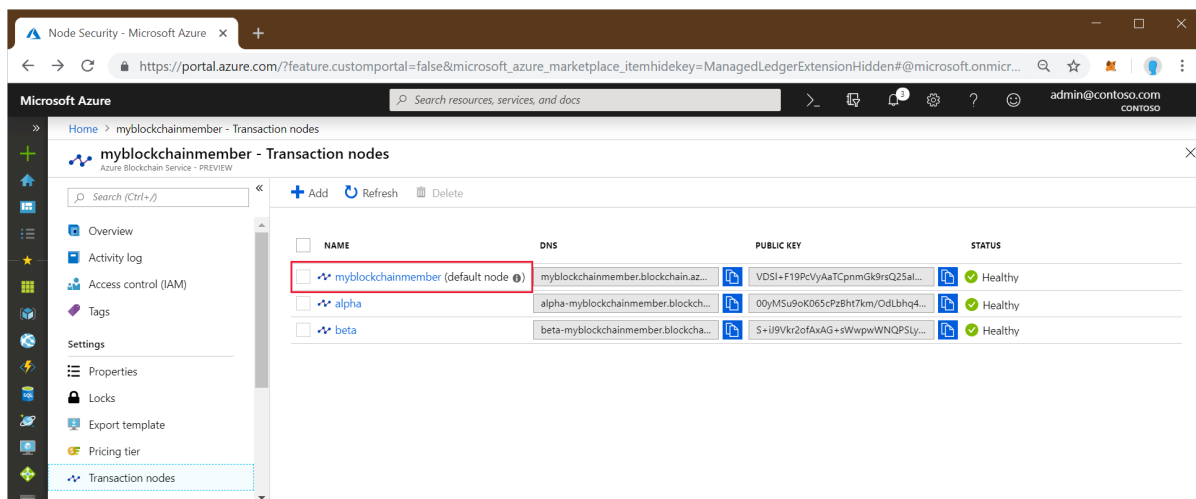
## Prerequisites

- [Create an Azure Blockchain member](#)
- Install [MetaMask browser extension](#)
- Generate a MetaMask [wallet](#)

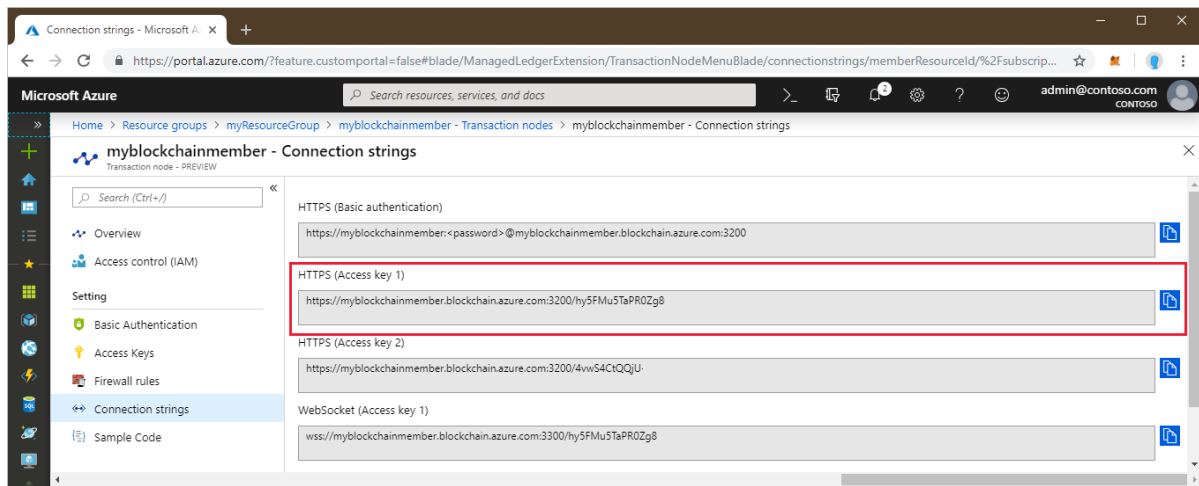
## Get endpoint address

You need the Azure Blockchain Service endpoint address to connect to the blockchain network. You can find the endpoint address and access keys in the Azure portal.

1. Sign in to the [Azure portal](#).
2. Navigate to your Azure Blockchain Service member. Select **Transaction nodes** and the default transaction node link.

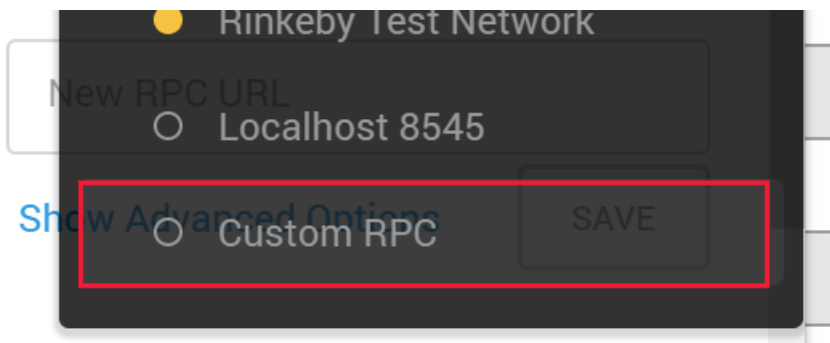


3. Select **Connection strings > Access keys**.
4. Copy the endpoint address from **HTTPS (Access key 1)**. You need the address for the next section.



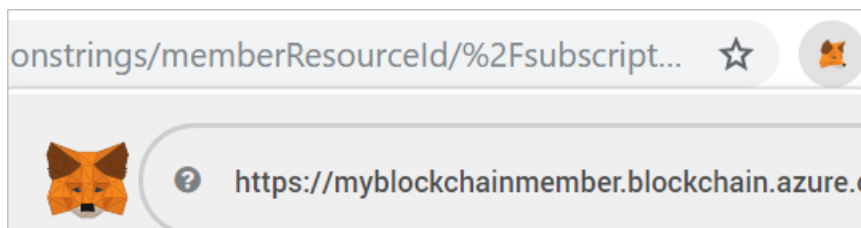
## Connect MetaMask

1. Open MetaMask browser extension and sign in.
2. In the network dropdown, select **Custom RPC**.



3. In **New Network > New RPC URL**, enter your endpoint address copied from the previous section.
4. Select **Save**.

If connection was successful, the private network is displayed in the network dropdown.

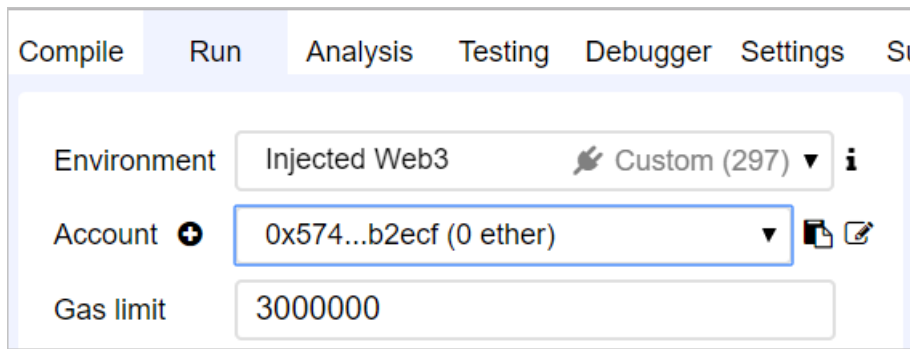


## Deploy smart contract

Remix is a browser-based Solidity development environment. Using MetaMask and Remix together, you can deploy and take actions on smart contracts.

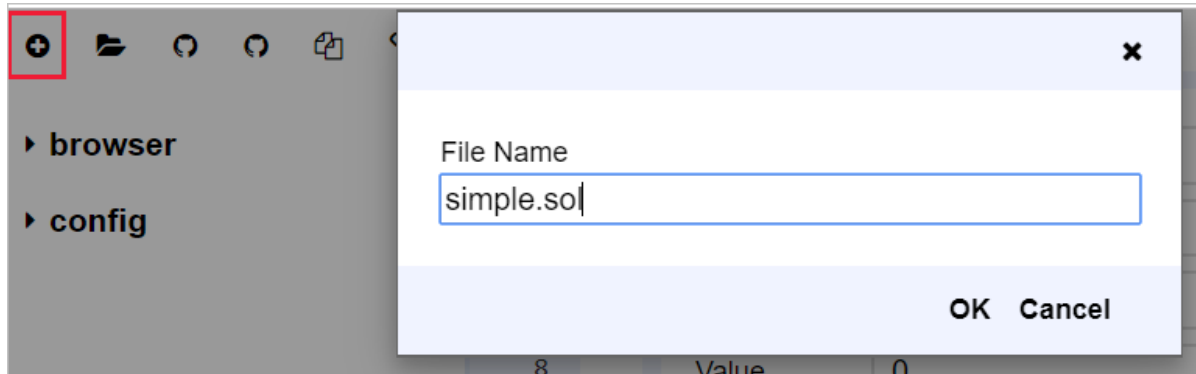
1. In your browser, navigate to `https://remix.ethereum.org`.
2. Select **Run**.

MetaMask sets your **Environment** to **Injected Web3** and **Account** to your network.



3. Select **Create new file**.

Name the new file `simple.sol`.



Select **OK**.

4. In the Remix editor, paste in the following **simple smart contract** code.

```
pragma solidity ^0.5.0;

contract simple {
    uint balance;

    constructor() public{
        balance = 0;
    }

    function add(uint _num) public {
        balance += _num;
    }

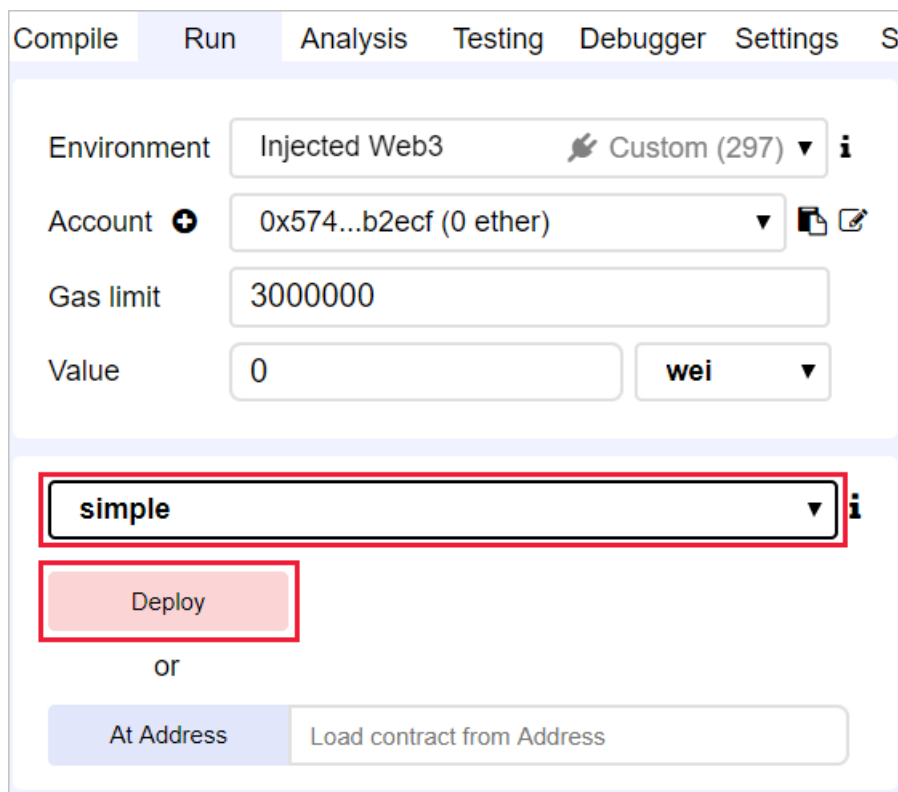
    function get() public view returns (uint){
        return balance;
    }
}
```

The **simple contract** declares a state variable named **balance**. There are two functions defined. The **add** function adds a number to **balance**. The **get** function returns the value of **balance**.

5. To compile the contract, select the **Compile > Start to compile**. If successful, a green box with contract name is displayed.



6. To execute the contract, select the **Run** tab. Select the **simple** contract then **Deploy**.



7. A MetaMask notification is displayed alerting you of insufficient funds to perform the transaction.

For a public blockchain network, you would need Ether to pay for the transaction cost. Since this is a private network in a consortium, you can set gas price to zero.

8. Select **Gas Fee > Edit > Advanced**, set the **Gas Price** to 0.



MetaMask Notification

Customize Gas

Close

Basic

Advanced

New Transaction Fee

~Transaction Time

0 ETH

...

Gas Price (GWEI)

i

0

^

v

Gas Limit

i

121729

^

v

Gas Price Extremely Low

Chart only available on Ethereum networks.

Select **Save**.

9. Select **Confirm** to deploy the smart contract to the blockchain.
10. In the **Deployed Contracts** section, expand the **simple** contract.

