Contents

What is Azure Blockchain Service?

5/11/2021 • 5 minutes to read • Edit Online

Azure Blockchain Service is a fully managed ledger service that gives users the ability to grow and operate blockchain networks at scale in Azure.

> **IMPORTANT**
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

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 Istanbul Byzantine Fault Tolerance (IBFT) consensus mechanism.

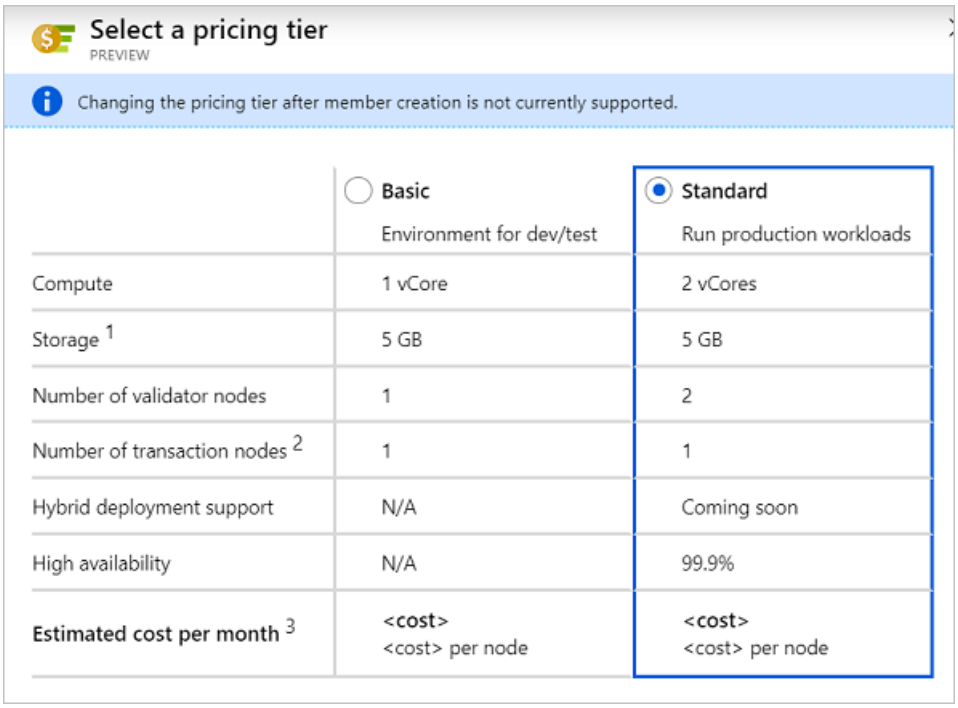
These capabilities require almost no administration and all are provided at no additional cost. You can focus on app development and business logic rather than allocating time and resources to managing virtual machines and infrastructure. In addition, you can continue to develop your application with 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 is done through the Azure portal, Azure CLI, or 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. Consortiums 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

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. Use the *Basic* tier for development, testing, and proof of concepts. Use the *Standard* tier for production grade deployments. Both tiers include at least one transaction node, and one validator node (Basic) or two validator nodes (Standard).



In addition to offering two validator nodes, the *Standard* tier provides two *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.

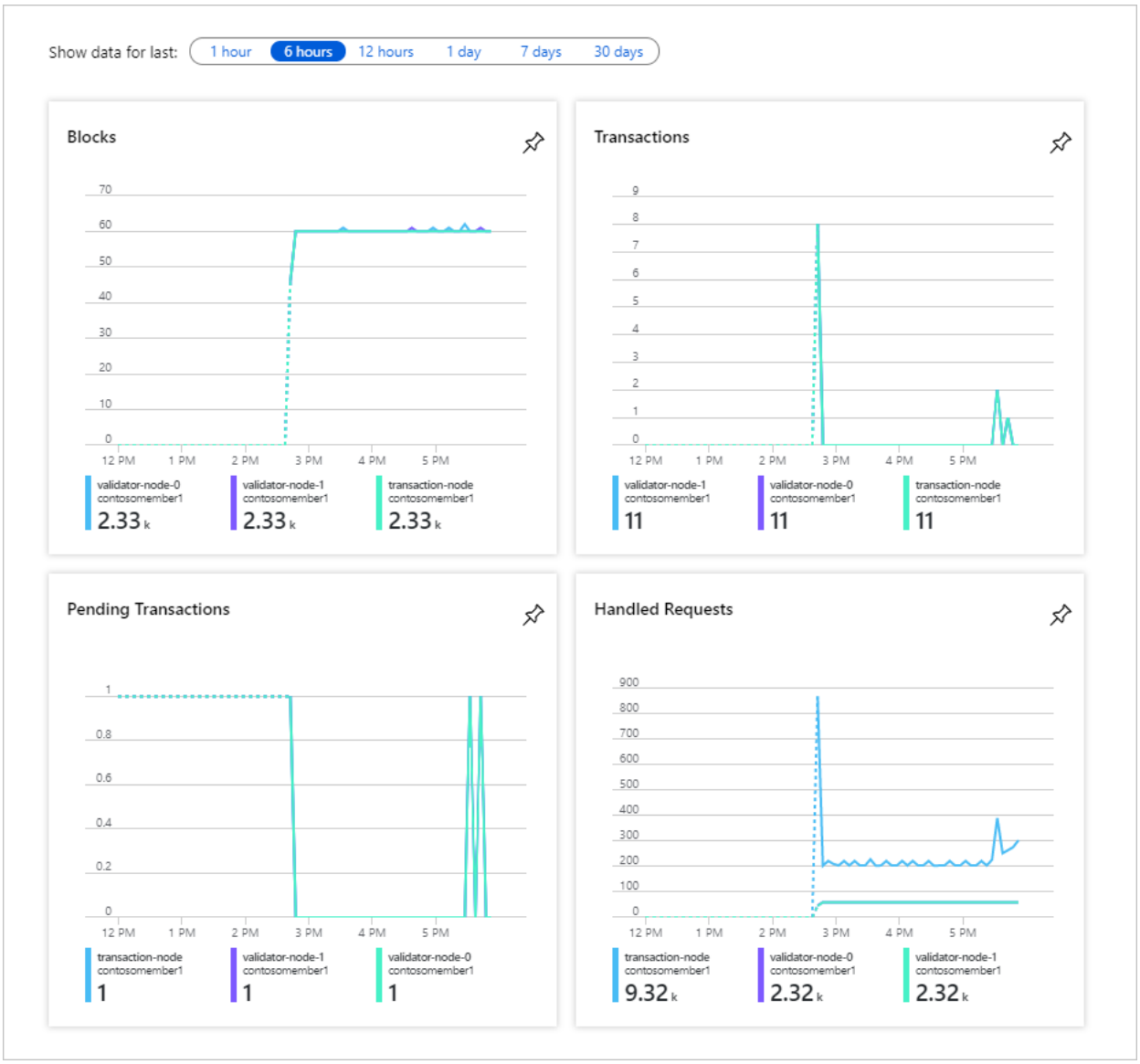
Security and maintenance

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 require configuration 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, and 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. Azure Monitor also provides helpful insights into blockchain network activity such as transactions and blocks mined, transaction queue depth, and 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.

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 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.

Publish blockchain data

Blockchain Data Manager for Azure Blockchain Service captures, transforms, and delivers Azure Blockchain Service transaction data to Azure Event Grid Topics providing reliable and scalable blockchain ledger integration with Azure services. You can use Blockchain Data Manager to integrate off-chain applications and data stores. For more information, see Blockchain Data Manager for Azure Blockchain Service.

Support and feedback

For Azure Blockchain news, visit the Azure Blockchain blog to stay up to date on blockchain service offerings and information from the Azure Blockchain engineering team.

To provide product feedback or to request new features, post or vote for an idea via the Azure feedback forum for blockchain.

Community support

Engage with Microsoft engineers and Azure Blockchain community experts.

- Microsoft Q&A question page for Azure Blockchain Service
- Microsoft Tech Community
- Stack Overflow

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
- Follow the Microsoft Learn path Get started with blockchain development
- Watch the Beginner's series to blockchain
- 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

What's new in Azure Blockchain Service?

5/11/2021 • 5 minutes to read • Edit Online

Get notified about when to revisit this page for updates by copying and pasting this URL:

```
https://docs.microsoft.com/api/search/rss?search=%22Release+notes+-
+Azure+Blockchain+Service%22&locale=en-us
```

into your RSS feed reader  .

Azure Blockchain Service receives improvements on an ongoing basis. To stay up to date with the most recent developments, this article provides you with information about:

- New capabilities
- Version upgrades
- Known issues

May 2021

> **IMPORTANT**
>
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

June 2020

Version upgrades

- Quorum version upgrade to 2.6.0. With version 2.6.0, you can send signed private transactions. For more information on sending private transactions, see the Quorum API documentation.
- Tessera version upgrade to 0.10.5.

Contract size and transaction size increased to 128 KB

Type: Configuration change

Contract size (MaxCodeSize) was increased to 128 KB so that you can deploy larger size smart contracts. Also, transaction size (txnSizeLimit) was increased to 128 KB. Configuration changes apply to new consortiums created on Azure Blockchain Service after June 19 2020.

TrieTimeout value reduced

Type: Configuration change

The TrieTimeout value was reduced so that in-memory state is written to disk more frequently. The lower value ensures faster recovery of a node in the rare case of a node crash.

May 2020

Version upgrades

- Ubuntu version upgrade to 18.04
- Quorum version upgrade to 2.5.0

- Tessera version upgrade 0.10.4

Azure Blockchain Service supports sending rawPrivate transactions

Type: Feature

Customers can sign private transactions outside of the account on the node.

Two-phase member provisioning

Type: Enhancement

Two phases help optimize scenarios where a member is being created in a long existing consortium. The member infrastructure is provisioned in first phase. In the second phase, the member is synchronized with blockchain. Two-phase provisioning helps avoid member creation failure due to timeouts.

Known issues

eth.estimateGas function throws exception in Quorum v2.6.0

In Quorum v2.6.0, calls to *eth.estimateGas* function without providing the additional *value* parameter cause a *method handler crashed* exception. The Quorum team has been notified and a fix is expected end of July 2020. You can use the following workarounds until a fix is available:

- Avoid using *eth.estimateGas* since it can affect performance. For more information about eth.estimateGas performance issues, see Calling eth.estimateGas function reduces performance. Include a gas value for each transaction. Most libraries will call eth.estimateGas if a gas value is not provided which causes Quorum v2.6.0 to crash.
- If you need to call *eth.estimateGas*, the Quorum team suggests you pass the additional parameter *value* as *0* as a workaround.

Mining stops if fewer than four validator nodes

Production networks should have at least four validator nodes. Quorum recommends at least four validator nodes are required to meet the IBFT crash fault tolerance (3F+1). You should have at least two Azure Blockchain Service *Standard* tier nodes to get four validator nodes. A standard node is provisioned with two validator nodes.

If the Blockchain network on Azure Blockchain Service doesn't have four validator nodes, then mining might stop on the network. You can detect mining has stopped by setting an alert on processed blocks. In a healthy network, processed block will be 60 blocks per node per five minutes.

As a mitigation, the Azure Blockchain Service team has to restart the node. Customers need to open a support request to restart the node. The Azure Blockchain Service team is working toward detecting and fixing mining issues automatically.

Use the *Standard* tier for production grade deployments. Use the *Basic* tier for development, testing, and proof of concepts. Changing the pricing tier between basic and standard after member creation is not supported.

Blockchain Data Manager requires Standard tier node

Use the *Standard* tier if you are using Blockchain Data Manager. The *Basic* tier has 4-GB memory only. Hence, it is not able to scale to the usage required by Blockchain Data Manager and other services running on it.

Use the *Basic* tier for development, testing, and proof of concepts. Changing the pricing tier between basic and standard after member creation is not supported.

Large volume of unlock account calls causes geth to crash

A large volume of unlock account calls while submitting transaction can cause geth to crash on a transaction node. As a result, you have to stop ingesting transactions. Otherwise, the pending transaction queue increases.

Geth restarts automatically within less than a minute. Depending on the node, the syncing might take a long time. The Azure Blockchain Service team is enabling an archiving feature that will reduce the time to sync.

To identify geth crashes, you can check logs for any error message in Blockchain messages in application logs. You can also check if processed blocks decrease while pending transactions increase.

To mitigate the issue, send signed transactions instead of sending unsigned transactions with a command to unlock the account. For transactions that are already signed externally, there is no need to unlock the account.

If you want to send unsigned transactions, unlock the account for infinite time by sending 0 as the time parameter in the unlock command. You can lock the account back after all the transactions are submitted.

The following are the geth parameters that Azure Blockchain Service uses. You cannot adjust these parameters.

- Istanbul block period: 5 secs
- Floor gas limit: 700000000
- Ceil gas limit: 800000000

Large volume of private transactions reduces performance

If you are using Azure Blockchain Service Basic tier and private transactions, Tessera may crash.

You can detect the Tessera crash by reviewing the Blockchain application logs and searching for the message `Tesseracrashed.RestartingTessera...`.

Azure Blockchain Service restarts Tessera when there is a crash. Restart takes about a minute.

Use the *Standard* tier if you are sending a high volume of private transactions. Use the *Basic* tier for development, testing, and proof of concepts. Changing the pricing tier between basic and standard after member creation is not supported.

Calling eth.estimateGas function reduces performance

Calling *eth.estimateGas* function multiple times reduces transactions per second drastically. Do not use *eth.estimateGas* function for each transaction submission. The *eth.estimateGas* function is memory intensive.

If possible, use a conservative gas value for submitting transactions and minimize the use of *eth.estimateGas*.

Unbounded loops in smart contracts reduces performance

Avoid unbounded loops in smart contracts as they can reduce performance. For more information, see the following resources:

- Avoid unbounded loops
- Smart contract security best practices
- Smart contract guidelines provided by Quorum
- Guidelines on gas limits and loops provided by Solidity

Quickstart: Create an Azure Blockchain Service blockchain member using the Azure portal

5/11/2021 • 3 minutes to read • Edit Online

In this quickstart, you deploy a new blockchain member and consortium in Azure Blockchain Service using the Azure portal.

> **IMPORTANT**
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

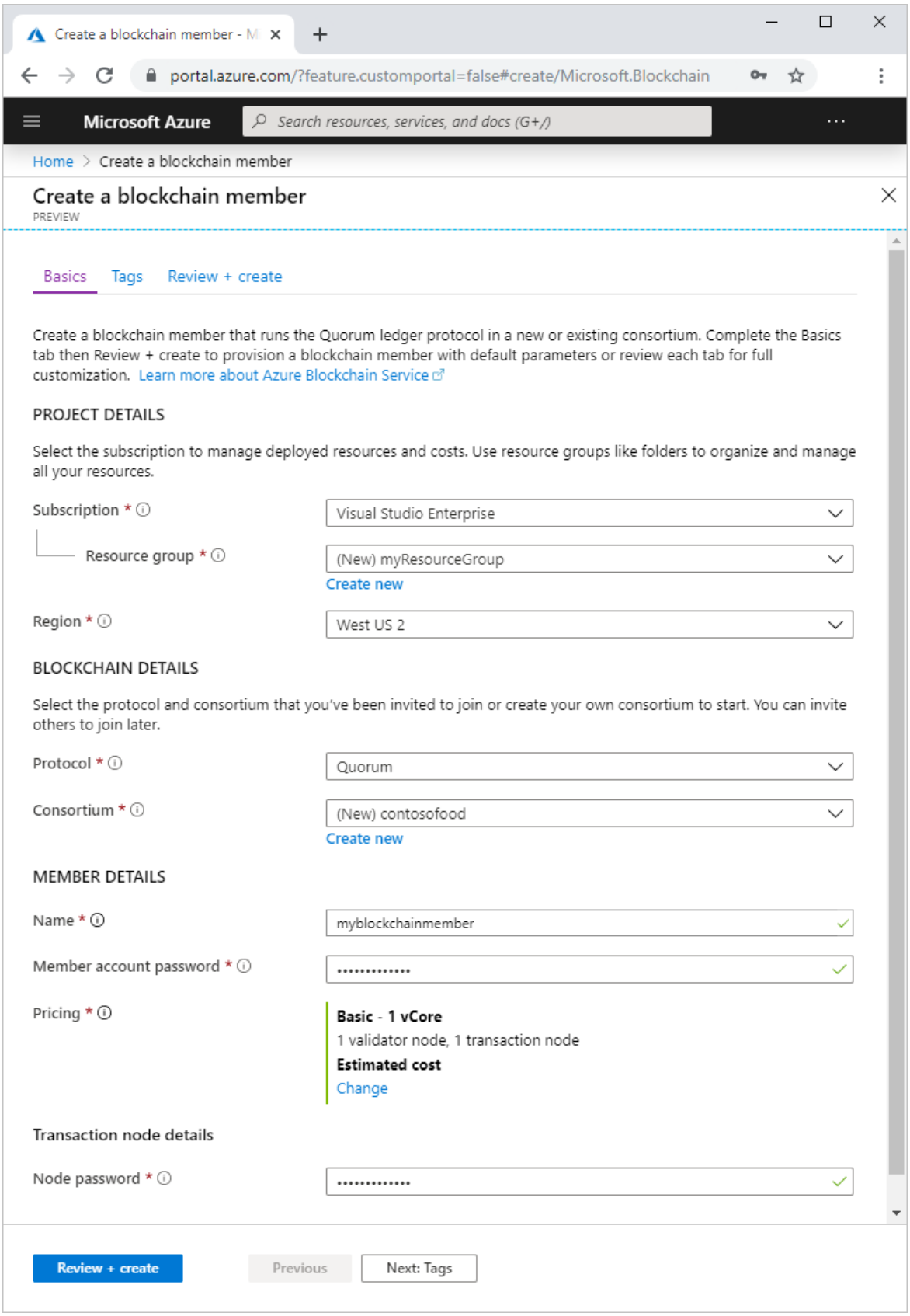
None.

Create a blockchain member

An Azure Blockchain Service member is a blockchain node in a private consortium blockchain network.

When provisioning a member, you can create or join a consortium network. You need at least one member for a consortium network. The number of blockchain members needed by participants depends on your scenario. Consortium participants may have one or more blockchain members or they may share members with other participants. For more information on consortia, see Azure Blockchain Service consortium.

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 (preview)**.

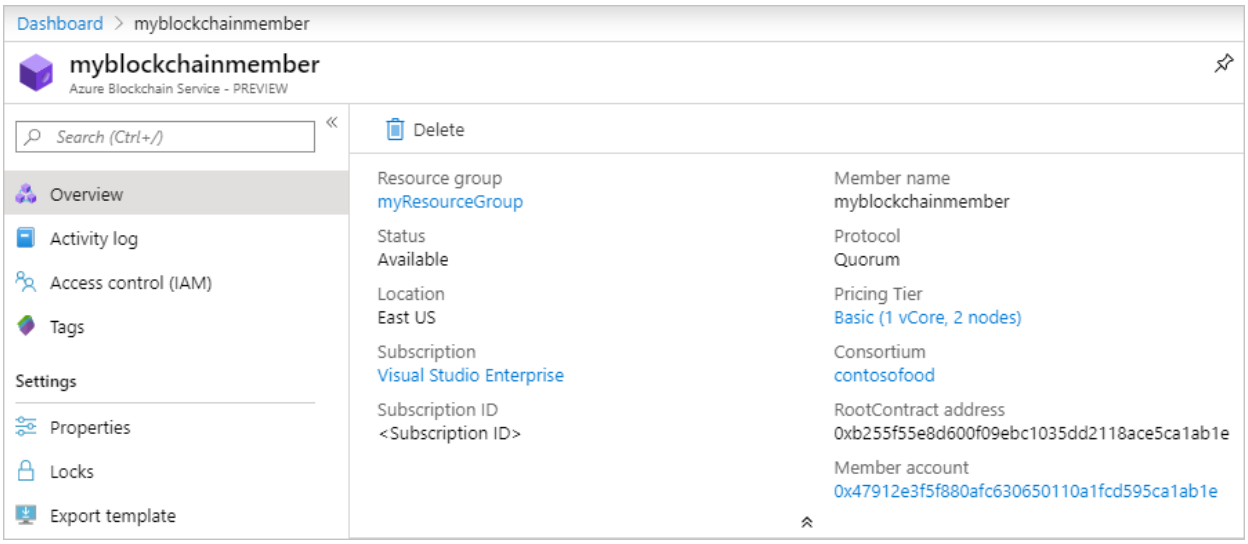


SETTING	DESCRIPTION
Subscription	Select the Azure subscription that you want to use for your service. If you have multiple subscriptions, choose the subscription in which you get billed for the resource.
Resource group	Create a new resource group name or choose an existing one from your subscription.

SETTING	DESCRIPTION
Region	Choose a region to create the member. All members of the consortium must be in the same location. Features may not be available in some regions. Azure Blockchain Data Manager is available in the following Azure regions: East US and West Europe.
Protocol	Currently, Azure Blockchain Service Preview supports the Quorum protocol.
Consortium	For a new consortium, enter a unique name. If joining a consortium through an invite, choose the consortium you are joining. For more information on consortia, see Azure Blockchain Service consortium.
Name	Choose a unique name for the 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.
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.
Pricing	The node configuration and cost for your new service. Select the **Change** link to choose between **Standard** and **Basic** tiers. Use the *Basic* tier for development, testing, and proof of concepts. Use the *Standard* tier for production grade deployments. Also use the *Standard* tier if you are using Blockchain Data Manager or sending a high volume of private transactions. Changing the pricing tier between basic and standard after member creation is not supported.
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.