Compile Run Analysis Testing Debugger Settings Su

Environment Injected Web3 Custom (297) i

Account + 0x574...b2ecf (0 ether) v i

Gas limit 3000000

Value 0 wei v

simple v i

Deploy

or

At Address Load contract from Address

Transactions recorded: 1 v

Deployed Contracts i

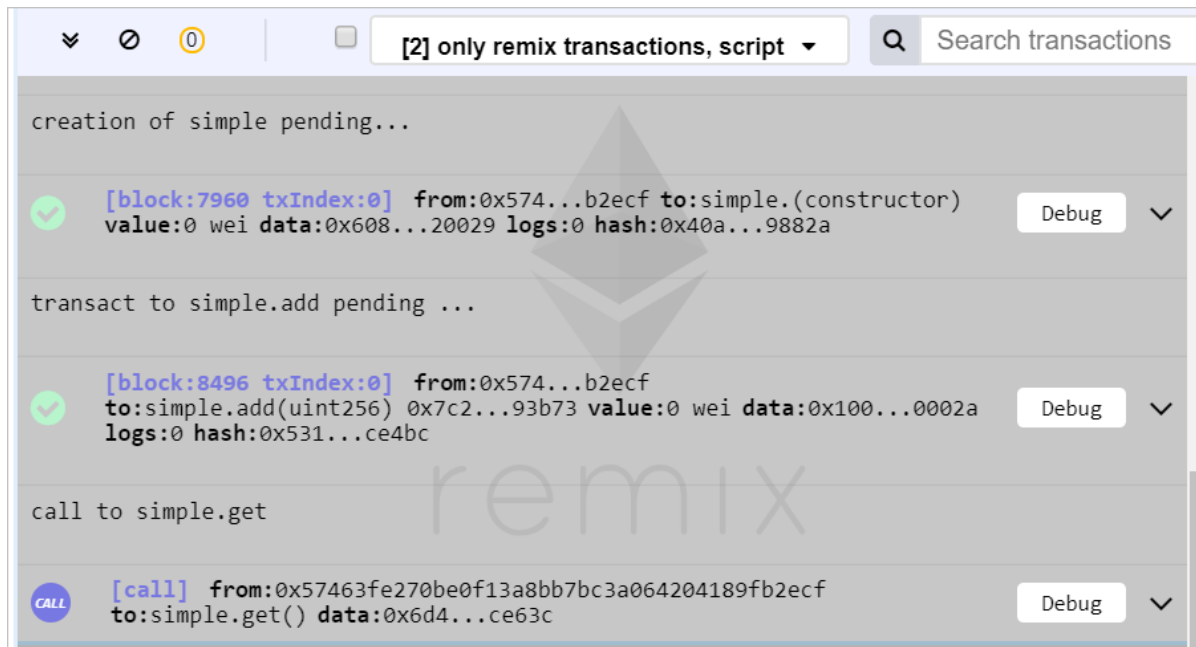
simple at 0x7c2...93b73 (blockchain) i x

add uint256 \_num v

get

There are two actions **add** and **get** that map to the functions defined in the contract.

11. To perform an **add** transaction on the blockchain, enter a number to add then select **add**.
  12. Similar to when you deployed the contract, a MetaMask notification is displayed alerting you of insufficient funds to perform the transaction.
- Since this is a private network in a consortium, we can set gas price to zero.
13. Select **Gas Fee > Edit > Advanced**, set the **Gas Price** to 0, and select **Save**.
  14. Select **Confirm** to perform the transaction on the blockchain.
  15. Select **get** action. This is a call to query node data. A transaction isn't needed.
  16. In the debug pane of Remix, you can see details about the transactions on the blockchain.



You can see the **simple** contract creation, transaction for **simple.add**, and call to **simple.get**.

17. You can also see transaction history in MetaMask. Open the MetaMask browser extension.

18. In the **History** section, you can see a log of the deployed contract and transactions.

## Next steps

In this quickstart, you used the MetaMask browser extension to connect to an Azure Blockchain Service transaction node, deploy a smart contract, and send a transaction to the blockchain. Try the next tutorial to deploy and send a transaction using Truffle.

[Send a transaction](#)

# Quickstart: Use Geth to connect to a transaction node

5/2/2019 • 2 minutes to read • [Edit Online](#)

Geth is a Go Ethereum client you can use to attach to a Geth instance on an Azure Blockchain Service transaction node.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

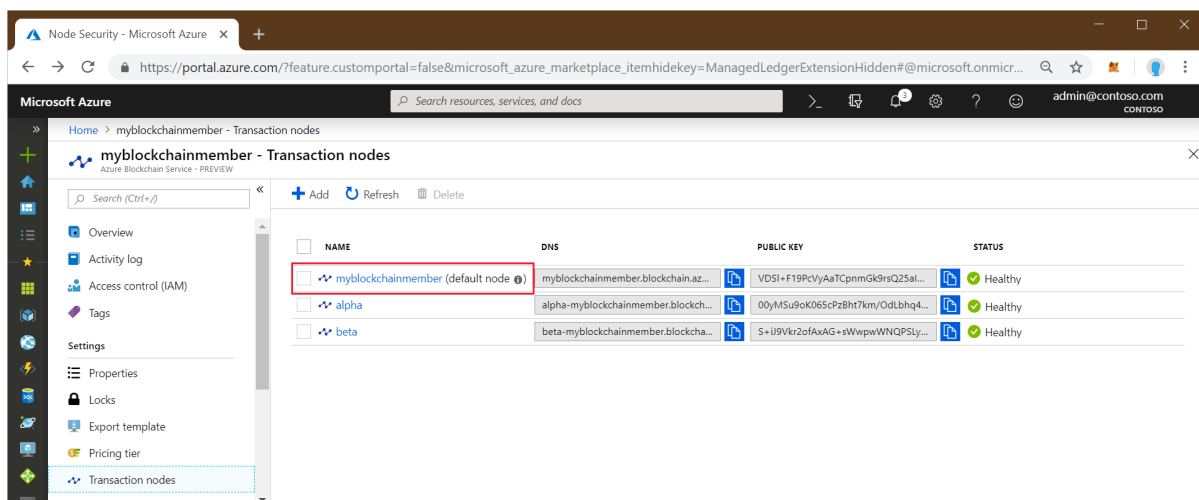
## Prerequisites

- Install [Geth](#)
- [Create an Azure Blockchain member](#)

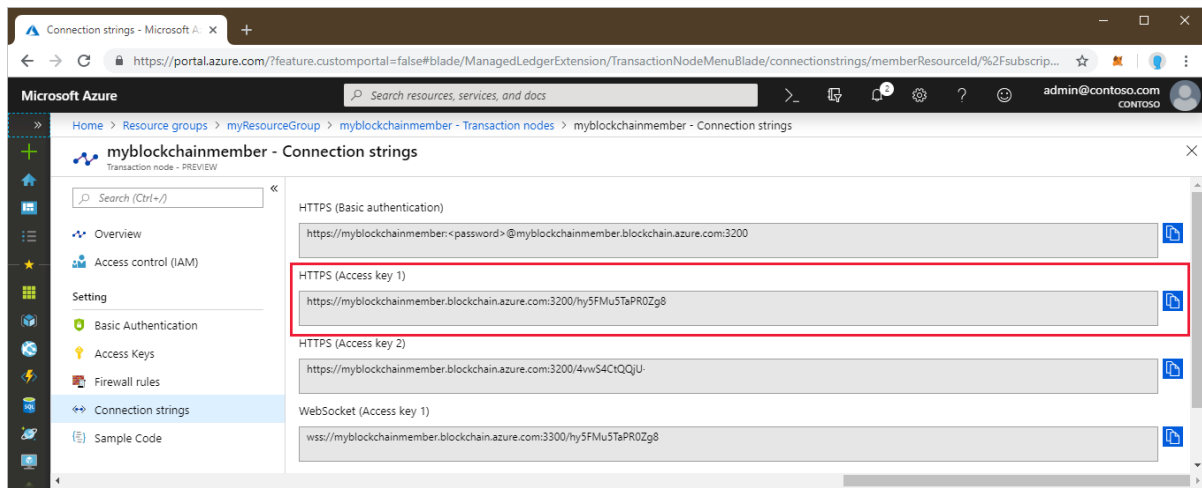
## Get the Geth connection string

You can find the Geth connection string in the Azure portal.

1. Sign in to the [Azure portal](#).
2. Navigate to your Azure Blockchain Service member. Select **Transaction nodes** and the default transaction node link.



3. Select **Connection strings**.
4. Copy the connection string from **HTTPS (Access key 1)**. You need the command for the next section.



## Connect to Geth

1. Open a command prompt or shell.
2. Use the Geth attach subcommand to attach to the running Geth instance on your transaction node. Paste the connection string as an argument for the attach subcommand. For example,

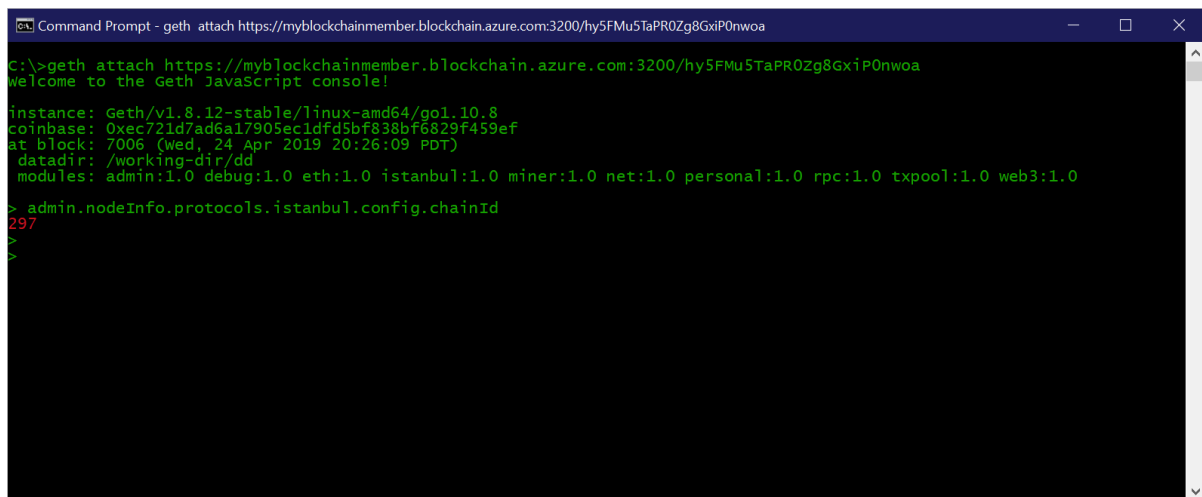
```
geth attach <connection string>
```

3. Once connected to the transaction node's Ethereum console, you can call the web3 JavaScript Dapp API or the admin API.

For example, Use the following API to find out the chainId.

```
admin.nodeInfo.protocols.istanbul.config.chainId
```

In this example, the chainId is 297.



4. To disconnect from the console, type `exit`.

## Next steps

In this quickstart, you used the Geth client to attach to a Geth instance on an Azure Blockchain Service transaction node. Try the next tutorial to deploy and send a transaction using Truffle.

[Send a transaction](#)

# Quickstart: Use Truffle to connect to an Azure Blockchain Service network

5/30/2019 • 2 minutes to read • [Edit Online](#)

Truffle is a blockchain development environment you can use to connect to an Azure Blockchain Service node.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- [Create an Azure Blockchain member](#)
- Install [Truffle](#). Truffle requires several tools to be installed including [Node.js](#), [Git](#).
- Install [Python 2.7.15](#). Python is needed for Web3.

## Create Truffle project

1. Open a Node.js command prompt or shell.
2. Change directory to where you want to create the Truffle project directory.
3. Create a directory for the project and change your path to the new directory. For example,

```
mkdir truffledemo
cd truffledemo
```

4. Initialize the Truffle project.

```
truffle init
```

5. Install Ethereum JavaScript API web3 in the project folder. Currently, version web3 version 1.0.0-beta.37 is required.

```
npm install web3@1.0.0-beta.37
```

You may receive npm warnings during installation.

6. Launch Truffle's interactive development console.

```
truffle develop
```

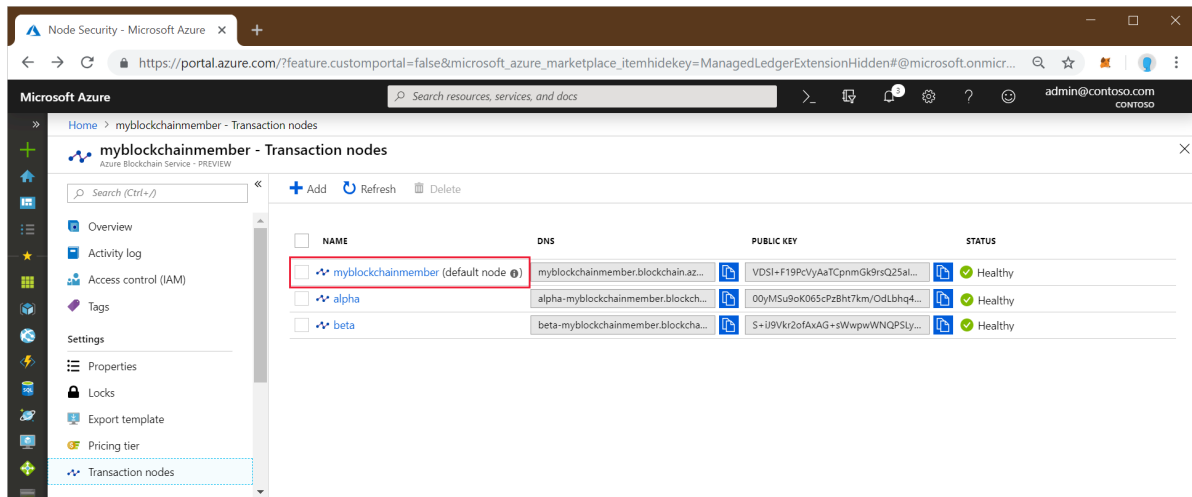
Truffle creates a local development blockchain and provides an interactive console.

## Connect to transaction node

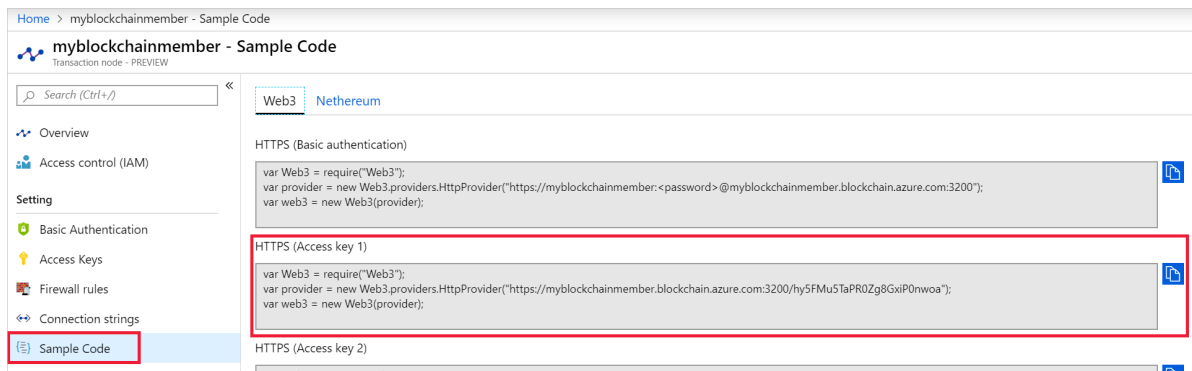
Use *Web3* to connect to the transaction node. You can get the *Web3* connection string from the Azure portal.

1. Sign in to the [Azure portal](#).
2. Navigate to your Azure Blockchain Service member. Select **Transaction nodes** and the default transaction

node link.



3. Select **Sample Code > Web3**.
4. Copy the JavaScript from **HTTPS (Access key 1)**. You need the code for Truffle's interactive development console.