4. Select **Review + create** to validate your settings. Select **Create** to provision the service. Provisioning takes about 10 minutes.
5. Select **Notifications** on the toolbar to monitor the deployment process.
6. 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.



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 for the quickstart.

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 quickstart, you deployed an Azure Blockchain Service member and a new consortium. Try the next quickstart to use Azure Blockchain Development Kit for Ethereum to attach to an Azure Blockchain Service member.

Use Visual Studio Code to connect to Azure Blockchain Service

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

5/11/2021 • 4 minutes to read • Edit Online

In this quickstart, you deploy a new blockchain member and consortium in Azure Blockchain Service using Azure CLI.

> **IMPORTANT**
>
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

None.

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.

Prepare your environment

1. Sign in.

   Sign in using the az login command if you're using a local install of the CLI.

   ```
   az login
   ```

   Follow the steps displayed in your terminal to complete the authentication process.

2. Install the Azure CLI extension.

   When working with extension references for the Azure CLI, you must first install the extension. Azure CLI extensions give you access to experimental and pre-release commands that have not yet shipped as part of the core CLI. To learn more about extensions including updating and uninstalling, see Use extensions with Azure CLI.

   Install the extension for Azure Blockchain Service by running the following command:

```
az extension add --name blockchain
```

3. Create a resource group.

   Azure Blockchain Service, like all Azure resources, must be deployed into a resource group. Resource groups allow you to organize and manage related Azure resources.

   For this quickstart, create a resource group named *myResourceGroup* in the *eastus* location with the following az group create command:

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

Create a blockchain member

An Azure Blockchain Service member is a blockchain node in a private consortium blockchain network. When provisioning a member, you can create or join a consortium network. You need at least one member for a consortium network. The number of blockchain members needed by participants depends on your scenario. Consortium participants may have one or more blockchain members or they may share members with other participants. For more information on consortia, see Azure Blockchain Service consortium.

There are several parameters and properties you need to pass. Replace the example parameters with your values.

```
az blockchain member create \
                            --resource-group "MyResourceGroup" \
                            --name "myblockchainmember" \
                            --location "eastus" \
                            --password "strongMemberAccountPassword@1" \
                            --protocol "Quorum" \
                            --consortium "myconsortium" \
                            --consortium-management-account-password "strongConsortiumManagementPassword@1"
\
                            --sku "Basic"
```

PARAMETER	DESCRIPTION
resource-group	Resource group name where Azure Blockchain Service resources are created. Use the resource group you created in the previous section.
name	A unique name that identifies your Azure Blockchain Service blockchain member. The name is used for the public endpoint address. For example, `myblockchainmember.blockchain.azure.com`.
location	Azure region where the blockchain member is created. For example, `westus2`. Choose the location that is closest to your users or your other Azure applications. Features may not be available in some regions. Azure Blockchain Data Manager is available in the following Azure regions: East US and West Europe.

PARAMETER	DESCRIPTION
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.
protocol	Blockchain protocol. Currently, *Quorum* protocol is supported.
consortium	Name of the consortium to join or create. For more information on consortia, see Azure Blockchain Service consortium.
consortium-management-account-password	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.
sku	Tier type. *Standard* or *Basic*. Use the *Basic* tier for development, testing, and proof of concepts. Use the *Standard* tier for production grade deployments. Also use the *Standard* tier if you are using Blockchain Data Manager or sending a high volume of private transactions. Changing the pricing tier between basic and standard after member creation is not supported.

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 for the quickstart.

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

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

Next steps

In this quickstart, you deployed an Azure Blockchain Service member and a new consortium. Try the next quickstart to use Azure Blockchain Development Kit for Ethereum to attach to an Azure Blockchain Service member.

Use Visual Studio Code to connect to Azure Blockchain Service

Quickstart: Create an Azure Blockchain Service blockchain member using Azure PowerShell

5/28/2021 • 5 minutes to read • Edit Online

In this quickstart, you deploy a new blockchain member and consortium in Azure Blockchain Service using Azure PowerShell.

> **IMPORTANT**
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

Prerequisites

If you don't have an Azure subscription, create a free account before you begin.

If you choose to use PowerShell locally, this article requires that you install the Az PowerShell module and connect to your Azure account using the Connect-AzAccount cmdlet. For more information about installing the Az PowerShell module, see Install Azure PowerShell.

> **IMPORTANT**
> While the **Az.Blockchain** PowerShell module is in preview, you must install it separately from from the Az PowerShell module using the `Install-Module` cmdlet. Once this PowerShell module becomes generally available, it becomes part of future Az PowerShell module releases and available natively from within Azure Cloud Shell.

```
Install-Module -Name Az.Blockchain
```

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select **Try It** in the upper-right corner of a code block. Selecting **Try It** doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	

OPTION	EXAMPLE/LINK
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code.

Register resource provider

If this is your first time using the Azure Blockchain service, you must register the **Microsoft.Blockchain** resource provider.

```
Register-AzResourceProvider -ProviderNamespace Microsoft.Blockchain
```

Choose a specific Azure subscription

If you have multiple Azure subscriptions, choose the appropriate subscription in which the resources should be billed. Select a specific subscription using the Set-AzContext cmdlet.

```
Set-AzContext -SubscriptionId 00000000-0000-0000-0000-000000000000
```

Define variables

You'll be using several pieces of information repeatedly. Create variables to store the information.

```
# Name of resource group used throughout this article
$resourceGroupName = 'myResourceGroup'

# Azure region
$location = 'eastus'
```

Create a resource group

Create an Azure resource group using the New-AzResourceGroup cmdlet. A resource group is a logical container in which Azure resources are deployed and managed as a group.

The following example creates a resource group based on the name in the `$resourceGroupName` variable in the region specified in the `$location` variable.

```
New-AzResourceGroup -Name $resourceGroupName -Location $location
```

Create a blockchain member

An Azure Blockchain Service member is a blockchain node in a private consortium blockchain network. When provisioning a member, you can create or join a consortium network. You need at least one member for a consortium network. The number of blockchain members needed by participants depends on your scenario. Consortium participants may have one or more blockchain members or they may share members with other participants. For more information on consortia, see Azure Blockchain Service consortium.

There are several parameters and properties you need to pass. Replace the example parameters with your values.

```
$passwd = Read-Host -Prompt 'Enter the members default transaction node password' -AsSecureString
$csPasswd = Read-Host -Prompt 'Enter  the consortium account password' -AsSecureString

$memberParams = @{
  Name = 'myblockchainmember'
  ResourceGroupName = $resourceGroupName
  Consortium = 'myconsortium'
  ConsortiumManagementAccountPassword = $csPasswd
  Location = $location
  Password = $passwd
  Protocol = 'Quorum'
  Sku = 'S0'
}
New-AzBlockchainMember @memberParams
```

PARAMETER	DESCRIPTION
ResourceGroupName	Resource group name where Azure Blockchain Service resources are created. Use the resource group you created in the previous section.
Name	A unique name that identifies your Azure Blockchain Service blockchain member. The name is used for the public endpoint address. For example, `myblockchainmember.blockchain.azure.com`.
Location	Azure region where the blockchain member is created. For example, `westus2`. Choose the location that is closest to your users or your other Azure applications. Features may not be available in some regions. Azure Blockchain Data Manager is available in the following Azure regions: East US and West Europe.
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.
Protocol	Blockchain protocol. Currently, *Quorum* protocol is supported.
Consortium	Name of the consortium to join or create. For more information on consortia, see Azure Blockchain Service consortium.

PARAMETER	DESCRIPTION
ConsortiumManagementAccountPassword	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.
Sku	Tier type. **S0** for standard or **B0** for basic. Use the *Basic* tier for development, testing, and proof of concepts. Use the *Standard* tier for production grade deployments. You should also use the *Standard* tier if you are using Blockchain Data Manager or sending a high volume of private transactions. Changing the pricing tier between basic and standard after member creation is not supported.

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 for the quickstart.

Caution

The following example deletes the specified resource group and all resources contained within it. If resources outside the scope of this article exist in the specified resource group, they will also be deleted.

```
Remove-AzResourceGroup -Name $resourceGroupName
```

Next steps

In this quickstart, you deployed an Azure Blockchain Service member and a new consortium. Try the next quickstart to use Azure Blockchain Development Kit for Ethereum to attach to an Azure Blockchain Service member.

Use Visual Studio Code to connect to Azure Blockchain Service

Quickstart: Create an Azure Blockchain Service member using an ARM template

6/11/2021 • 6 minutes to read • Edit Online

In this quickstart, you deploy a new blockchain member and consortium in Azure Blockchain Service using an Azure Resource Manager template (ARM template).

> **IMPORTANT**
>
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

An Azure Blockchain Service member is a blockchain node in a private consortium blockchain network. When provisioning a member, you can create or join a consortium network. You need at least one member for a consortium network. The number of blockchain members needed by participants depends on your scenario. Consortium participants may have one or more blockchain members or they may share members with other participants. For more information on consortia, see Azure Blockchain Service consortium.

An ARM template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal.

Prerequisites

If you don't have an Azure subscription, create a free account before you begin.

Review the template

The template used in this quickstart is from Azure Quickstart Templates.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "bcMemberName": {
      "type": "String",
      "metadata": {
        "description": "Azure Blockchain Service member name. The blockchain member name must be unique and
can only contain lowercase letters and numbers. The first character must be a letter. The value must be
between 2 and 20 characters long."
      }
    },
    "consortiumName": {
      "type": "String",
      "metadata": {
        "description": "Consortium name. The new consortium name must be unique."
      }
```

```
    },
    "memberPassword": {
      "type": "secureString",
      "metadata": {
        "description": "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. The password
must have 3 of the following: 1 lower case character, 1 upper case character, 1 number, and 1 special
character that is not '#', '`', '*', '\"', ''', '-', '%',' ' or ';'."
      }
    },
    "consortiumManagementAccountPassword": {
      "type": "secureString",
      "metadata": {
        "description": "The consortium management 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. The password must have 3 of the following: 1 lower case character, 1 upper case
character, 1 number, and 1 special character that is not '#', '`', '*', '\"', ''', '-', '%',' ' or ';'."
      }
    },
    "skuTier": {
      "type": "string",
      "defaultValue": "Basic",
      "metadata": {
        "description": "Use Basic or Standard. Use the Basic tier for development, testing, and proof of
concepts. Use the Standard tier for production grade deployments. You should also use the Standard tier if
you are using Blockchain Data Manager or sending a high volume of private transactions. Changing the pricing
tier between basic and standard after member creation is not supported."
      }
    },
    "skuName": {
      "type": "string",
      "defaultValue": "B0",
      "metadata": {
        "description": "Use S0 for Standard and B0 for Basic."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for all resources."
      }
    }
  },
  "variables": {},
  "resources": [
    {
      "type": "Microsoft.Blockchain/blockchainMembers",
      "apiVersion": "2018-06-01-preview",
      "name": "[parameters('bcMemberName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "[parameters('skuName')]",
        "tier": "[parameters('skuTier')]"
      },
      "tags": {
        "consortium": "Consortium"
      },
      "properties": {
        "protocol": "Quorum",
        "consensus": "Default",
        "password": "[parameters('memberPassword')]",
        "validatorNodesSku": {
          "capacity": 1
        },
        "consortium": "[parameters('consortiumName')]",
        "consortiumManagementAccountPassword": "[parameters('consortiumManagementAccountPassword')]",
        "firewallRules": [
          {
```