5. Paste the JavaScript code from the previous step into the Truffle interactive development console. The code creates a web3 object that is connected to your Azure Blockchain Service transaction node.

Example output:

```
truffle(develop)> var Web3 = require("Web3");
truffle(develop)> var provider = new
Web3.providers.HttpProvider("https://myblockchainmember.blockchain.azure.com:3200/");
truffle(develop)> var web3 = new Web3(provider);
```

You can call methods on the **web3** object to interact with your transaction node.

6. Call the **getBlockNumber** method to return the current block number.

```
web3.eth.getBlockNumber();
```

Example output:

```
truffle(develop)> web3.eth.getBlockNumber();
18567
```

7. Exit the Truffle development console.

```
.exit
```

## Next steps

In this quickstart, you created a Truffle project to connect to your Azure Blockchain Service default transaction node.

Try the next tutorial to use Truffle to send a transaction to your consortium blockchain network.

[Send a transaction](#)



# Tutorial: Send transactions using Azure Blockchain Service

5/30/2019 • 8 minutes to read • [Edit Online](#)

In this tutorial, you'll create transaction nodes to test contract and transaction privacy. You'll use Truffle to create a local development environment and deploy a smart contract and send a private transaction.

You'll learn how to:

- Add transaction nodes
- Use Truffle to deploy a smart contract
- Send a transaction
- Validate transaction privacy

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

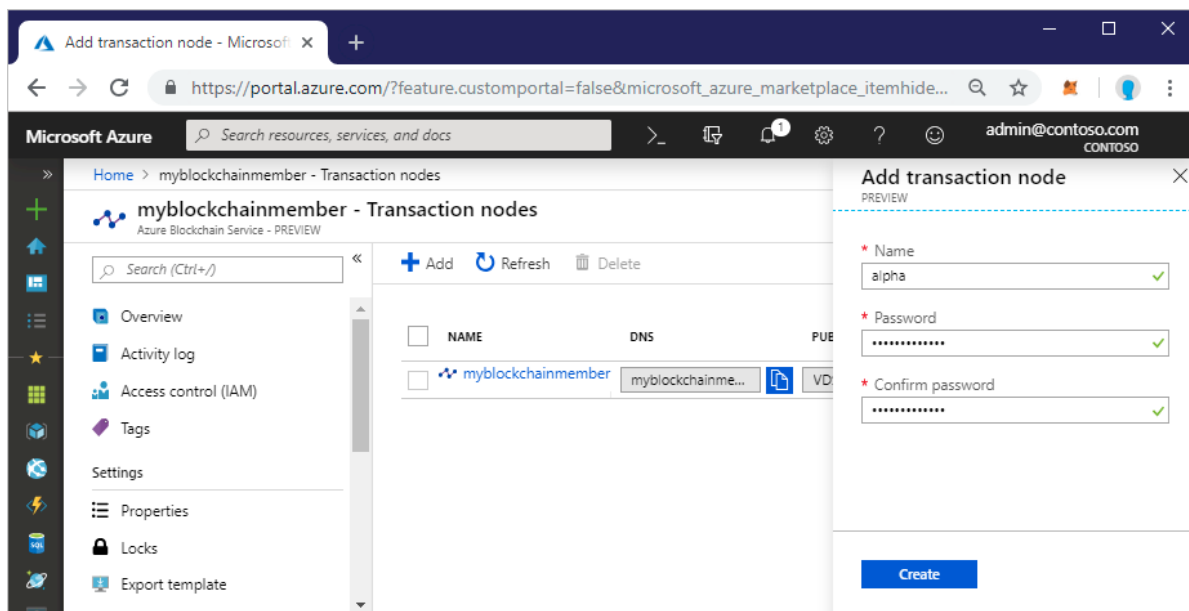
## Prerequisites

- Complete [Create a blockchain member using the Azure portal](#)
- Complete [Quickstart: Use Truffle to connect to a consortium network](#)
- Install [Truffle](#). Truffle requires several tools to be installed including [Node.js](#), [Git](#).
- Install [Python 2.7.15](#). Python is needed for Web3.
- Install [Visual Studio Code](#)
- Install [Visual Studio Code Solidity extension](#)

## Create transaction nodes

By default, you have one transaction node. We're going to add two more. One of the nodes participates in the private transaction. The other is not included in the private transaction.

1. Sign in to the [Azure portal](#).
2. Navigate to your Azure Blockchain member and select **Transaction nodes > Add**.
3. Complete the settings for a new transaction node named `alpha`.



SETTING	VALUE	DESCRIPTION
Name	alpha	Transaction node name. The name is used to create the DNS address for the transaction node endpoint. For example, alpha-mymanagedledger.blockchain.azure.com
Password	Strong password	The password is used to access the transaction node endpoint with basic authentication.

#### 4. Select **Create**.

Provisioning a new transaction node takes about 10 minutes.

#### 5. Repeat steps 2 through 4 to add a transaction node named `beta`.

You can continue with the tutorial while the nodes are being provisioned. When provisioning is finished, you'll have three transaction nodes.

## Open Truffle console

1. Open a Node.js command prompt or shell.
2. Change your path to the Truffle project directory from the prerequisite [Quickstart: Use Truffle to connect to a consortium network](#). For example,

```
cd truffledemo
```

#### 3. Launch Truffle's interactive development console.

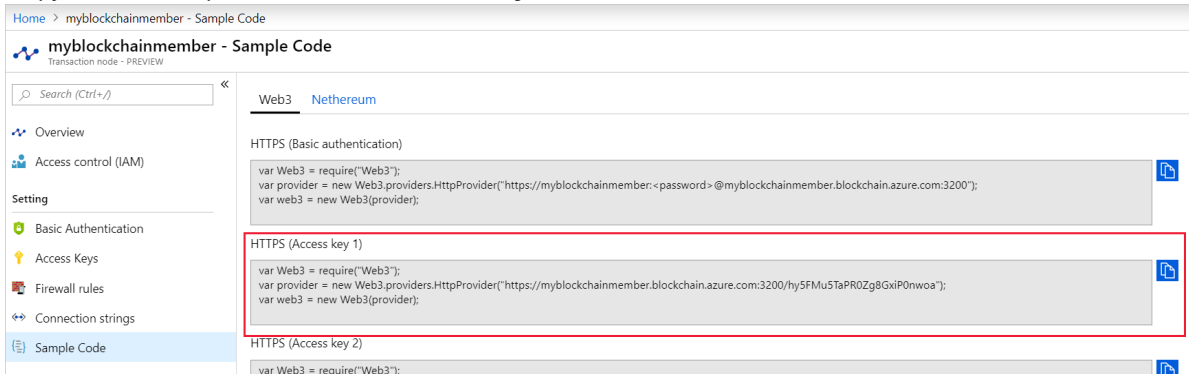
```
truffle develop
```

Truffle creates a local development blockchain and provides an interactive console.

# Create Ethereum account

Use Web3 to connect to the default transaction node and create an Ethereum account. You can get the Web3 connection string from the Azure portal.

1. In the Azure portal, navigate to the default transaction node and select **Transaction nodes > Sample code > Web3**.
2. Copy the JavaScript from **HTTPS (Access key 1)**



3. Paste the Web3 JavaScript code for the default transaction node into the Truffle interactive development console. The code creates a Web3 object that is connected to your Azure Blockchain Service transaction node.

```
truffle(develop)> var Web3 = require("Web3");
truffle(develop)> var provider = new
Web3.providers.HttpProvider("https://myblockchainmember.blockchain.azure.com:3200/hy5FMu5TaPR0Zg8GxiPwn
ed");
truffle(develop)> var web3 = new Web3(provider);
```

You can call methods on the Web3 object to interact with your transaction node.

4. Create a new account on the default transaction node. Replace the password parameter with your own strong password.

```
web3.eth.personal.newAccount("1@myStrongPassword");
```

Make note of the account address returned and password. You need the Ethereum account address and password in the next section.

5. Exit the Truffle development environment.

```
.exit
```

## Configure Truffle project

To configure the Truffle project, you need some transaction node information from the Azure portal.

### Transaction node public key

Each transaction node has a public key. The public key enables you to send a private transaction to the node. In order to send a transaction from the default transaction node to the *alpha* transaction node, you need the *alpha* transaction node's public key.

You can get the public key from the transaction node list. Copy the public key for the alpha node and save the value for later in the tutorial.

Home > myblockchainmember - Transaction nodes > myblockchainmember - Sample Code

### myblockchainmember - Transaction nodes

Azure Blockchain Service - PREVIEW

Search (Ctrl+/)

+ Add Refresh Delete

	NAME	DNS	PUBLIC KEY	STATUS
<input checked="" type="checkbox"/>	myblockchainmember (default node)	myblockchainmember.blockchain.azure.com	VDSI+F19PcVyAaTCpnmGk9rsQ25al...	Healthy
<input type="checkbox"/>	alpha	alpha-myblockchainmember.blockchain.azure.com	00yMSu9oK065cPzBht7km/OdLbhq...	Healthy
<input type="checkbox"/>	beta	beta-myblockchainmember.blockchain.azure.com	S+IJ9Vkr2ofAxAG+sWwpwWNQPSL...	Healthy

## Transaction node endpoint addresses

1. In the Azure portal, navigate to each transaction node and select **Transaction nodes > Connection strings**.
2. Copy and save the endpoint URL from **HTTPS (Access key 1)** for each transaction node. You need the endpoint addresses for the smart contract configuration file later in the tutorial.

Home > myblockchainmember - Transaction nodes > myblockchainmember - Connection strings

### myblockchainmember - Connection strings

Transaction node - PREVIEW

Search (Ctrl+/)

Overview

Access control (IAM)

Setting

Basic Authentication

Access Keys

Firewall rules

Connection strings

HTTPS (Basic authentication)

https://myblockchainmember-<password>@myblockchainmember.blockchain.azure.com:3200

HTTPS (Access key 1)

https://myblockchainmember.blockchain.azure.com:3200/ny5FMuSTaPR0Zg8GxiP0nwoa

Copy to clipboard

HTTPS (Access key 2)

https://myblockchainmember.blockchain.azure.com:3200/4vwS4CtQQU-R2GR63kpFCT

WebSocket (Access key 1)

## Edit configuration file

1. Launch Visual Studio Code and open the Truffle project directory folder using the **File > Open Folder** menu.
2. Open the Truffle configuration file `truffle-config.js`.
3. Replace the contents of the file with the following configuration information. Add variables containing the endpoints addresses and account information. Replace the angle bracket sections with values you collected from previous sections.

```

var defaultnode = "<default transaction node connection string>";
var alpha = "<alpha transaction node connection string>";
var beta = "<beta transaction node connection string>";

var myAccount = "<Ethereum account address>";
var myPassword = "<Ethereum account password>";

var Web3 = require("web3");

module.exports = {
  networks: {
    defaultnode: {
      provider: (() => {
        const AzureBlockchainProvider = new Web3.providers.HttpProvider(defaultnode);

        const web3 = new Web3(AzureBlockchainProvider);
        web3.eth.personal.unlockAccount(myAccount, myPassword);

        return AzureBlockchainProvider;
      })(),

      network_id: "*",
      gas: 0,
      gasPrice: 0,
      from: myAccount
    },
    alpha: {
      provider: new Web3.providers.HttpProvider(alpha),
      network_id: "*",
      gas: 0,
      gasPrice: 0
    },
    beta: {
      provider: new Web3.providers.HttpProvider(beta),
      network_id: "*",
      gas: 0,
      gasPrice: 0
    }
  }
}

```

4. Save the changes to `truffle-config.js`.

## Create smart contract

1. In the **contracts** folder, create a new file named `SimpleStorage.sol`. Add the following code.

```
pragma solidity >=0.4.21 <0.6.0;

contract SimpleStorage {
    string public storedData;

    constructor(string memory initVal) public {
        storedData = initVal;
    }

    function set(string memory x) public {
        storedData = x;
    }

    function get() view public returns (string memory retVal) {
        return storedData;
    }
}
```

2. In the **migrations** folder, create a new file named `2_deploy_simplestorage.js`. Add the following code.

```
var SimpleStorage = artifacts.require("SimpleStorage.sol");

module.exports = function(deployer) {

    // Pass 42 to the contract as the first constructor parameter
    deployer.deploy(SimpleStorage, "42", {privateFor: ["<alpha node public key>"], from:"<Ethereum account address>"})
};
```

3. Replace the values in the angle brackets.

VALUE	DESCRIPTION
<alpha node public key>	Public key of the alpha node
<Ethereum account address>	Ethereum account address created in the default transaction node

In this example, the initial value of the **storeData** value is set to 42.

**privateFor** defines the nodes to which the contract is available. In this example, the default transaction node's account can cast private transactions to the **alpha** node. You add public keys for all private transaction participants. If you don't include **privateFor:** and **from;**, the smart contract transactions are public and can be seen by all consortium members.

4. Save all files by selecting **File > Save All**.

## Deploy smart contract

Use Truffle to deploy `SimpleStorage.sol` to default transaction node network.

```
truffle migrate --network defaultnode
```

Truffle first compiles and then deploys the **SimpleStorage** smart contract.

Example output:

```

admin@desktop:/mnt/c/truffledemo$ truffle migrate --network defaultnode

2_deploy_simplestorage.js
=====

Deploying 'SimpleStorage'
-----
> transaction hash:    0x3f695ff225e7d11a0239ffcaaab0d5f72adb545912693a77fbfc11c0dbe7ba72
> Blocks: 2           Seconds: 12
> contract address:   0x0b15c15C739c1F3C1e041ef70E0011e641C9D763
> account:            0x1a0B9683B449A8FcAd294A01E881c90c734735C3
> balance:            0
> gas used:           0
> gas price:          0 gwei
> value sent:         0 ETH
> total cost:         0 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost:                0 ETH

Summary
=====
> Total deployments:   2
> Final cost:          0 ETH

```

## Validate contract privacy

Because of contract privacy, contract values can only be queried from nodes we declared in **privateFor**. In this example, we can query the default transaction node because the account exists in that node.

1. Using the Truffle console, connect to the default transaction node.

```
truffle console --network defaultnode
```

2. In the Truffle console, execute code that returns the value of the contract instance.

```
SimpleStorage.deployed().then(function(instance){return instance.get();})
```

If querying the default transaction node is successful, the value 42 is returned. For example:

```

admin@desktop:/mnt/c/truffledemo$ truffle console --network defaultnode
truffle(defaultnode)> SimpleStorage.deployed().then(function(instance){return instance.get();})
'42'

```

3. Exit the Truffle console.

```
.exit
```

Since we declared **alpha** node's public key in **privateFor**, we can query the **alpha** node.