```
            {
              "ruleName": "AllowAll",
              "startIpAddress": "0.0.0.0",
              "endIpAddress": "255.255.255.255"
            }
          ]
        }
      }
    ]
}
```

Azure resources defined in the template:

- Microsoft.Blockchain/blockchainMembers

Deploy the template

1. Select the following link to sign in to Azure and open a template.

   Deploy to Azure

2. Specify the settings for the Azure Blockchain Service member.

SETTING	DESCRIPTION
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.
Resource group	Create a new resource group name or choose an existing one from your subscription.
Region	Choose a region to create the resource group. All members of the consortium must be in the same location. Available locations for the deployment are *westeurope, eastus, southeastasia, westeurope, northeurope, westus2*, and *japaneast*. Features may not be available in some regions. Azure Blockchain Data Manager is available in the following Azure regions: East US and West Europe.
Bc Member name	Choose a unique name for the 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.
Consortium name	Enter a unique name. For more information on consortia, see Azure Blockchain Service consortium.
Member 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.
Consortium Management Account Password	The consortium account password is used to encrypt the private key for the Ethereum account that is created for your member. It is used for consortium management.

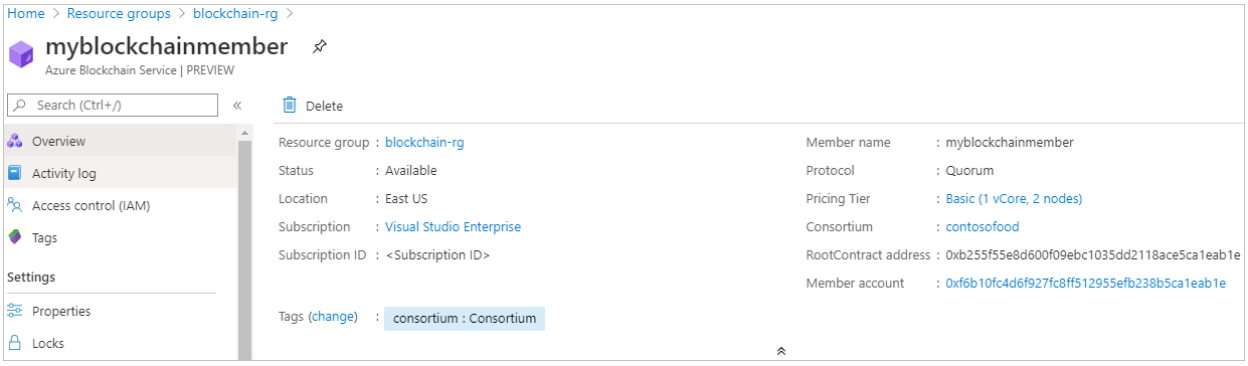
SETTING	DESCRIPTION
Sku tier	The pricing tier for your new service. Choose between **Standard** and **Basic** tiers. Use the *Basic* tier for development, testing, and proof of concepts. Use the *Standard* tier for production grade deployments. Also use the *Standard* tier if you are using Blockchain Data Manager or sending a high volume of private transactions. Changing the pricing tier between basic and standard after member creation is not supported.
Sku name	The node configuration and cost for your new service. Use **B0** for Basic and **S0** for Standard.
Location	Choose a location to create the member. By default, the resource group location is used `[resourceGroup().location]`. All members of the consortium must be in the same location. Available locations for the deployment are *westeurope, eastus, southeastasia, westeurope, northeurope, westus2*, and *japaneast*. Features may not be available in some regions. Azure Blockchain Data Manager is available in the following Azure regions: East US and West Europe.

3. Select **Review + Create** to verify and deploy the template.

The Azure portal is used here to deploy the template. You can also use the Azure PowerShell, Azure CLI, and REST API. To learn other deployment methods, see Deploy templates.

Review deployed resources

You can use the Azure portal to view details of the deployed Azure Blockchain Service member. In the portal, go to the resource group containing your Azure Blockchain Service member. Select the blockchain member you created.



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 resource group you created for the quickstart.

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 quickstart, you deployed an Azure Blockchain Service member and a new consortium. Try the next quickstart to use Azure Blockchain Development Kit for Ethereum to attach to an Azure Blockchain Service member.

Use Visual Studio Code to connect to Azure Blockchain Service

Quickstart: Use Visual Studio Code to connect to an Azure Blockchain Service consortium network

5/11/2021 • 2 minutes to read • Edit Online

In this quickstart, you install and use the Azure Blockchain Development Kit for Ethereum Visual Studio Code (VS Code) extension to attach to a consortium on Azure Blockchain Service. The Azure Blockchain Development Kit simplifies how you create, connect, build, and deploy smart contracts on Ethereum blockchain ledgers.

> **IMPORTANT**
>
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- Complete Quickstart: Create a blockchain member using the Azure portal or Quickstart: Create an Azure Blockchain Service blockchain member using Azure CLI
- Visual Studio Code
- Azure Blockchain Development Kit for Ethereum extension
- Node.js 10.15.x or higher
- Git 2.10.x or higher
- Truffle 5.0.0
- Ganache CLI 6.0.0

On Windows, an installed C++ compiler is required for the node-gyp module. You can use the MSBuild tools:

- If Visual Studio 2017 is installed, configure npm to use the MSBuild tools with the command `npm config set msvs_version 2017 -g`
- If Visual Studio 2019 is installed, set the MS build tools path for npm. For example,
  ```
  npm config set msbuild_path "C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\MSBuild\Current\Bin\MSBuild.exe"
  ```
- Otherwise, install the stand-alone VS Build tools using `npm install --global windows-build-tools` in an elevated *Run as administrator* command shell.

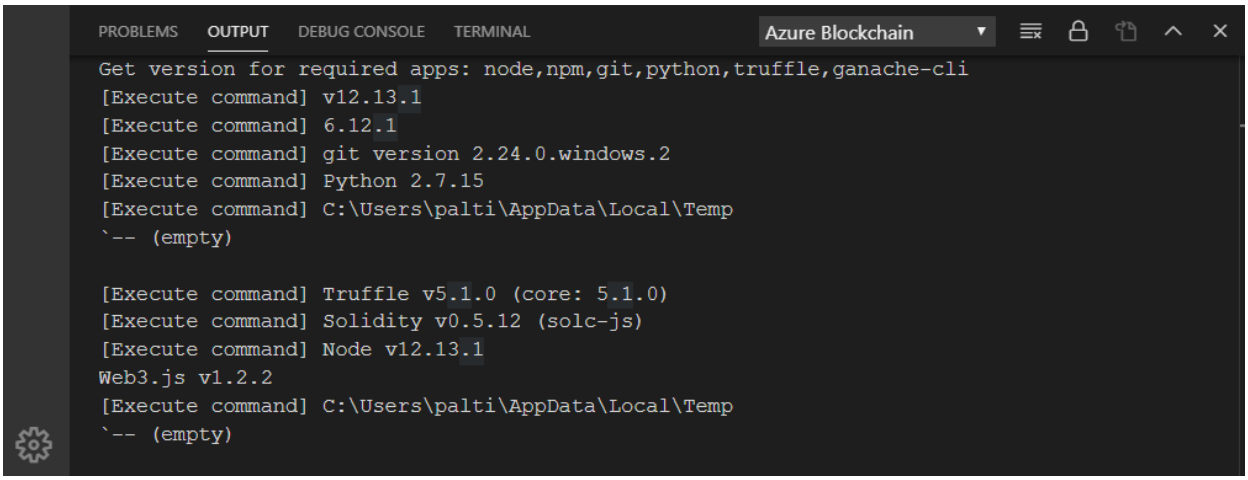
For more information about node-gyp, see the node-gyp repository on GitHub.

Verify Azure Blockchain Development Kit environment

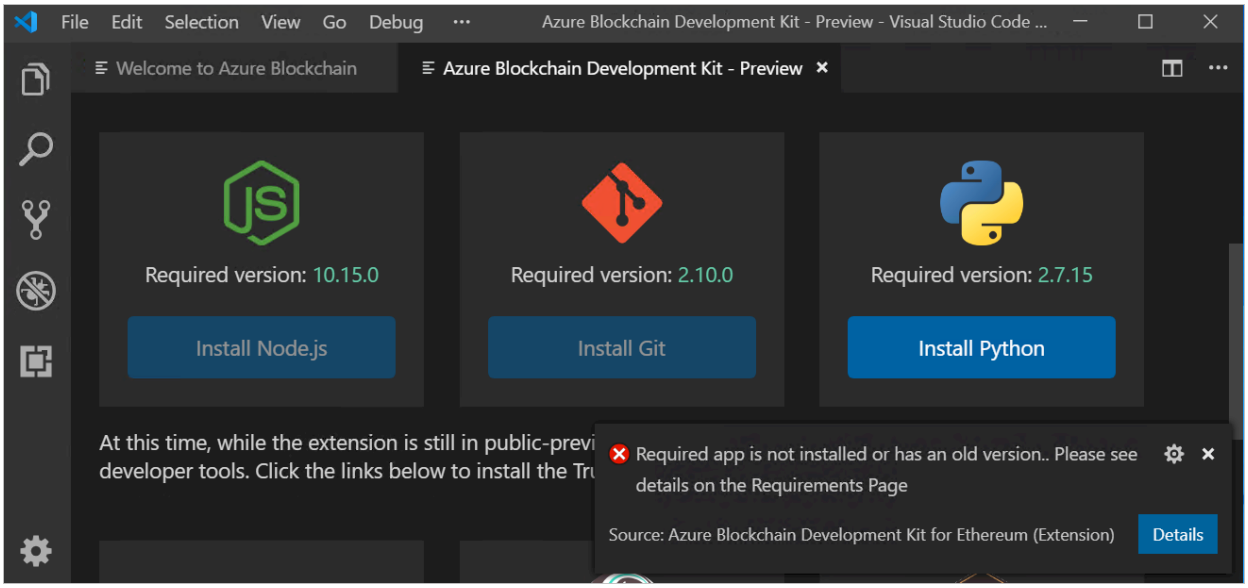
Azure Blockchain Development Kit verifies your development environment prerequisites have been met. To verify your development environment:

From the VS Code command palette, choose **Blockchain: Show Welcome Page**.

Azure Blockchain Development Kit runs a validation script that takes about a minute to complete. You can view the output by selecting **Terminal > New Terminal**. In the terminal menu bar, select the **Output** tab and **Azure Blockchain** in the dropdown. Successful validation looks like the following image:



If you are missing a required tool, a new tab named **Azure Blockchain Development Kit - Preview** lists the required tools with download links.



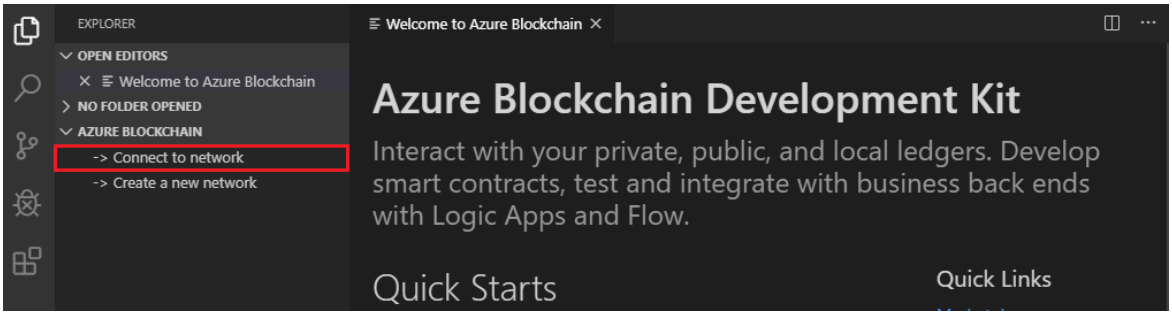
Install any missing prerequisites before continuing with the quickstart.

Connect to consortium member

You can connect to consortium members using the Azure Blockchain Development Kit VS Code extension. Once connected to a consortium, you can compile, build, and deploy smart contracts to an Azure Blockchain Service consortium member.

If you don't have access to an Azure Blockchain Service consortium member, complete the prerequisite Quickstart: Create a blockchain member using the Azure portal or Quickstart: Create an Azure Blockchain Service blockchain member using Azure CLI.

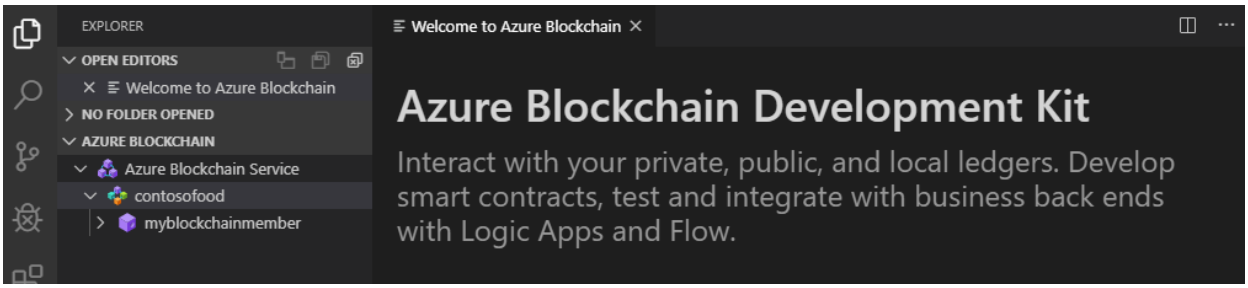
1. In the VS Code explorer pane, expand the **Azure Blockchain** extension.
2. Select **Connect to network**.



If prompted for Azure authentication, follow the prompts to authenticate using a browser.

3. Choose **Azure Blockchain Service** in the command palette dropdown.
4. Choose the subscription and resource group associated with your Azure Blockchain Service consortium member.
5. Choose your consortium from the list.

The consortium and blockchain members are listed in the VS Code explorer side bar.



Next steps

In this quickstart, you used Azure Blockchain Development Kit for Ethereum VS Code extension to attach to a consortium on Azure Blockchain Service. Try the next tutorial to use Azure Blockchain Development Kit for Ethereum to create, build, deploy, and execute a smart contract function via a transaction.

Create, build, and deploy smart contracts on Azure Blockchain Service

Quickstart: Use MetaMask to connect and deploy a smart contract

5/11/2021 • 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.

> **IMPORTANT**
>
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

If you don't have an Azure subscription, create a free account before you begin.

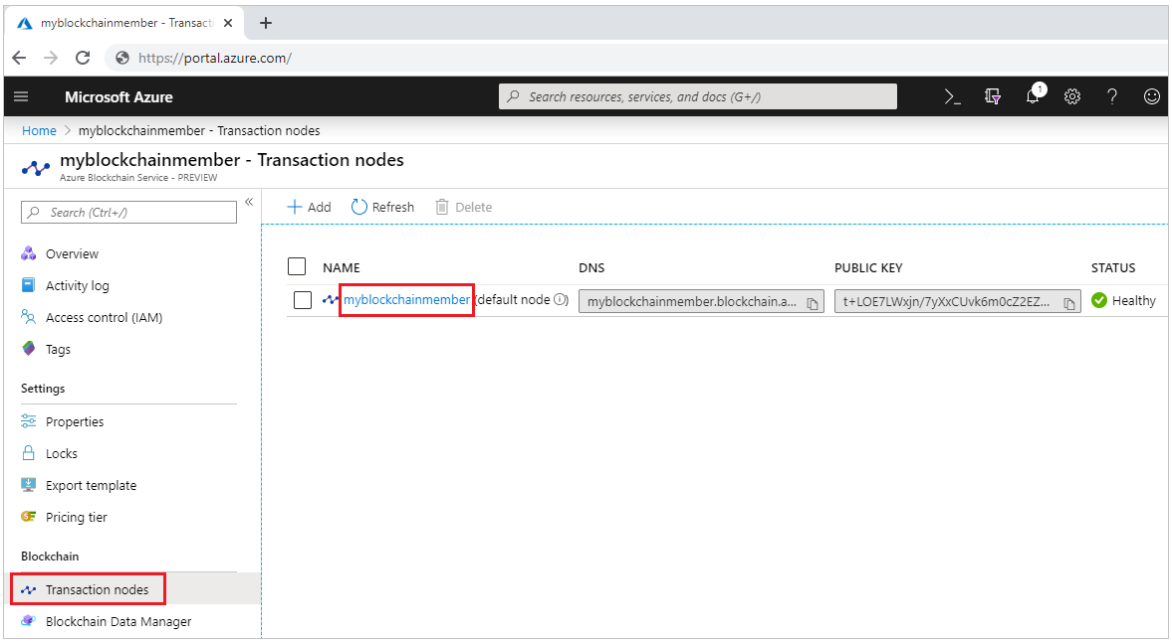
Prerequisites

- Complete Quickstart: Create a blockchain member using the Azure portal or Quickstart: Create an Azure Blockchain Service blockchain member using Azure CLI
- Install MetaMask browser extension
- Generate a MetaMask wallet

Get endpoint address

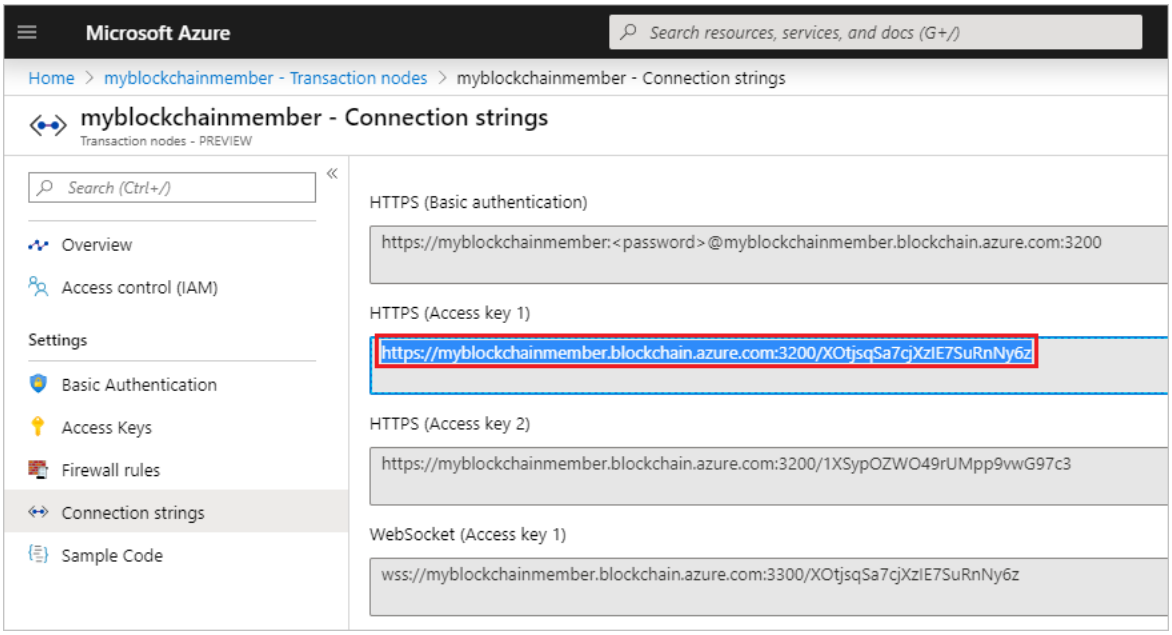
You need the Azure Blockchain Service endpoint address to connect to the blockchain network. The endpoint address and access keys are in the Azure portal.

1. Sign in to the Azure portal.
2. Navigate to your Azure Blockchain Service member.
3. Select **Transaction nodes** and the default transaction node link.



4. Select **Connection strings > Access keys**.

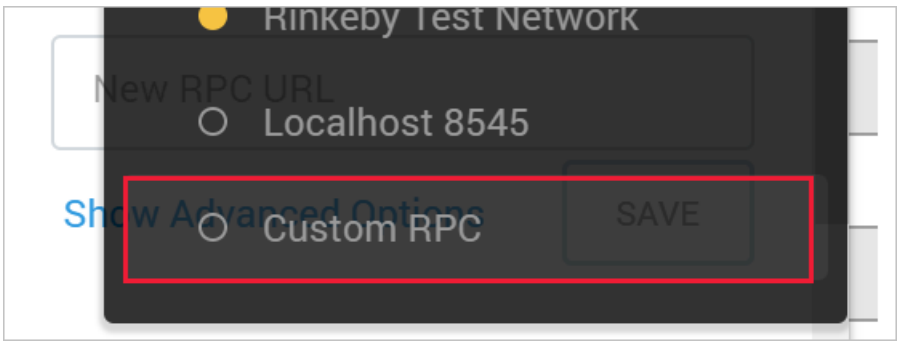
5. Copy the endpoint address from **HTTPS (Access key 1)**.



Connect MetaMask

1. Open MetaMask browser extension and sign in.

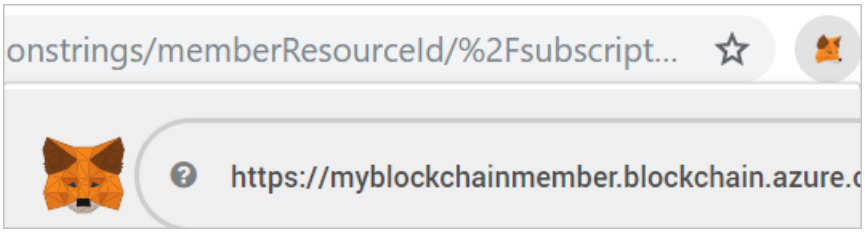
2. In the network dropdown, select **Custom RPC**.



3. In **New Network > New RPC URL**, paste the endpoint address you copied above.

4. Select **Save**.

   If connection was successful, the private network displays in the network drop-down.



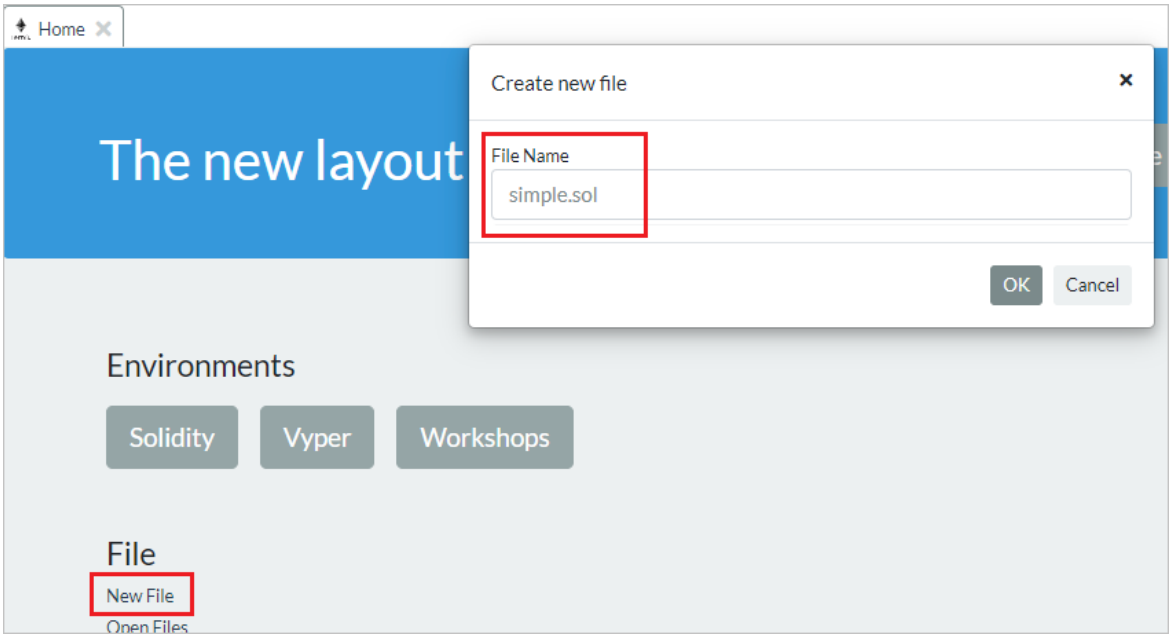
Deploy smart contract

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 **New file** in the **Home** tab under **File**.

   Name the new file `simple.sol`.



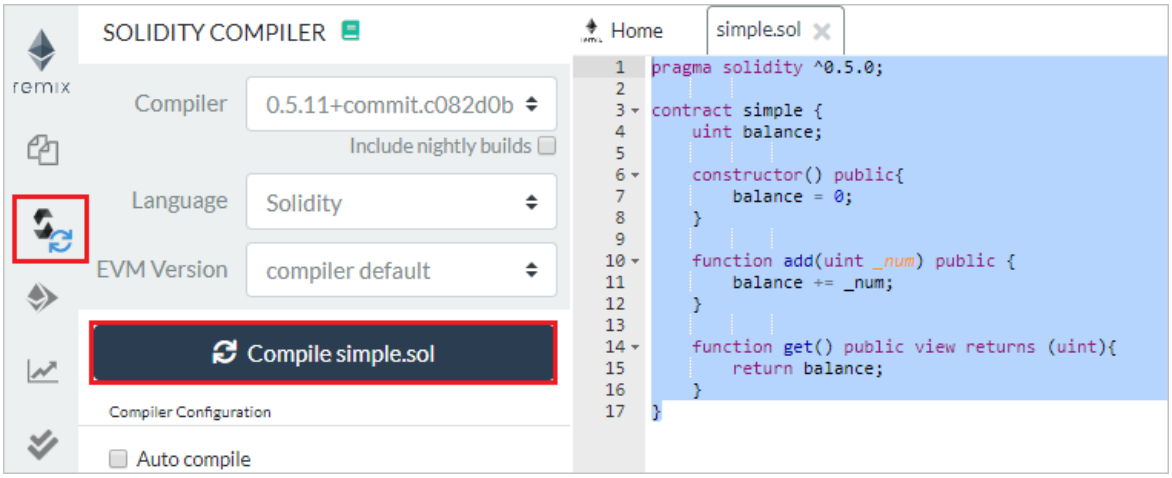
   Select **OK**.

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

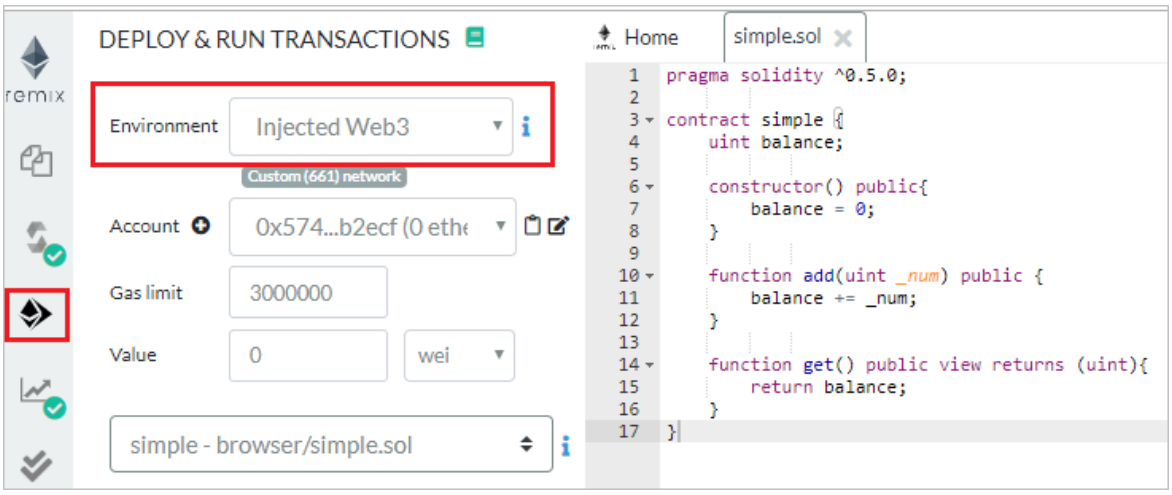
```
pragma solidity ^0.5.0;

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**.

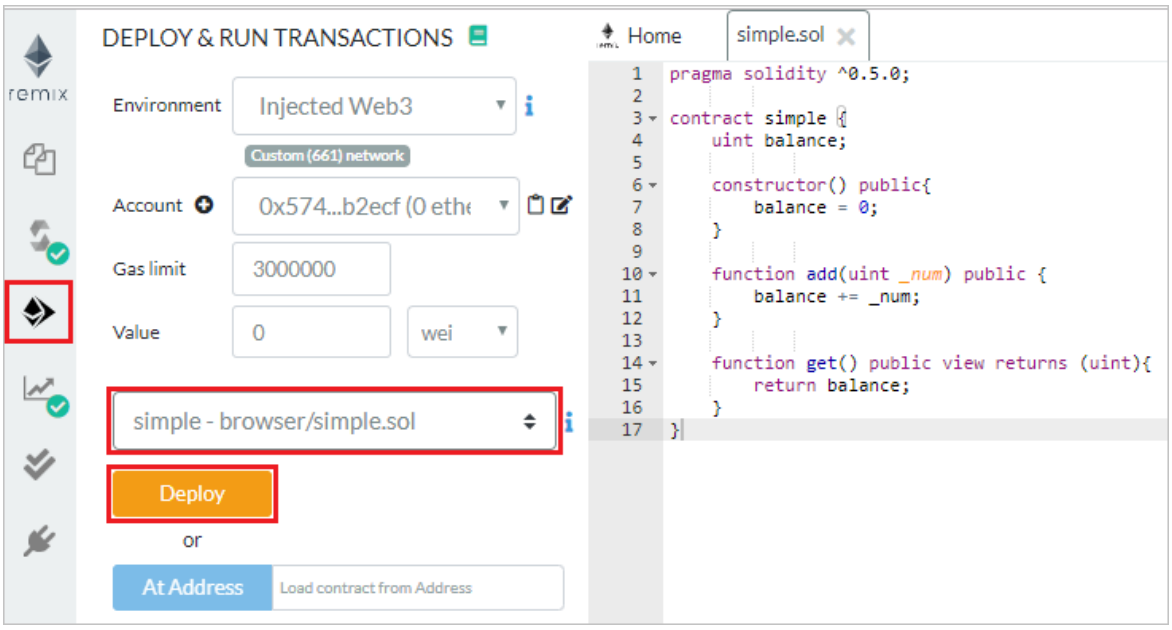
4. To compile the contract, first select the Solidity compiler pane then select the **Compile simple.sol**.



5. Select the **Deploy & Run** pane then set the **Environment** to **Injected Web3** to connect through MetaMask to your blockchain member.