1. Using the Truffle console, connect to the **alpha** node.

```
truffle console --network alpha
```

2. In the Truffle console, execute code that returns the value of the contract instance.

```
SimpleStorage.deployed().then(function(instance){return instance.get();})
```

If querying the **alpha** node is successful, the value 42 is returned. For example:

```
admin@desktop:/mnt/c/truffledemo$ truffle console --network alpha
truffle(alpha)> SimpleStorage.deployed().then(function(instance){return instance.get();})
'42'
```

3. Exit the Truffle console.

```
.exit
```

Since we did not declare **beta** node's public key in **privateFor**, we won't be able to query the **beta** node because of contract privacy.

1. Using the Truffle console, connect to the **beta** node.

```
truffle console --network beta
```

2. Execute a code that returns the value of the contract instance.

```
SimpleStorage.deployed().then(function(instance){return instance.get();})
```

3. Querying the **beta** node fails since the contract is private. For example:

```
admin@desktop:/mnt/c/truffledemo$ truffle console --network beta
truffle(beta)> SimpleStorage.deployed().then(function(instance){return instance.get();})
Thrown:
Error: Returned values aren't valid, did it run Out of Gas?
    at XMLHttpRequest._onHttpResponseEnd (/mnt/c/truffledemo/node_modules/xhr2-cookies/xml-http-
request.ts:345:8)
    at XMLHttpRequest._setReadyState (/mnt/c/truffledemo/node_modules/xhr2-cookies/xml-http-
request.ts:219:8)
    at XMLHttpRequestEventTarget.dispatchEvent (/mnt/c/truffledemo/node_modules/xhr2-cookies/xml-http-
request-event-target.ts:44:13)
    at XMLHttpRequest.request.onreadystatechange (/mnt/c/truffledemo/node_modules/web3-providers-
http/src/index.js:96:13)
```

4. Exit the Truffle console.

```
.exit
```

## Send a transaction

1. Create a file called `sampletx.js`. Save it in the root of your project.
2. The following script sets the contract **storedData** variable value to 65. Add the code to the new file.



```

var SimpleStorage = artifacts.require("SimpleStorage");

module.exports = function(done) {
  console.log("Getting deployed version of SimpleStorage...")
  SimpleStorage.deployed().then(function(instance) {
    console.log("Setting value to 65...");
    return instance.set("65", {privateFor: ["<alpha node public key>"], from:"<Ethereum account address>"});
  }).then(function(result) {
    console.log("Transaction:", result.tx);
    console.log("Finished!");
    done();
  }).catch(function(e) {
    console.log(e);
    done();
  });
};

```

Replace the values in the angle brackets then save the file.

VALUE	DESCRIPTION
<alpha node public key>	Public key of the alpha node
<Ethereum account address>	Ethereum account address created in the default transaction node.

**privateFor** defines the nodes to which the transaction is available. In this example, the default transaction node's account can cast private transactions to the **alpha** node. You need to add public keys for all private transaction participants.

3. Use Truffle to execute the script for the default transaction node.

```
truffle exec sampletx.js --network defaultnode
```

4. In the Truffle console, execute code that returns the value of the contract instance.

```
SimpleStorage.deployed().then(function(instance){return instance.get();})
```

If the transaction was successful, the value 65 is returned. For example:

```

Getting deployed version of SimpleStorage...
Setting value to 65...
Transaction: 0x864e67744c2502ce75ef6e5e09d1bfeb5cdfb7b880428fceca84bc8fd44e6ce0
Finished!

```

5. Exit the Truffle console.

```
.exit
```

## Validate transaction privacy

Because of transaction privacy, transactions can only be performed on nodes we declared in **privateFor**. In this example, we can perform transactions since we declared **alpha** node's public key in **privateFor**.

1. Use Truffle to execute the transaction on the **alpha** node.

```
truffle exec sampletx.js --network alpha
```

2. Execute code that returns the value of the contract instance.

```
SimpleStorage.deployed().then(function(instance){return instance.get();})
```

If the transaction was successful, the value 65 is returned. For example:

```
Getting deployed version of SimpleStorage...
Setting value to 65...
Transaction: 0x864e67744c2502ce75ef6e5e09d1bfeb5cdfb7b880428fceca84bc8fd44e6ce0
Finished!
```

3. Exit the Truffle console.

```
.exit
```

## Clean up resources

When no longer needed, you can delete the resources by deleting the `myResourceGroup` resource group you created by the Azure Blockchain Service.

To delete the resource group:

1. In the Azure portal, navigate to **Resource group** in the left navigation pane and select the resource group you want to delete.
2. Select **Delete resource group**. Verify deletion by entering the resource group name and select **Delete**.

## Next steps

In this tutorial, you added two transaction nodes to demonstrate contract and transaction privacy. You used the default node to deploy a private smart contract. You tested privacy by querying contract values and performing transactions on the blockchain.

[Developing blockchain applications using Azure Blockchain Service](#)

# Azure Blockchain Service Consortium

5/2/2019 • 3 minutes to read • [Edit Online](#)

Using Azure Blockchain Service, you can create private consortium blockchain networks where each blockchain network can be limited to specific participants in the network. Only participants in the private consortium blockchain network can view and interact with the blockchain. Consortium networks in Azure Blockchain Service can contain two types of member participant roles:

- **Administrator** - Privileged participants who can take consortium management actions and can participate in blockchain transactions.
- **User** - Participants who cannot take any consortium management action but can participate in blockchain transactions.

Consortium networks can be a mix of participant roles and can have an arbitrary number of each role type. There must be at least one administrator.

The following diagram shows a consortium network with multiple participants:

## Private Blockchain Consortium



With consortium management in Azure Blockchain Service, you can manage participants in the consortium network. Management of the consortium is based on the consensus model of the network. In the current preview release, Azure Blockchain Service provides a centralized consensus model for consortium management. Any privileged participant with an administer role can take consortium management actions, such as adding or

removing participants from a network.

## Roles

Participants in a consortium can be individuals or organizations and can be assigned a user role or an administrator role. The following table lists the high-level differences between the two roles:

ACTION	USER ROLE	ADMINISTRATOR ROLE
Create new member	Yes	Yes
Invite new members	No	Yes
Set or change member participant role	No	Yes
Change member display name	Only for own member	Only for own member
Remove members	Only for own member	Yes
Participate in blockchain transactions	Yes	Yes

### User role

Users are consortium participants with no administrator capabilities. They cannot participate in managing members related to the consortium. Users can change their member display name and can remove themselves from a consortium.

### Administrator

An administrator can manage members within the consortium. An administrator can invite members, remove members, or update members roles within the consortium. There must always be at least one administrator within a consortium. The last administrator must specify another participant as an administrator role before leaving a consortium.

## Managing members

Only administrators can invite other participants to the consortium. Administrators invite participants using their Azure subscription ID.

Once invited, participants can join the blockchain consortium by deploying a new member in Azure Blockchain Service. To view and join the invited consortium, you must specify the same Azure subscription ID used in the invite by the network administrator.

Administrators can remove any participant from the consortium, including other administrators. Members can only remove themselves from a consortium.

## Consortium management smart contract

Consortium management in Azure Blockchain Service is done via consortium management smart contracts. The smart contracts are automatically deployed to your nodes when you deploy a new blockchain member.

The address of the root consortium management smart contract can be viewed in the Azure portal. The **RootContract address** is in blockchain member's overview section.

Home > myblockchainmember

**myblockchainmember**  
Azure Blockchain Service

Search (Ctrl+/) « Delete

Overview

Activity log

Access control (IAM)

Tags

Settings

Properties

Locks

Export template

Resource group  
[myResourceGroup](#)

Status  
Available

Location  
East US

Subscription  
[Visual Studio Enterprise](#)

Subscription ID  
<Subscription ID>

Member name  
myblockchainmember

Protocol  
Quorum

Pricing Tier  
[Standard \(2 vCores, 3 nodes\)](#)

Consortium  
myCompany

RootContract address  
**0xb255f55e8d600f09ebc1035dd2118acec101beef**

Member account  
[0x85b911c9e103d6405573151258d668479e9beefc](#)

You can interact with the consortium management smart contract using the consortium management [PowerShell module](#), Azure portal, or directly through the smart contract using the Azure Blockchain Service generated Ethereum account.

## Ethereum account

When a member is created, an Ethereum account key is created. Azure Blockchain Service uses the key to create transactions related to consortium management. The Ethereum account key is managed by Azure Blockchain Service automatically.

The member account can be viewed in the Azure portal. The member account is in blockchain member's overview section.

Home > myblockchainmember

**myblockchainmember**  
Azure Blockchain Service

Search (Ctrl+/) « Delete

Overview

Activity log

Access control (IAM)

Tags

Settings

Properties

Locks

Export template

Resource group  
[myResourceGroup](#)

Status  
Available

Location  
East US

Subscription  
[Visual Studio Enterprise](#)

Subscription ID  
<Subscription ID>

Member name  
myblockchainmember

Protocol  
Quorum

Pricing Tier  
[Standard \(2 vCores, 3 nodes\)](#)

Consortium  
myCompany

RootContract address  
0xb255f55e8d600f09ebc1035dd2118acec101beef

Member account  
**0x85b911c9e103d6405573151258d668479e9beefc**

You can reset your Ethereum account by clicking on your member account and entering a new password. Both the Ethereum account address and the password will be reset.

## Next steps

[How to manage members in Azure Blockchain Service using PowerShell](#)

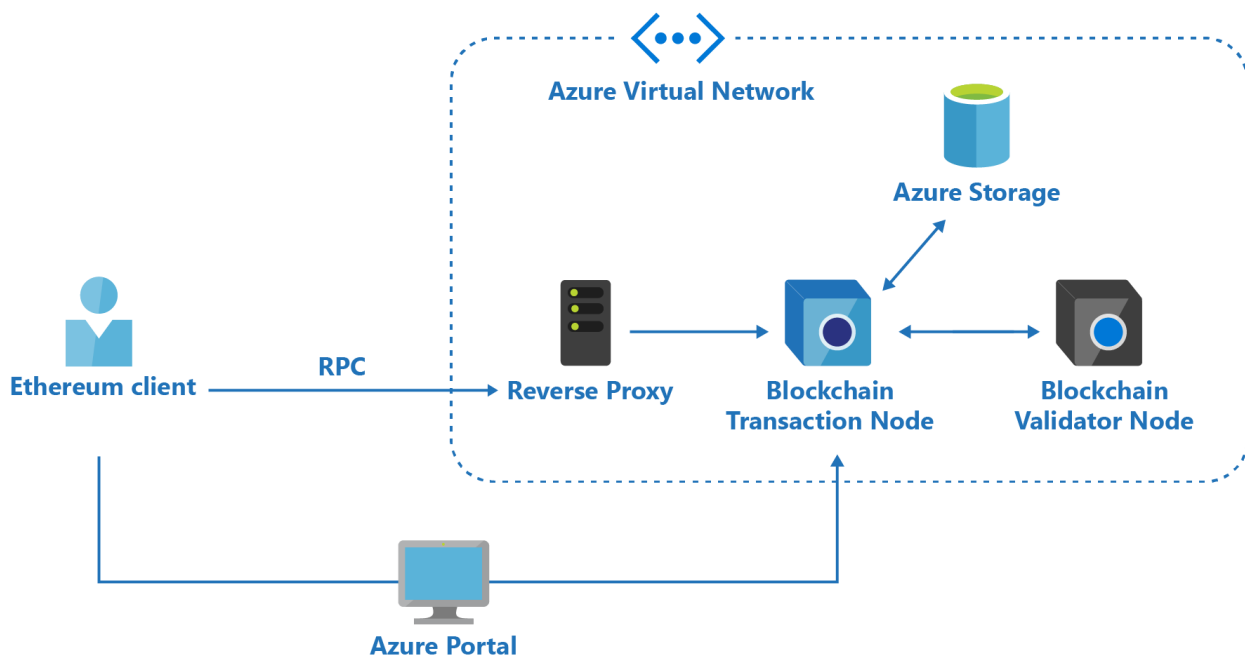
# Azure Blockchain Service security

5/2/2019 • 2 minutes to read • [Edit Online](#)

Azure Blockchain Service uses several Azure capabilities to keep your data secure and available. Data is secured using isolation, encryption, and authentication.

## Isolation

Azure Blockchain Service resources are isolated in a private virtual network. Each transaction and validation node is a virtual machine (VM). VMs in one virtual network cannot communicate directly to VMs in a different virtual network. Isolation ensures communication remains private within the virtual network. For more information on Azure virtual network isolation, see [isolation in the Azure Public Cloud](#).

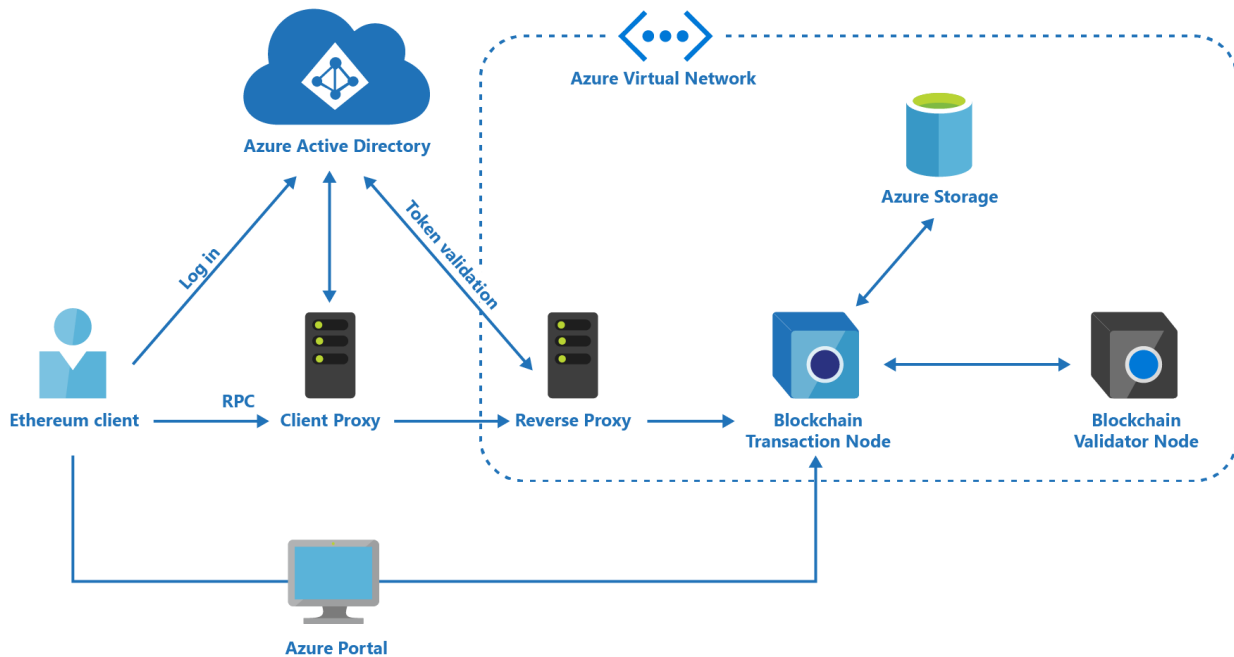


## Encryption

User data is stored in Azure storage. User data is encrypted in motion and at rest for security and confidentiality. For more information, see: [Azure Storage security guide](#).