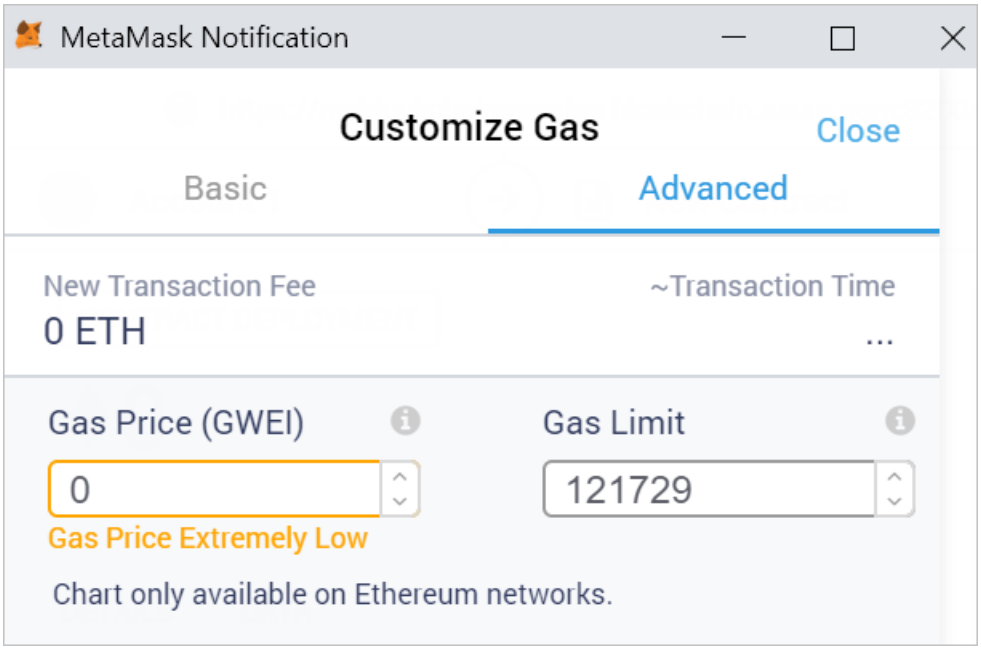
6. Select the **simple** contract, then **Deploy**.



7. A MetaMask notification alerts 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.

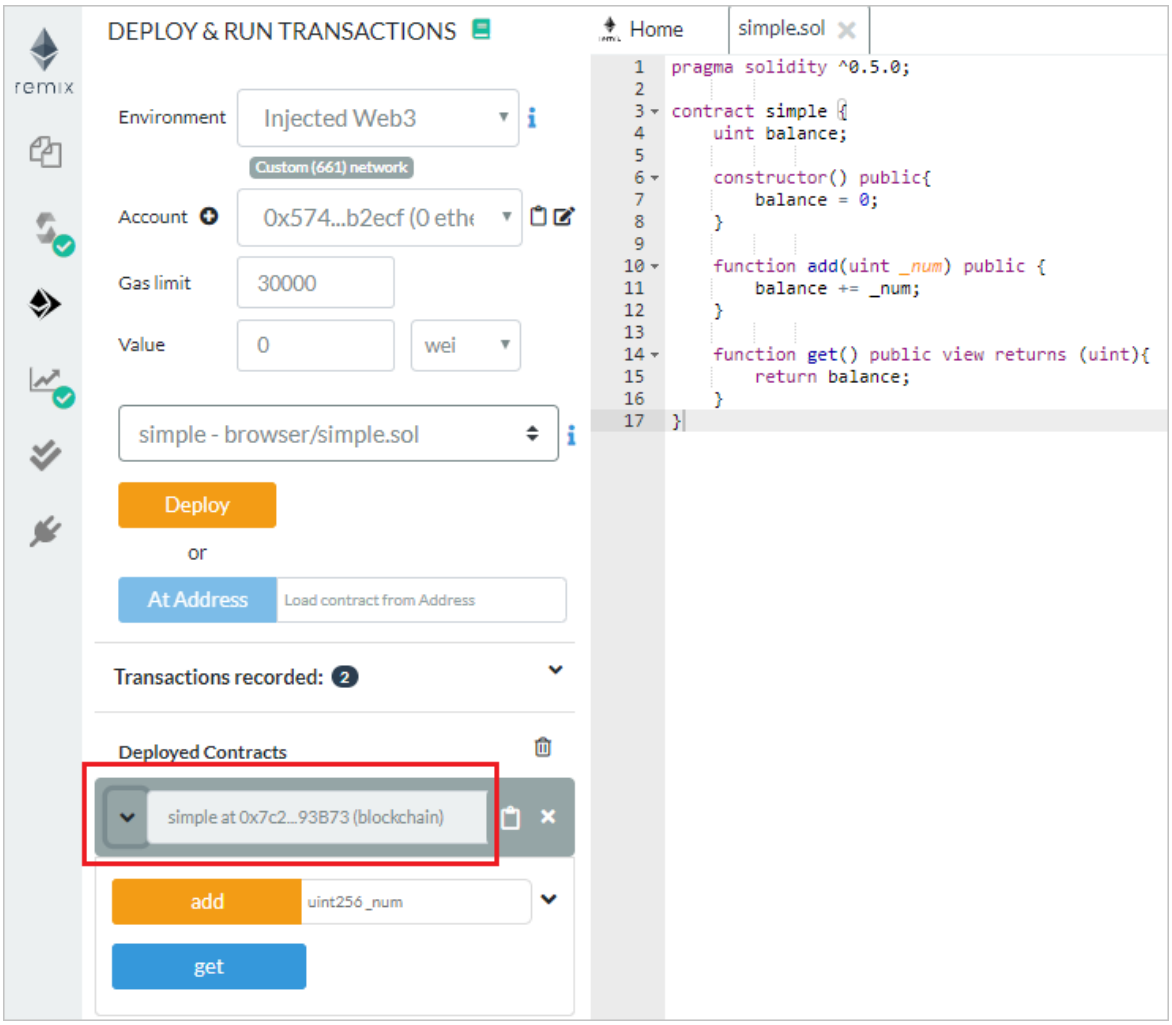
8. Select **Gas Fee** > **Edit** > **Advanced**, set the **Gas Price** to 0.



   Select **Save**.

9. Select **Confirm** to deploy the smart contract to the blockchain.

10. In the **Deployed Contracts** section, expand the **simple** contract.



Two actions, **add** and **get**, 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**. You may get a gas estimation failure message from Remix: "You are sending the transaction to a private blockchain that does not require gas." Select **Send Transaction** to force the transaction.

12. Similar to when you deployed the contract, a MetaMask notification alerts you of insufficient funds to perform the transaction.

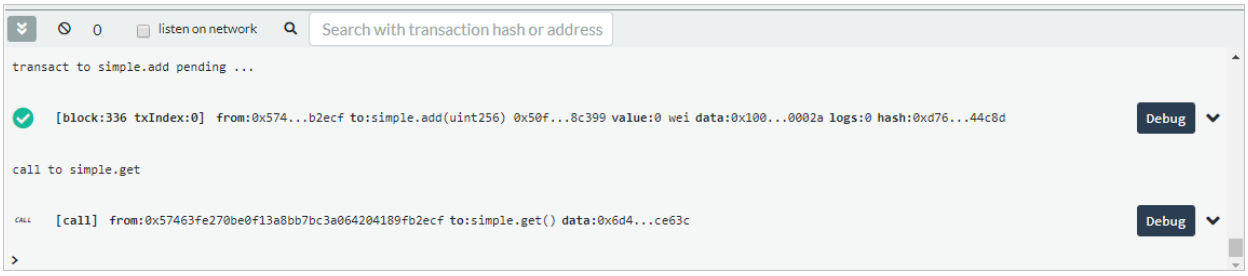
    Since this is a private network in a consortium, we can set gas price to zero.

13. Select **Gas Fee > Edit > Advanced**, set the **Gas Price** to 0, and select **Save**.

14. Select **Confirm** to send the transaction to the blockchain.

15. Select **get** action. This is a call to query node data. A transaction isn't needed.

The debug pane of Remix shows details about the transactions on the blockchain:



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

To see transaction history in MetaMask, open the MetaMask browser extension and look in the **History** section

for 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 use Azure Blockchain Development Kit for Ethereum and Truffle to create, build, deploy, and execute a smart contract function via a transaction.

Create, build, and deploy smart contracts on Azure Blockchain Service

Quickstart: Use Geth to attach to an Azure Blockchain Service transaction node

5/11/2021 • 2 minutes to read • Edit Online

In this quickstart, you use the Geth client to attach to a Geth instance on an Azure Blockchain Service transaction node. Once attached, you use the Geth console to call an Ethereum JavaScript API.

> **IMPORTANT**
>
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

If you don't have an Azure subscription, create a free account before you begin.

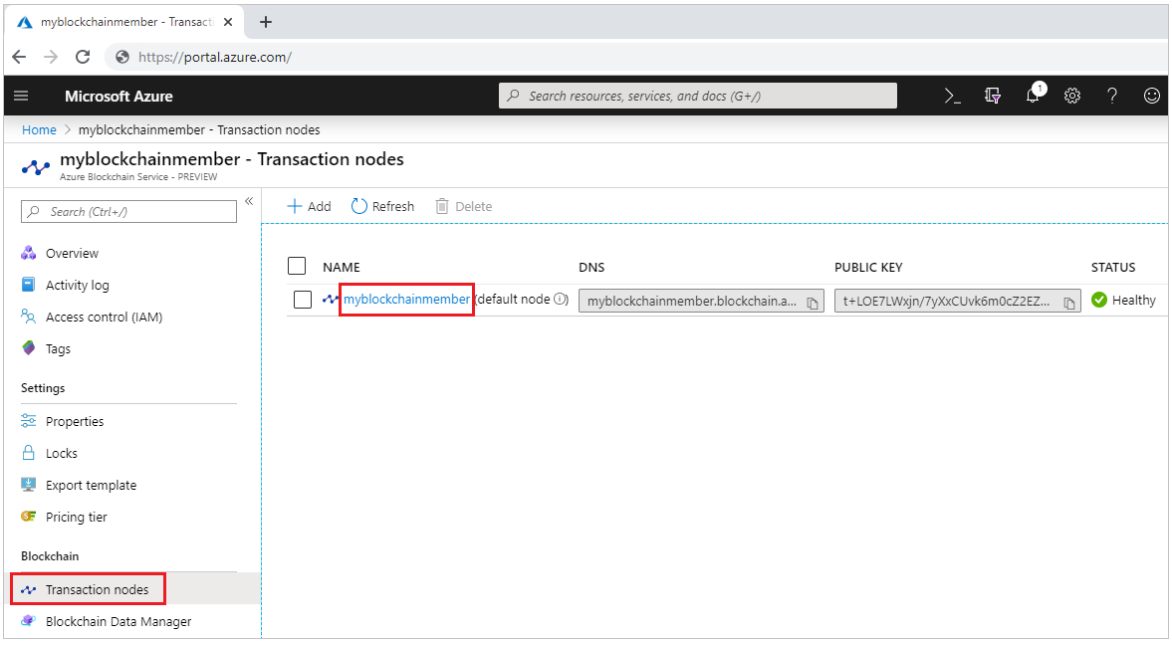
Prerequisites

- Install Geth
- Complete Quickstart: Create a blockchain member using the Azure portal or Quickstart: Create an Azure Blockchain Service blockchain member using Azure CLI