## Authentication

Transactions can be sent to blockchain nodes via an RPC endpoint. Clients communicate with a transaction node using a reverse proxy server that handles user authentication and encrypts data over SSL.



There are three modes of authentication for RPC access.

### Basic authentication

Basic authentication uses an HTTP authentication header containing the user name and password. User name is the name of the blockchain node. Password is set during provisioning of a member or node. The password can be changed using the Azure portal or CLI.

### Access keys

Access keys use a randomly generated string included in the endpoint URL. Two access keys help enable key rotation. Keys can be regenerated from the Azure portal and CLI.

### Azure Active Directory

Azure Active Directory (Azure AD) uses a claim-based authentication mechanism where the user is authenticated by Azure AD using Azure AD user credentials. Azure AD provides cloud-based identity management and allows customers to use a single identity across an entire enterprise and access applications on the cloud. Azure Blockchain Service integrates with Azure AD enabling ID federation, single sign-on and multi-factor authentication. You can assign users, groups, and application roles in your organization for blockchain member and node access.

The Azure AD client proxy is available on [GitHub](#). The client proxy directs the user to the Azure AD sign-in page and obtains a bearer token upon successful authentication. Subsequently, the user connects an Ethereum client application such as Geth or Truffle to the client proxy's endpoint. Finally, when a transaction is submitted, the client proxy injects the bearer token in the http header and the reverse proxy validates the token using OAuth protocol.

## Keys and Ethereum accounts

When provisioning an Azure Blockchain Service member, an Ethereum account and a public and private key pair is generated. The private key is used to send transactions to the blockchain. The Ethereum account is the last 20 bytes of the public key's hash. The Ethereum account is also called a wallet.

The private and public key pair is stored as a keyfile in JSON format. The private key is encrypted using the password entered when the blockchain ledger service is created.

Private keys are used to digitally sign transactions. In private blockchains, a smart contract signed by a private key represents the signer's identity. To verify the validity of the signature, the receiver can compare the public key of the signer with the address computed from the signature.

Constellation keys are used to uniquely identify a Quorum node. Constellation keys are generated at the time of node provisioning and are specified in the `privateFor` parameter of a private transaction in Quorum.

## Next steps

[Configure Azure Blockchain Service transaction nodes](#)



# Azure Blockchain Service development overview

5/2/2019 • 3 minutes to read • [Edit Online](#)

With Azure Blockchain Service, you can create consortium blockchain networks to enable enterprise scenarios like asset tracking, digital token, loyalty and reward, supply chain financial, and provenance. This article is an introduction to Azure Blockchain Service development overview and key topics to implement blockchain for enterprise.

## Client connection to Azure Blockchain Service

There are different types of clients for blockchain networks including full nodes, light nodes, and remote clients. Azure Blockchain Service builds a blockchain network that includes nodes. You can use different clients as your gateway to Azure Blockchain Service for blockchain development. Azure Blockchain Service offers basic authentication or access key as a development endpoint. The following are popular clients you can use connect.

### MetaMask

MetaMask is a browser-based wallet (remote client), RPC client, and basic contract explorer. Unlike other browser wallets, MetaMask injects a web3 instance into the browser JavaScript context, acting as an RPC client that connects to a variety of Ethereum blockchains (*mainnet*, *Ropsten testnet*, *Kovan testnet*, local RPC node, etc.). You can set up custom RPC easily to connect to Azure Blockchain Service and start blockchain development using Remix.

### Geth

Geth is the command-line interface for running a full Ethereum node implemented in Go. You don't need to run full node but can launch its interactive console that provides a JavaScript runtime environment exposing a JavaScript API to interact with Azure Blockchain Service.

## Development framework configuration

To develop sophisticated enterprise blockchain solutions, a development framework is needed to connect to different blockchain networks, manage smart contract lifecycle, automate testing, deploy smart contract with scripts, and equip an interactive console.

Truffle is a popular blockchain development framework to write, compile, deploy, and test decentralized applications on Ethereum blockchains. You can also think of Truffle as a framework that attempts to seamlessly integrate smart contract development and traditional web development.

Even the smallest project interacts with at least two blockchain nodes: One on the developer's machine, and the other representing the network where the developer deploys their application. For example, the main public Ethereum network or Azure Blockchain Service. Truffle provides a system for managing the compilation and deployment artifacts for each network and does so in a way that simplifies final application deployment. For more information, see [Quickstart: Use Truffle to connect to an Azure Blockchain Service network](#).

## Ethereum Quorum private transaction

Quorum is an Ethereum-based distributed ledger protocol with transaction plus contract privacy and new consensus mechanisms. Key enhancements over Go-Ethereum include:

- Privacy - Quorum supports private transactions and private contracts through public and private state separation and utilizes peer-to-peer encrypted message exchanges for directed transfer of private data to network participants.

- Alternative Consensus Mechanisms - with no need for proof-of-work or proof-of-stake consensus in a permissioned network. Quorum offers multiple consensus mechanisms that are designed for consortium chains such as RAFT and IBFT. Azure Blockchain Services uses the IBFT consensus mechanism.
- Peer Permissioning - node and peer permissioning using smart contracts, ensuring only known parties can join the network
- Higher Performance - Quorum offers higher performance than public Geth

See [Tutorial: Send a transaction using Azure Blockchain Service](#) for an example of private transaction.

## Block explorers

Block explorers are online blockchain browsers that display individual block content, transaction address data, and history. Basic block information is available through Azure Monitor in Azure Blockchain Service, however, if you need more detail information during development, block explorers can be useful. There are popular open-source block explorers you can use. The following is a list of block explorers that work with Azure Blockchain Service:

- [Azure Blockchain Service Explorer](#) from Web3 Labs
- [BlockScout](#)

## TPS measurement

As blockchain is used in more enterprise scenarios, transactions per second (TPS) speed is important to avoid bottlenecks and system inefficiencies. High transaction rates can be difficult to maintain within a decentralized blockchain. An accurate TPS measurement may be affected by different factors such as server thread, transaction queue size, network latency, and security. If you need to measure TPS speed during development, a popular open-source tool is [ChainHammer](#).

## Next steps

[Quickstart: Use Truffle to connect to a an Azure Blockchain Service network](#)

# Limits in Azure Blockchain Service

5/2/2019 • 2 minutes to read • [Edit Online](#)

Azure Blockchain Service has service and functional limits such as the number of nodes a member can have, consortium restrictions, and storage amounts.

## Pricing tier

Maximum limits on transactions and validator nodes depend on whether you provision Azure Blockchain Service at Basic or Standard pricing tiers.

PRICING TIER	MAX TRANSACTION NODES	MAX VALIDATOR NODES
Basic	10	1
Standard	10	2

Changing the pricing tier between Basic and Standard after member creation is not supported.

## Storage capacity

The maximum amount of storage that can be used per node for ledger data and logs is 1 terabyte.

Decreasing ledger and log storage size is not supported.

## Consortium limits

- **Consortium and member names must be unique** from other consortium and member names in the Azure Blockchain Service.
- **Member and consortium names cannot be changed**
- **All members in a consortium must be in the same pricing tier**
- **All members that participate in a consortium must reside in the same region**

The first member created in a consortium dictates the region. Invited members to the consortium must reside in the same region as the first member. Limiting all members to the same region helps ensure network consensus is not negatively impacted.

- **A consortium must have at least one administrator**

If there is only one administrator in a consortium, they cannot remove themselves from the consortium or delete their member until another administrator is added or promoted in the consortium.

- **Members removed from the consortium cannot be added again**

Rather, they must be reinvited to join the consortium and create a new member. Their existing member resource are not deleted to preserve historical transactions.

- **All members in a consortium must be using the same ledger version**

For more information on the patching, updates, and ledger versions available in Azure Blockchain Service, see [Patching, updates, and versions](#).

## Next steps

- [Patching, updates, and versions](#)

# Supported Azure Blockchain Service ledger versions

5/30/2019 • 2 minutes to read • [Edit Online](#)

Azure Blockchain Service uses the Ethereum-based [Quorum](#) ledger designed for the processing of private transactions within a group of known participants, identified as a consortium in Azure Blockchain Service.

Currently, Azure Blockchain Service supports [Quorum version 2.2.1](#) and [Tessera transaction manager](#).

## Managing updates and upgrades

Versioning in Quorum is done through a major, minor and patch releases. For example, if the Quorum version is 2.0.1, release type would be categorized as follows:

MAJOR	MINOR	PATCH
2	0	1

Azure Blockchain Service automatically updates patch releases of Quorum to existing running members within 30 days of being made available from Quorum.

## Availability of new ledger versions

Azure Blockchain Service provides the latest major and minor versions of the Quorum ledger within 60 days of being available from the Quorum manufacturer. A maximum of four minor releases are provided for consortia to choose from when provisioning a new member and consortium. Upgrading from to a major or minor release is currently not supported.

## Next steps

[Limits in Azure Blockchain Service](#)

# How to configure Azure Active Directory access

5/2/2019 • 2 minutes to read • [Edit Online](#)

In this article, you learn how to grant access and connect to Azure Blockchain Service nodes using Azure Active Directory (Azure AD) user, group, or application IDs.

Azure AD provides cloud-based identity management and allows you to use a single identity across an entire enterprise and access applications in Azure. Azure Blockchain Service is integrated with Azure AD and offers benefits such as ID federation, single sign-on and multi-factor authentication.

## Prerequisites

- [Create a blockchain member using the Azure portal](#)

## Grant access

You can grant access at both the member level and the node level. Granting access rights at the member level will in turn grant access to all nodes under the member.

### Grant member level access

To grant access permission at the member level.

1. Sign in to the [Azure portal](#).
2. Navigate to **Access control (IAM) > Add > Add role assignment**.
3. Select the **Blockchain Member Node Access (Preview)** role and add the Azure AD ID object you wish to grant access to. Azure AD ID object can be:

AZURE AD OBJECT	EXAMPLE
Azure AD user	<code>frank@contoso.onmicrosoft.com</code>
Azure AD group	<code>sales@contoso.onmicrosoft.com</code>
Application ID	<code>13925ab1-4161-4534-8d18-812f5ca1ab1e</code>

4. Select **Save**.

### Grant node level access

1. You can grant access at the node level by navigating to node security and click on the node name that you wish to grant access.
2. Select the Blockchain Member Node Access (Preview) role and add the Azure AD ID object you wish to grant

access to.

## Connect using Azure Blockchain Connector

Download or clone the [Azure Blockchain Connector from GitHub](#).

```
git clone https://github.com/Microsoft/azure-blockchain-connector.git
```

Then follow the quickstart section in the **readme** to build the connector from the source code.

### Connect using an Azure AD user account

1. Run the following command to authenticate using an Azure AD user account. Replace <myAADDirectory> with an Azure AD domain. For example, `yourdomain.onmicrosoft.com`.

```
connector.exe -remote <myMemberName>.blockchain.azure.com:3200 -method aadauthcode -tenant-id <myAADDirectory>
```

2. Azure AD prompts for credentials.
3. Sign in with your user name and password.
4. Upon successful authentication, your local proxy connects to your blockchain node. You can now attach your Geth client with the local endpoint.

```
geth attach http://127.0.0.1:3100
```

### Connect using an application ID

Many applications authenticate with Azure AD using an application ID instead of an Azure AD user account.

To connect to your node using an application ID, replace **aadauthcode** with **aadclient**.

```
connector.exe -remote <myBlockchainEndpoint> -method aadclient -client-id <myClientID> -client-secret "<myClientSecret>" -tenant-id <myAADDirectory>
```

PARAMETER	DESCRIPTION
tenant-id	Azure AD domain, For example, <code>yourdomain.onmicrosoft.com</code>
client-id	Client ID of the registered application in Azure AD
client-secret	Client secret of the registered application in Azure AD

For more information on how to register an application in Azure AD, see [How to: Use the portal to create an Azure AD application and service principal that can access resources](#)

### Connect a mobile device or text browser

For a mobile device or text-based browser where the Azure AD authentication pop-up display is not possible, Azure AD generates a one-time passcode. You can copy the passcode and proceed with Azure AD authentication in another environment.

To generate the passcode, replace **aadauthcode** with **aaddevice**. Replace <myAADDirectory> with an Azure AD domain. For example, `yourdomain.onmicrosoft.com`.

```
connector.exe -remote <myBlockchainEndpoint> -method aaddevice -tenant-id <myAADDirectory>
```

## Next steps

For more information about data security in Azure Blockchain Service, see:

[Azure Blockchain Service security](#)



# Configure Azure Blockchain Service transaction nodes

5/2/2019 • 4 minutes to read • [Edit Online](#)

To interact with Azure Blockchain Service, you do so through connecting to one or more transaction nodes in your blockchain member. In order to interact with transaction nodes, you will need to configure your nodes for access.

## Prerequisites

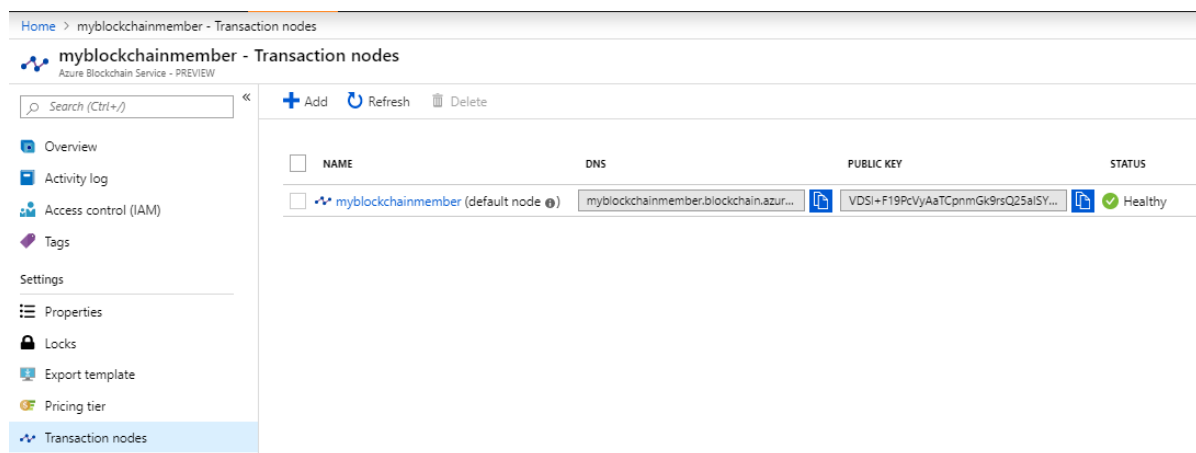
- [Create an Azure Blockchain member](#)

## Transaction node overview

Transaction nodes are used to send blockchain transactions to Azure Blockchain Service through a public endpoint. The default transaction node contains the private key of the Ethereum account registered on the blockchain, and as such cannot be deleted.

To view the default transaction node details:

1. Sign in to the [Azure portal](#).
2. Navigate to your Azure Blockchain Service member. Select **Transaction nodes**.



The screenshot shows the Azure portal interface for the 'myblockchainmember' resource. The left-hand navigation pane is expanded, showing various settings options. The main content area displays the 'Transaction nodes' overview, which includes a table with columns for NAME, DNS, PUBLIC KEY, and STATUS. A single transaction node is listed, identified as 'myblockchainmember (default node)' with a status of 'Healthy'.

NAME	DNS	PUBLIC KEY	STATUS
myblockchainmember (default node)	myblockchainmember.blockchain.azure...	VDSI=F19PcVyAaTCpnmGk9rsQ25aISY...	Healthy

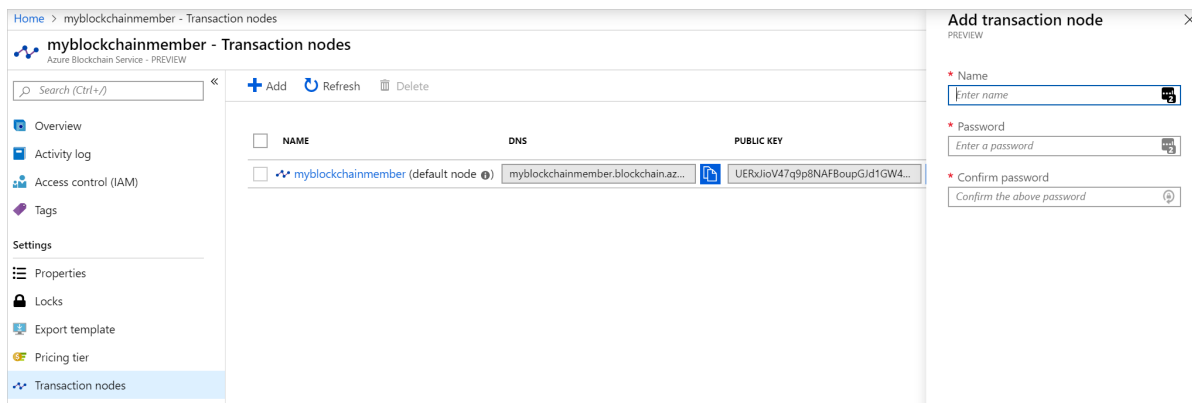
Overview details include public endpoint addresses and public key.

## Create transaction node

You can add up to nine additional transaction nodes to your blockchain member, for a total of ten transaction nodes. By adding transaction nodes, you can increase scalability or distribute load. For example, you could have a transaction node endpoint for different client applications.

To add a transaction node:

1. In the Azure portal, navigate to your Azure Blockchain Service member and select **Transaction nodes > Add**.
2. Complete the settings for the new transaction node.



SETTING	DESCRIPTION
Name	Transaction node name. The name is used to create the DNS address for the transaction node endpoint. For example, <code>newnode-myblockchainmember.blockchain.azure.com</code> . The node name cannot be changed once it is created.
Password	Set a strong password. Use the password to access the transaction node endpoint with basic authentication.

### 3. Select **Create**.

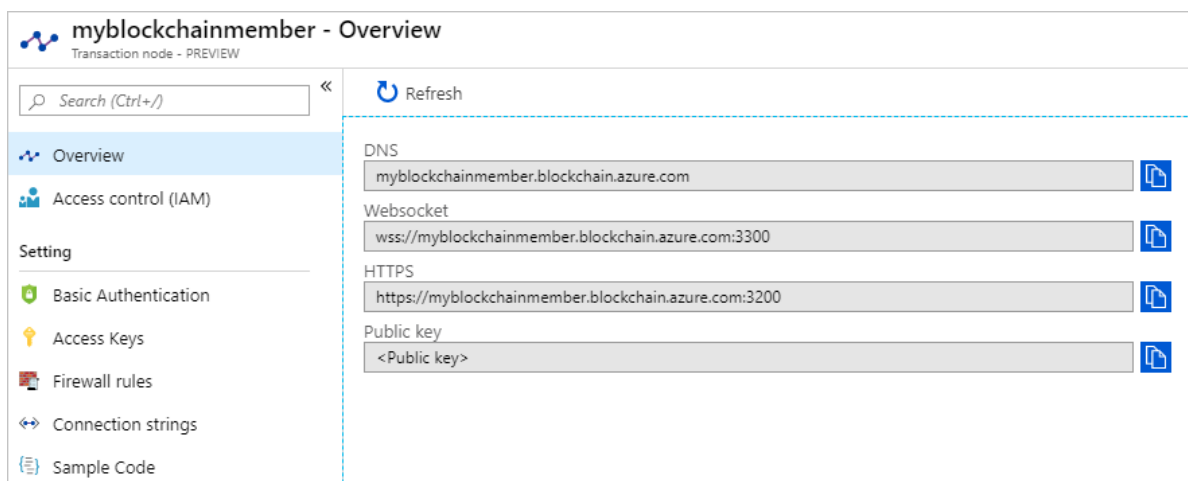
Provisioning a new transaction node takes about 10 minutes. Additional transaction nodes incur cost. For more information on costs, see [Azure pricing](#).

## Endpoints

Transaction nodes have a unique DNS name and public endpoints.

To view a transaction node's endpoint details:

1. In the Azure portal, navigate to one of your Azure Blockchain Service member transaction nodes and select **Overview**.



Transaction node endpoints are secure and require authentication. You can connect to a transaction endpoint using Azure AD authentication, HTTPS basic authentication, and using an access key over HTTPS or Websocket over SSL.

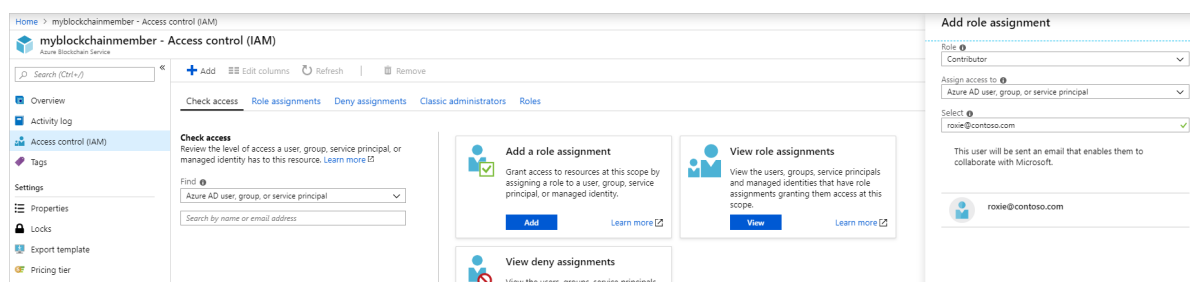
### Azure Active Directory access control

Azure Blockchain Service transaction node endpoints support Azure Active Directory (Azure AD) authentication.

You can grant Azure AD user, group, and service principal access to your endpoint.

To grant Azure AD access control to your endpoint:

1. In the Azure portal, navigate to your Azure Blockchain Service member and select **Transaction nodes > Access control (IAM) > Add > Add role assignment**.
2. Create a new role assignment for a user, group, or service principal (application roles).



SETTING	ACTION
Role	Select <b>Owner</b> , <b>Contributor</b> , or <b>Reader</b> .
Assign access to	Select <b>Azure AD user, group, or service principal</b> .
Select	Search for the user, group, or service principal you want to add.

3. Select **Save** to add the role assignment.

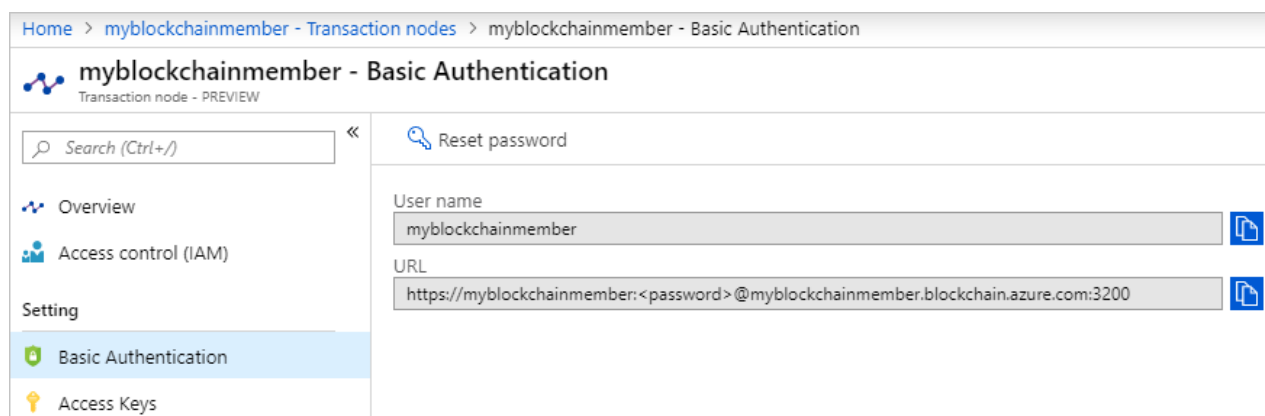
For more information on Azure AD access control, see [Manage access to Azure resources using RBAC and the Azure portal](#)

For details on how to connect using Azure AD authentication, see [connect to your node using AAD authentication](#).

## Basic authentication

For HTTPS basic authentication, user name and password credentials are passed in the HTTPS header of the request to the endpoint.

You can view a transaction node's basic authentication endpoint details in the Azure portal. Navigate to one of your Azure Blockchain Service member transaction nodes and select **Basic Authentication** in settings.



The user name is the name of your node and cannot be changed.

To use the URL, replace <password> with the password set when the node was provisioned. You can update the password by selecting **Reset password**.

## Access keys

For access key authentication, the access key is included in the endpoint URL. When the transaction node is provisioned, two access keys are generated. Either access key can be used for authentication. Two keys enable you change and rotate keys.

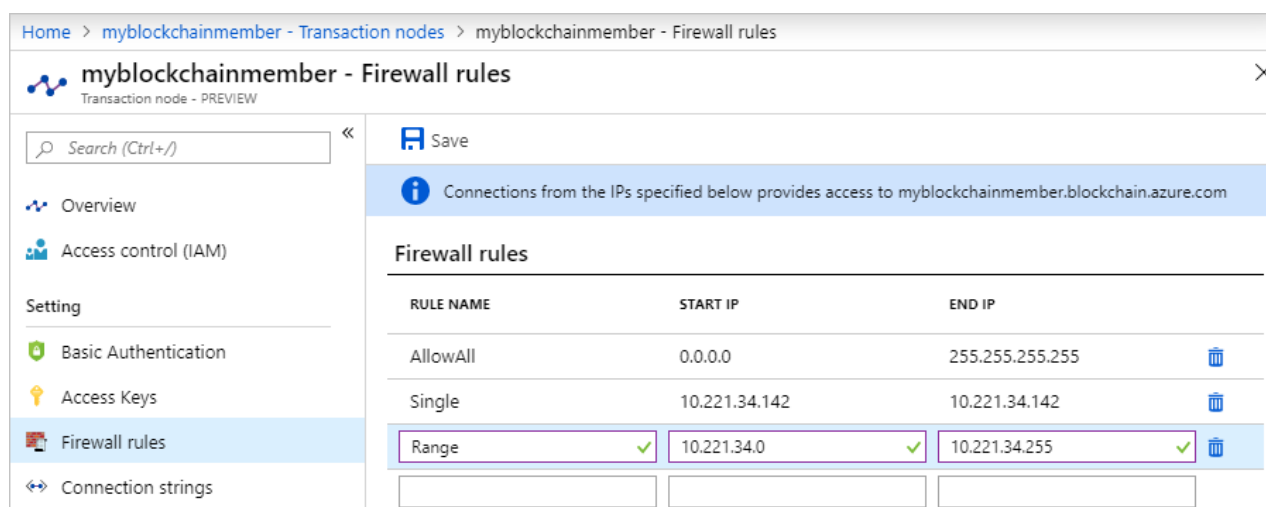
You can view a transaction node's access key details and copy endpoint addresses that include the access keys. Navigate to one of your Azure Blockchain Service member transaction nodes and select **Access Keys** in settings.

## Firewall rules

Firewall rules enable you to limit the IP addresses that can attempt to authenticate to your transaction node. If no firewall rules are configured for your transaction node, it cannot be accessed by any party.

To view a transaction node's firewall rules, navigate to one of your Azure Blockchain Service member transaction nodes and select **Firewall rules** in settings.

You can add firewall rules by entering a rule name, starting IP address, and an ending IP address in the **Firewall rules** grid.



Home > myblockchainmember - Transaction nodes > myblockchainmember - Firewall rules

myblockchainmember - Firewall rules  
Transaction node - PREVIEW

Search (Ctrl+/)

Save

Connections from the IPs specified below provides access to myblockchainmember.blockchain.azure.com

**Firewall rules**

RULE NAME	START IP	END IP
AllowAll	0.0.0.0	255.255.255.255
Single	10.221.34.142	10.221.34.142
Range	10.221.34.0	10.221.34.255

To enable:

- **Single IP address:** Configure the same IP address for the starting and ending IP addresses.
- **IP address range:** Configure the starting and ending IP address range. For example, a range starting at 10.221.34.0 and ending at 10.221.34.255 would enable the entire 10.221.34.xxx subnet.
- **Allow all IP addresses:** Configure the starting IP address to 0.0.0.0 and the ending IP address to 255.255.255.255.

## Connection strings

Connection string syntax for your transaction node is provided for basic authentication or using access keys. Connection strings including access keys over HTTPS and WebSockets are provided.

You can view a transaction node's connection strings and copy endpoint addresses. Navigate to one of your Azure Blockchain Service member transaction nodes and select **Connection strings** in settings.

Home > myblockchainmember - Transaction nodes > myblockchainmember - Connection strings

## myblockchainmember - Connection strings

Transaction node - PREVIEW

Search (Ctrl+/)

- Overview
- Access control (IAM)
- Setting
  - Basic Authentication
  - Access Keys
  - Firewall rules
  - Connection strings**
  - Sample Code

HTTPS (Basic authentication)

```
https://myblockchainmember:<password>@myblockchainmember.f
```

HTTPS (Access key 1)

```
https://myblockchainmember.blockchain.azure.com:3200/hy5FMu5T
```

HTTPS (Access key 2)

```
https://myblockchainmember.blockchain.azure.com:3200/4vwS4CtQ
```

WebSocket (Access key 1)

```
wss://myblockchainmember.blockchain.azure.com:3300/hy5FMu5Ta
```

WebSocket (Access key 2)

```
wss://myblockchainmember.blockchain.azure.com:3300/4vwS4CtQC
```

## Sample code

Sample code is provided to quickly enable connecting to your transaction node via Web3, Nethereum, Web3js, and Truffle.

You can view a transaction node's sample connection code and copy it to use with popular developer tools. Navigate to one of your Azure Blockchain Service member transaction nodes and select **Sample Code** in settings.

Choose the Web3 or Nethereum tab to view the code sample you want to use.

Home > myblockchainmember - Transaction nodes > myblockchainmember - Sample Code

## myblockchainmember - Sample Code

Transaction node - PREVIEW

Search (Ctrl+/)

- Overview
- Access control (IAM)
- Setting
  - Basic Authentication
  - Access Keys
  - Firewall rules
  - Connection strings
  - Sample Code**

Web3 **Nethereum**

HTTPS (Basic authentication)

```
var Web3 = require("Web3");
var provider = new Web3.providers.HttpProvider("https://myblockchainmember:<password>@myblockchainmember.bl");
var web3 = new Web3(provider);
```

HTTPS (Access key 1)

```
var Web3 = require("Web3");
var provider = new Web3.providers.HttpProvider("https://myblockchainmember.blockchain.azure.com:3200/hy5FMu5Ta");
var web3 = new Web3(provider);
```

HTTPS (Access key 2)

```
var Web3 = require("Web3");
var provider = new Web3.providers.HttpProvider("https://myblockchainmember.blockchain.azure.com:3200/4vwS4CtQC");
var web3 = new Web3(provider);
```

## Next steps

[Configure transaction nodes using Azure CLI](#)

# Manage Azure Blockchain Service with Azure CLI

5/6/2019 • 7 minutes to read • [Edit Online](#)

In addition to the Azure portal, you can use Azure CLI to quickly create and manage blockchain members and transaction nodes for your Azure Blockchain Service.

Make sure that you have installed the latest [Azure CLI](#) and logged in to an Azure account in with `az login`.

In the following examples, replace example `<parameter names>` with your own values.

## Create blockchain member

Example creates a blockchain member in Azure Blockchain Service that runs the Quorum ledger protocol in a new consortium.

```
az resource create --resource-group <myResourceGroup> --name <myMemberName> --resource-type
Microsoft.Blockchain/blockchainMembers --is-full-object --properties "{ \"location\": \"
<myBlockchainLocation>\", \"properties\": {\"password\": \"<myStrongPassword>\", \"protocol\": \"Quorum\",
\"consortium\": \"<myConsortiumName>\", \"consortiumManagementAccountPassword\": \"
<myConsortiumManagementAccountPassword>\", \"firewallRules\": [ { \"ruleName\": \"<myRuleName>\",
\"startIpAddress\": \"<myStartIpAddress>\", \"endIpAddress\": \"<myEndIpAddress>\" } ] }, \"sku\": { \"name\":
\"<skuName>\" } }"
```

PARAMETER	DESCRIPTION
<b>resource-group</b>	Resource group name where Azure Blockchain Service resources are created.
<b>name</b>	A unique name that identifies your Azure Blockchain Service blockchain member. The name is used for the public endpoint address. For example, <code>myblockchainmember.blockchain.azure.com</code> .
<b>location</b>	Azure region where the blockchain member is created. For example, <code>eastus</code> . Choose the location that is closest to your users or your other Azure applications.
<b>password</b>	The member account password. The member account password is used to authenticate to the blockchain member's public endpoint using basic authentication. The password must meet three of the following four requirements: length needs to be between 12 & 72 characters, 1 lower case character, 1 upper case character, 1 number, and 1 special character that is not number sign(#), percent(%), comma(,), star(*), back quote(`), double quote("), single quote('), dash(-) and semicolon(;)
<b>protocol</b>	Public preview supports Quorum.
<b>consortium</b>	Name of the consortium to join or create.
<b>consortiumManagementAccountPassword</b>	The consortium management password. The password is used for joining a consortium.

PARAMETER	DESCRIPTION
<b>ruleName</b>	Rule name for whitelisting an IP address range. Optional parameter for firewall rules.
<b>startIpAddress</b>	Start of the IP address range for whitelisting. Optional parameter for firewall rules.
<b>endIpAddress</b>	End of the IP address range for whitelisting. Optional parameter for firewall rules.
<b>skuName</b>	Tier type. Use S0 for Standard and B0 for Basic.

## Change blockchain member password

Example changes a blockchain member's password.

```
az resource update --resource-group <myResourceGroup> --name <myMemberName> --resource-type
Microsoft.Blockchain/blockchainMembers --set properties.password="<myStrongPassword>" --remove
properties.consortiumManagementAccountAddress
```

PARAMETER	DESCRIPTION
<b>resource-group</b>	Resource group name where Azure Blockchain Service resources are created.
<b>name</b>	Name that identifies your Azure Blockchain Service member.
<b>password</b>	The member account password. The password must meet three of the following four requirements: length needs to be between 12 & 72 characters, 1 lower case character, 1 upper case character, 1 number, and 1 special character that is not number sign(#), percent(%), comma(,), star(*), back quote(`), double quote("), single quote('), dash(-) and semicolon(;).

## Create transaction node

Create a transaction node inside an existing blockchain member. By adding transaction nodes, you can increase security isolation and distribute load. For example, you could have a transaction node endpoint for different client applications.

```
az resource create --resource-group <myResourceGroup> --name
<myMemberName>/transactionNodes/<myTransactionNode> --resource-type Microsoft.Blockchain/blockchainMembers --
is-full-object --properties "{ \"location\": \"<myRegion>\", \"properties\": { \"password\": \"
<myStrongPassword>\", \"firewallRules\": [ { \"ruleName\": \"<myRuleName>\", \"startIpAddress\": \"
<myStartIpAddress>\", \"endIpAddress\": \"<myEndIpAddress>\" } ] } }"
```

PARAMETER	DESCRIPTION
<b>resource-group</b>	Resource group name where Azure Blockchain Service resources are created.

PARAMETER	DESCRIPTION
<b>name</b>	Name of the Azure Blockchain Service blockchain member that also includes the new transaction node name.
<b>location</b>	Azure region where the blockchain member is created. For example, <code>eastus</code> . Choose the location that is closest to your users or your other Azure applications.
<b>password</b>	The transaction node password. The password must meet three of the following four requirements: length needs to be between 12 & 72 characters, 1 lower case character, 1 upper case character, 1 number, and 1 special character that is not number sign(#), percent(%), comma(,), star(*), back quote(`), double quote("), single quote('), dash(-) and semicolon(;).
<b>ruleName</b>	Rule name for whitelisting an IP address range. Optional parameter for firewall rules.
<b>startIpAddress</b>	Start of the IP address range for whitelisting. Optional parameter for firewall rules.
<b>endIpAddress</b>	End of the IP address range for whitelisting. Optional parameter for firewall rules.

## Change transaction node password

Example changes a transaction node password.

```
az resource update --resource-group <myResourceGroup> --name
<myMemberName>/transactionNodes/<myTransactionNode> --resource-type Microsoft.Blockchain/blockchainMembers --
set properties.password="<myStrongPassword>"
```

PARAMETER	DESCRIPTION
<b>resource-group</b>	Resource group name where Azure Blockchain Service resources exist.
<b>name</b>	Name of the Azure Blockchain Service blockchain member that also includes the new transaction node name.
<b>password</b>	The transaction node password. The password must meet three of the following four requirements: length needs to be between 12 & 72 characters, 1 lower case character, 1 upper case character, 1 number, and 1 special character that is not number sign(#), percent(%), comma(,), star(*), back quote(`), double quote("), single quote('), dash(-) and semicolon(;).

## Change consortium management account password

The consortium management account is used for consortium membership management. Each member is uniquely identified by a consortium management account and you can change the password of this account with the following command.



```
az resource update --resource-group <myResourceGroup> --name <myMemberName> --resource-type
Microsoft.Blockchain/blockchainMembers --set properties.consortiumManagementAccountPassword="
<myConsortiumManagementAccountPassword>" --remove properties.consortiumManagementAccountAddress
```

PARAMETER	DESCRIPTION
<b>resource-group</b>	Resource group name where Azure Blockchain Service resources are created.
<b>name</b>	Name that identifies your Azure Blockchain Service member.
<b>consortiumManagementAccountPassword</b>	The consortium management account password. The password must meet three of the following four requirements: length needs to be between 12 & 72 characters, 1 lower case character, 1 upper case character, 1 number, and 1 special character that is not number sign(#), percent(%), comma(,), star(*), back quote(`), double quote("), single quote('), dash(-) and semicolon(;).

## Update firewall rules

```
az resource update --resource-group <myResourceGroup> --name <myMemberName> --resource-type
Microsoft.Blockchain/blockchainMembers --set properties.firewallRules="[ { \"ruleName\": \"<myRuleName>\",
\"startIpAddress\": \"<myStartIpAddress>\", \"endIpAddress\": \"<myEndIpAddress>\" } ]" --remove
properties.consortiumManagementAccountAddress
```

PARAMETER	DESCRIPTION
<b>resource-group</b>	Resource group name where Azure Blockchain Service resources exist.
<b>name</b>	Name of the Azure Blockchain Service blockchain member.
<b>ruleName</b>	Rule name for whitelisting an IP address range. Optional parameter for firewall rules.
<b>startIpAddress</b>	Start of the IP address range for whitelisting. Optional parameter for firewall rules.
<b>endIpAddress</b>	End of the IP address range for whitelisting. Optional parameter for firewall rules.

## List API keys

API keys can be used for node access similar to user name and password. There are two API keys to support key rotation. Use the following command to list your API keys.

```
az resource invoke-action --resource-group <myResourceGroup> --name
<myMemberName>/transactionNodes/<myTransactionNode> --action "listApiKeys" --resource-type
Microsoft.Blockchain/blockchainMembers
```

PARAMETER	DESCRIPTION
<b>resource-group</b>	Resource group name where Azure Blockchain Service resources exist.
<b>name</b>	Name of the Azure Blockchain Service blockchain member that also includes the new transaction node name.

## Regenerate API keys

Use the following command to regenerate your API keys.

```
az resource invoke-action --resource-group <myResourceGroup> --name
<myMemberName>/transactionNodes/<myTransactionNode> --action "regenerateApiKeys" --resource-type
Microsoft.Blockchain/blockchainMembers --request-body '{"keyName":"<keyValue>"}'
```

PARAMETER	DESCRIPTION
<b>resource-group</b>	Resource group name where Azure Blockchain Service resources exist.
<b>name</b>	Name of the Azure Blockchain Service blockchain member that also includes the new transaction node name.
<b>keyName</b>	Replace <keyValue> with either key1 or key2.

## Delete a transaction node

Example deletes a blockchain member transaction node.

```
az resource delete --resource-group <myResourceGroup> --name
<myMemberName>/transactionNodes/<myTransactionNode> --resource-type Microsoft.Blockchain/blockchainMembers
```

PARAMETER	DESCRIPTION
<b>resource-group</b>	Resource group name where Azure Blockchain Service resources exist.
<b>name</b>	Name of the Azure Blockchain Service blockchain member that also includes the new transaction node name to be deleted.

## Delete a blockchain member

Example deletes a blockchain member.

```
az resource delete --resource-group <myResourceGroup> --name <myMemberName> --resource-type
Microsoft.Blockchain/blockchainMembers
```

PARAMETER	DESCRIPTION
-----------	-------------

PARAMETER	DESCRIPTION
<b>resource-group</b>	Resource group name where Azure Blockchain Service resources exist.
<b>name</b>	Name of the Azure Blockchain Service blockchain member to be deleted.

## Azure Active Directory

### Grant access for Azure AD user

```
az role assignment create --role <role> --assignee <assignee> --scope
/subscriptions/<subId>/resourceGroups/<groupName>/providers/Microsoft.Blockchain/blockchainMembers/<myMemberName>
```

PARAMETER	DESCRIPTION
<b>role</b>	Name of the Azure AD role.
<b>assignee</b>	Azure AD user ID. For example, <code>user@contoso.com</code>
<b>scope</b>	Scope of the role assignment. Can be either a blockchain member of transaction node.

### Example:

Grant node access for Azure AD user to blockchain **member**:

```
az role assignment create \
  --role "myRole" \
  --assignee user@contoso.com \
  --scope
/subscriptions/mySubscriptionId/resourceGroups/contosoResourceGroup/providers/Microsoft.Blockchain/blockchainMembers/contosoMember1
```

### Example:

Grant node access for Azure AD user to blockchain **transaction node**:

```
az role assignment create \
  --role "MyRole" \
  --assignee user@contoso.com \
  --scope
/subscriptions/mySubscriptionId/resourceGroups/contosoResourceGroup/providers/Microsoft.Blockchain/blockchainMembers/contosoMember1/transactionNodes/contosoTransactionNode1
```

### Grant node access for Azure AD group or application role

```
az role assignment create --role <role> --assignee-object-id <assignee_object_id>
```

PARAMETER	DESCRIPTION
<b>role</b>	Name of the Azure AD role.
<b>assignee-object-id</b>	Azure AD group ID or application ID.
<b>scope</b>	Scope of the role assignment. Can be either a blockchain member of transaction node.

### Example:

Grant node access for **application role**

```
az role assignment create \
  --role "myRole" \
  --assignee-object-id 22222222-2222-2222-222222222222 \
  --scope
/subscriptions/mySubscriptionId/resourceGroups/contosoResourceGroup/providers/Microsoft.Blockchain/blockchainMembers/contosoMember1
```

### Remove Azure AD node access

```
az role assignment delete --role <myRole> --assignee <assignee> --scope
/subscriptions/mySubscriptionId/resourceGroups/<myResourceGroup>/providers/Microsoft.Blockchain/blockchainMembers/<myMemberName>/transactionNodes/<myTransactionNode>
```

PARAMETER	DESCRIPTION
<b>role</b>	Name of the Azure AD role.
<b>assignee</b>	Azure AD user ID. For example, <code>user@contoso.com</code>
<b>scope</b>	Scope of the role assignment. Can be either a blockchain member of transaction node.

## Next steps

[Configure Azure Blockchain Service transaction nodes with the Azure portal](#)

# Manage consortium members in Azure Blockchain Service by using PowerShell

7/16/2019 • 5 minutes to read • [Edit Online](#)

You can use PowerShell to manage blockchain consortium members for your Azure Blockchain Service. Members who have administrator privileges can invite, add, remove, and change roles for all participants in the blockchain consortium. Members who have user privileges can view all participants in the blockchain consortium and change their member display name.

## Prerequisites

- Create a blockchain member by using the [Azure portal](#).
- For more information about consortia, members, and nodes, see [Azure Blockchain Service consortium](#).

## Open Azure Cloud Shell

Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

You can also open Cloud Shell in a separate browser tab by going to [shell.azure.com/powershell](https://shell.azure.com/powershell). Select **Copy** to copy the blocks of code, paste it into Cloud Shell, and select **Enter** to run it.

## Install the PowerShell module

Install the Microsoft.AzureBlockchainService.ConsortiumManagement.PS package from the PowerShell Gallery.

```
Install-Module -Name Microsoft.AzureBlockchainService.ConsortiumManagement.PS -Scope CurrentUser
Import-Module Microsoft.AzureBlockchainService.ConsortiumManagement.PS
```

## Set the information preference

You can get more information when executing the cmdlets by setting the information preference variable. By default, *\$InformationPreference* is set to *SilentlyContinue*.

For more verbose information from cmdlets, set the preference in the PowerShell as follows:

```
$InformationPreference = 'Continue'
```

## Establish a Web3 connection

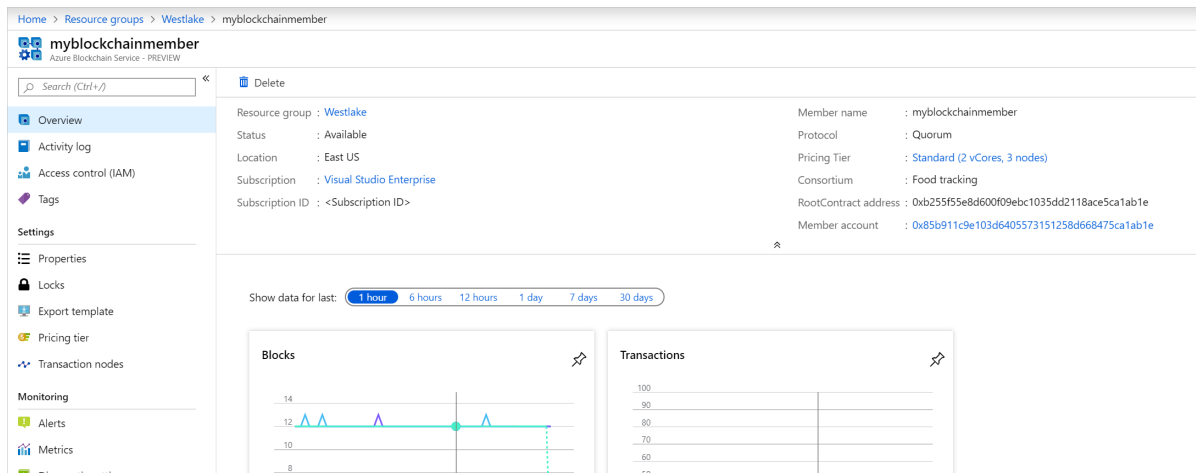
To manage consortium members, establish a Web3 connection to your Blockchain Service member endpoint. You can use this script to set global variables for calling the consortium management cmdlets.

```
$Connection = New-Web3Connection -RemoteRPCEndpoint '<Endpoint address>'
$MemberAccount = Import-Web3Account -ManagedAccountAddress '<Member account address>' -ManagedAccountPassword '<Member account password>'
$ContractConnection = Import-ConsortiumManagementContracts -RootContractAddress '<RootContract address>' -Web3Client $Connection
```

Replace *<Member account password>* with the member account password that you used when you created the member.

Find the other values in the Azure portal:

1. Sign in to the [Azure portal](#).
2. Go to your default Blockchain Service member **Overview** page.



Replace *<Member account>* and *<RootContract address>* with the values from the portal.

3. For the endpoint address, select **Transaction nodes**, and then select the **default transaction node**. The default node has the same name as the blockchain member.
4. Select **Connection strings**.

Home > myblockchainmember - Transaction nodes > myblockchainmember - Connection strings

**myblockchainmember - Connection strings**  
Transaction node - PREVIEW

Search (Ctrl+/) <<

Overview

Access control (IAM)

Setting

Basic Authentication

Access Keys

Firewall rules

Connection strings

Sample Code

HTTPS (Basic authentication)

https://myblockchainmember:<password>@myblockchainmember.l

HTTPS (Access key 1)

https://myblockchainmember.blockchain.azure.com:3200/hy5FMu5T

HTTPS (Access key 2)

https://myblockchainmember.blockchain.azure.com:3200/4vwS4CtQ

WebSocket (Access key 1)

wss://myblockchainmember.blockchain.azure.com:3300/hy5FMu5Ta

WebSocket (Access key 2)

wss://myblockchainmember.blockchain.azure.com:3300/4vwS4CtQC

Replace *<Endpoint address>* with the value from **HTTPS (Access key 1)** or **HTTPS (Access key 2)**.

## Manage the network and smart contracts

Use the network and smart contract cmdlets to establish a connection to the blockchain endpoint's smart contracts responsible for consortium management.

## Import-ConsortiumManagementContracts

Use this cmdlet to connect to the consortium management's smart contracts. These contracts are used to manage and enforce members within the consortium.

```
Import-ConsortiumManagementContracts -RootContractAddress <String> -Web3Client <IClient>
```

PARAMETER	DESCRIPTION	REQUIRED
RootContractAddress	Root contract address of the consortium management smart contracts	Yes
Web3Client	Web3Client object obtained from New-Web3Connection	Yes

### Example

```
Import-ConsortiumManagementContracts -RootContractAddress '<RootContract address>' -Web3Client $Connection
```

## Import-Web3Account

Use this cmdlet to create an object to hold the information for a remote node's management account.

```
Import-Web3Account -ManagedAccountAddress <String> -ManagedAccountPassword <String>
```

PARAMETER	DESCRIPTION	REQUIRED
ManagedAccountAddress	Blockchain member account address	Yes
ManagedAccountPassword	Account address password	Yes

### Example

```
Import-Web3Account -ManagedAccountAddress '<Member account address>' -ManagedAccountPassword '<Member account password>'
```

## New-Web3Connection

Use this cmdlet to establish a connection to the RPC endpoint of a transaction node.

```
New-Web3Connection [-RemoteRPCEndpoint <String>]
```

PARAMETER	DESCRIPTION	REQUIRED
RemoteRPCEndpoint	Blockchain member endpoint address	Yes

### Example

```
New-Web3Connection -RemoteRPCEndpoint '<Endpoint address>'
```

# Manage the consortium members

Use consortium member management cmdlets to manage members within the consortium. The available actions depend on your consortium role.

## Get-BlockchainMember

Use this cmdlet to get member details or list members of the consortium.

```
Get-BlockchainMember [[-Name] <String>] -Members <IContract> -Web3Client <IClient>
```

PARAMETER	DESCRIPTION	REQUIRED
Name	The name of the Blockchain Service member that you want to retrieve details about. When a name is entered, it returns the member's details. When a name is omitted, it returns a list of all consortium members.	No
Members	Members object obtained from Import-ConsortiumManagementContracts	Yes
Web3Client	Web3Client object obtained from New-Web3Connection	Yes

#### Example

```
$ContractConnection | Get-BlockchainMember -Name <Member Name>
```

#### Example output

```
Name           : myblockchainmember
CorrelationId   : 0
DisplayName     : myCompany
SubscriptionId  : <Azure subscription ID>
AccountAddress : 0x85b911c9e103d6405573151258d668479e9ebeef
Role           : ADMIN
```

### Remove-BlockchainMember

Use this cmdlet to remove a blockchain member.

```
Remove-BlockchainMember -Name <String> -Members <IContract> -Web3Account <IAccount> -Web3Client <IClient>
```

PARAMETER	DESCRIPTION	REQUIRED
Name	Member name to remove	Yes
Members	Members object obtained from Import-ConsortiumManagementContracts	Yes
Web3Account	Web3Account object obtained from Import-Web3Account	Yes
Web3Client	Web3Client object obtained from New-Web3Connection	Yes

#### Example

```
$ContractConnection | Remove-BlockchainMember -Name <Member Name> -Web3Account $MemberAccount
```

### Set-BlockchainMember

Use this cmdlet to set blockchain member attributes, including the display name and the consortium role.



Consortium administrators can set **DisplayName** and **Role** for all members. A consortium member with the user role can change only their own member's display name.

```
Set-BlockchainMember -Name <String> [-DisplayName <String>] [-AccountAddress <String>] [-Role <String>]  
-Members <IContract> -Web3Account <IAccount> -Web3Client <IClient>
```

PARAMETER	DESCRIPTION	REQUIRED
Name	Name of the blockchain member	Yes
DisplayName	New display name	No
AccountAddress	Account address	No
Members	Members object obtained from Import-ConsortiumManagementContracts	Yes
Web3Account	Web3Account object obtained from Import-Web3Account	Yes
Web3Client	Web3Client object obtained from New-Web3Connection	Yes

#### Example

```
$ContractConnection | Set-BlockchainMember -Name <Member Name> -DisplayName <Display name> -Web3Account  
$MemberAccount
```

## Manage the consortium members' invitations

Use the consortium member invitation management cmdlets to manage consortium members' invitations. The available actions depend on your consortium role.

### New-BlockchainMemberInvitation

Use this cmdlet to invite new members to the consortium.

```
New-BlockchainMemberInvitation -SubscriptionId <String> -Role <String> -Members <IContract>  
-Web3Account <IAccount> -Web3Client <IClient>
```

PARAMETER	DESCRIPTION	REQUIRED
SubscriptionId	Azure subscription ID of the member to invite	Yes
Role	The consortium role. Values can be ADMIN or USER. ADMIN is the consortium administrator role. USER is the consortium member role.	Yes
Members	Members object obtained from Import-ConsortiumManagementContracts	Yes

PARAMETER	DESCRIPTION	REQUIRED
Web3Account	Web3Account object obtained from Import-Web3Account	Yes
Web3Client	Web3Client object obtained from New-Web3Connection	Yes

#### Example

```
$ContractConnection | New-BlockchainMemberInvitation -SubscriptionId <Azure Subscription ID> -Role USER -Web3Account $MemberAccount
```

### Get-BlockchainMemberInvitation

Use this cmdlet to retrieve or list a consortium member's invitation status.

```
Get-BlockchainMemberInvitation [[-SubscriptionId] <String>] -Members <IContract> -Web3Client <IClient>
```

PARAMETER	DESCRIPTION	REQUIRED
SubscriptionId	The Azure subscription ID of the member to invite. If the subscription ID is provided, it returns the subscription ID's invitation details. If the subscription ID is omitted, it returns a list of all member invitations.	No
Members	Members object obtained from Import-ConsortiumManagementContracts	Yes
Web3Client	Web3Client object obtained from New-Web3Connection	Yes

#### Example

```
$ContractConnection | Get-BlockchainMemberInvitation - SubscriptionId <Azure subscription ID>
```

#### Example output

```
SubscriptionId      Role CorrelationId
-----
<Azure subscription ID>  USER          2
```

### Remove-BlockchainMemberInvitation

Use this cmdlet to revoke a consortium member's invitation.

```
Remove-BlockchainMemberInvitation -SubscriptionId <String> -Members <IContract> -Web3Account <IAccount> -Web3Client <IClient>
```

PARAMETER	DESCRIPTION	REQUIRED
SubscriptionId	Azure subscription ID of the member to revoke	Yes

PARAMETER	DESCRIPTION	REQUIRED
Members	Members object obtained from Import-ConsortiumManagementContracts	Yes
Web3Account	Web3Account object obtained from Import-Web3Account	Yes
Web3Client	Web3Client object obtained from New-Web3Connection	Yes

#### Example

```
$ContractConnection | Remove-BlockchainMemberInvitation -SubscriptionId <Subscription ID> -Web3Account $MemberAccount
```

### Set-BlockchainMemberInvitation

Use this cmdlet to set the **Role** for an existing invitation. Only consortium administrators can change invitations.

```
Set-BlockchainMemberInvitation -SubscriptionId <String> -Role <String> -Members <IContract> -Web3Account <IAccount> -Web3Client <IClient>
```

PARAMETER	DESCRIPTION	REQUIRED
SubscriptionId	Azure subscription ID of the member to invite	Yes
Role	New consortium role for invitation. Values can be <b>USER</b> or <b>ADMIN</b> .	Yes
Members	Members object obtained from Import-ConsortiumManagementContracts	Yes
Web3Account	Web3Account object obtained from Import-Web3Account	Yes
Web3Client	Web3Client object obtained from New-Web3Connection	Yes

#### Example

```
$ContractConnection | Set-BlockchainMemberInvitation -SubscriptionId <Azure subscription ID> -Role USER -Web3Account $MemberAccount
```

## Next steps

For more information about consortia, members, and nodes, see:

[Azure Blockchain Service consortium](#)