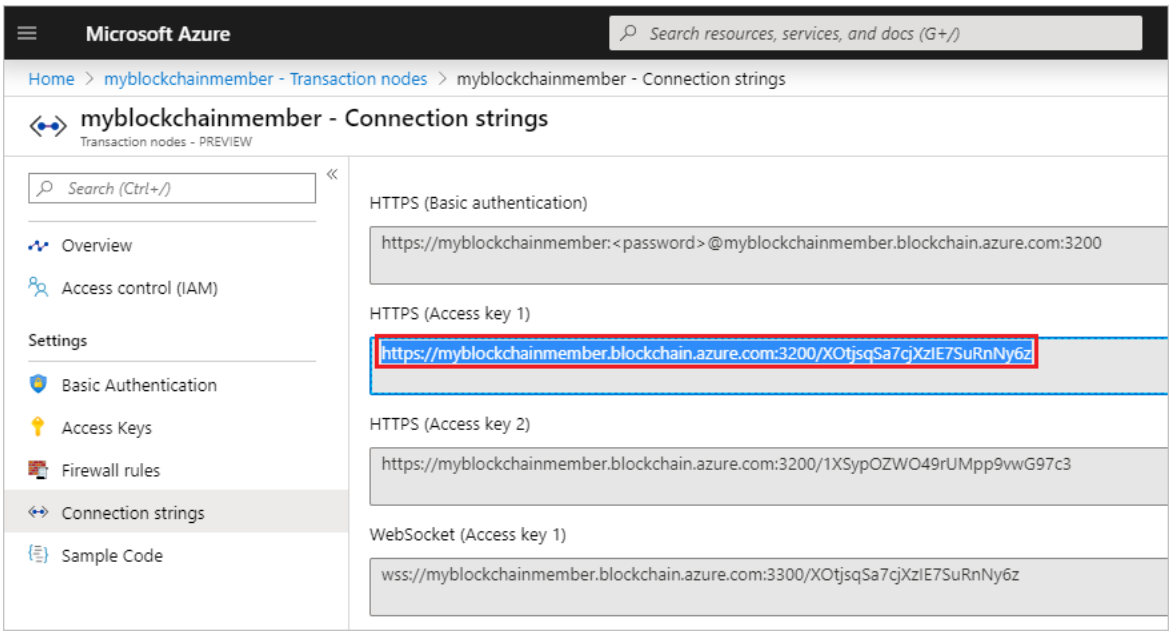
Get Geth connection string

You can get the Geth connection string for an Azure Blockchain Service transaction node in the Azure portal.

1. Sign in to the Azure portal.
2. Go 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 string 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 use the Ethereum JavaScript API.

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

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

   In this example, the chainId is 661.

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 use Azure Blockchain Development Kit for Ethereum to create, build, deploy, and execute a smart contract function via a transaction.

Create, build, and deploy smart contracts on Azure Blockchain Service

Tutorial: Create, build, and deploy smart contracts on Azure Blockchain Service

5/11/2021 • 6 minutes to read • Edit Online

In this tutorial, use the Azure Blockchain Development Kit for Ethereum extension in Visual Studio Code to create, build, and deploy a smart contract on Azure Blockchain Service. You also use the development kit to execute a smart contract function via a transaction.

> **IMPORTANT**
>
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

You use Azure Blockchain Development Kit for Ethereum to:

- Create a smart contract
- Deploy a smart contract
- Execute a smart contract function via a transaction

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

- Complete Quickstart: Use Visual Studio Code to connect to a Azure Blockchain Service consortium network
- Visual Studio Code
- Azure Blockchain Development Kit for Ethereum extension
- Node.js 10.15.x or higher
- Git 2.10.x or higher
- Truffle 5.0.0
- Ganache CLI 6.0.0

On Windows, an installed C++ compiler is required for the node-gyp module. You can use the MSBuild tools:

- If Visual Studio 2017 is installed, configure npm to use the MSBuild tools with the command `npm config set msvs_version 2017 -g`
- If Visual Studio 2019 is installed, set the MS build tools path for npm. For example,
  ```
  npm config set msbuild_path "C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\MSBuild\Current\Bin\MSBuild.exe"
  ```
- Otherwise, install the stand-alone VS Build tools using `npm install --global windows-build-tools` in an elevated *Run as administrator* command shell.

For more information about node-gyp, see the node-gyp repository on GitHub.

Create a smart contract

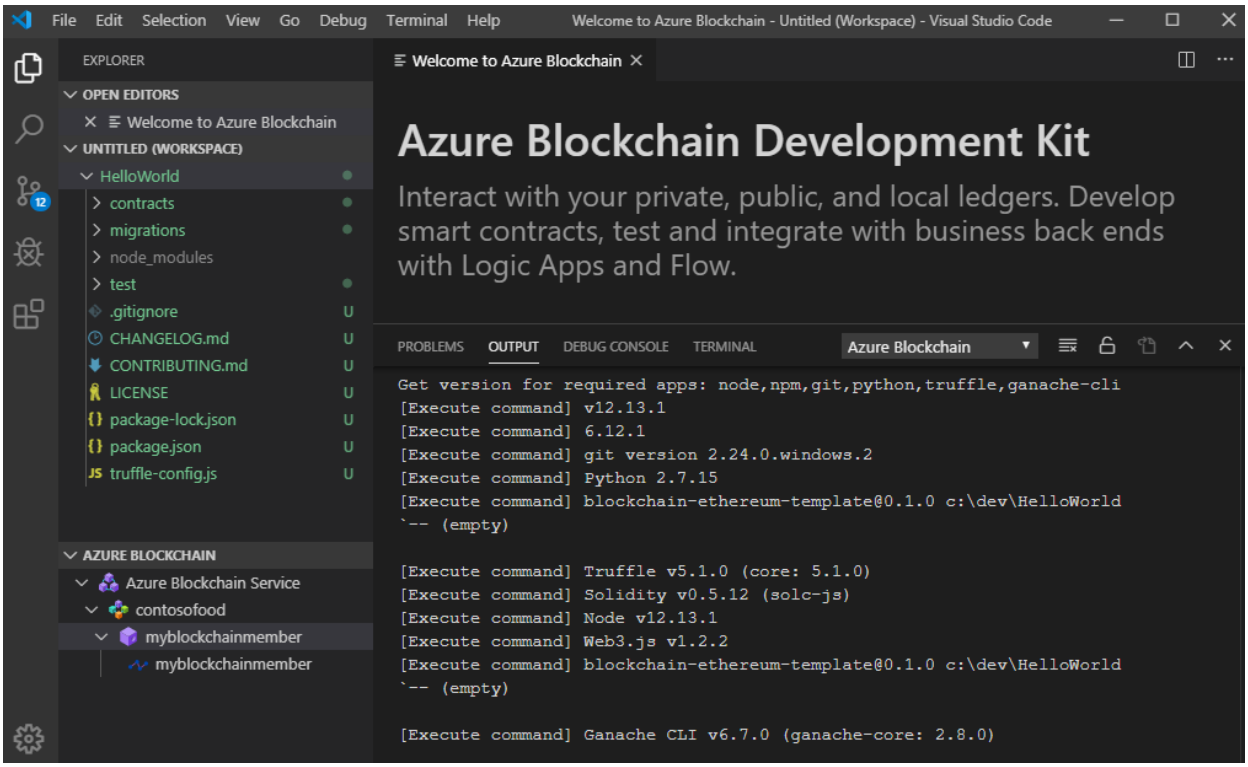
The Azure Blockchain Development Kit for Ethereum uses project templates and Truffle tools to help scaffold, build, and deploy contracts. Before you begin, complete the prerequisite Quickstart: Use Visual Studio Code to connect to a Azure Blockchain Service consortium network. The quickstart guides you through the installation

and configuration of the Azure Blockchain Development Kit for Ethereum.

1. From the VS Code command palette, choose **Blockchain: New Solidity Project**.
2. Choose **Create basic project**.
3. Create a new folder named `HelloBlockchain` and **Select new project path**.

The Azure Blockchain Development Kit creates and initializes a new Solidity project for you. The basic project includes a sample **HelloBlockchain** smart contract and all the necessary files to build and deploy to your consortium member in Azure Blockchain Service. It may take several minutes for the project to be created. You can monitor the progress in VS Code's terminal panel by selecting the output for Azure Blockchain.

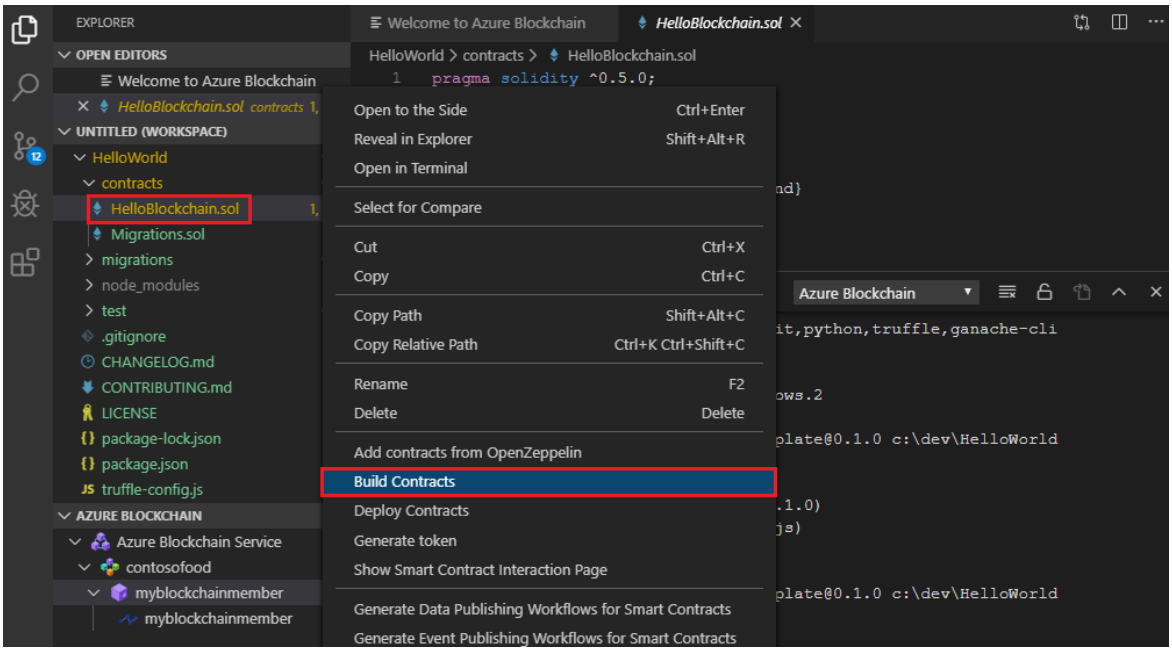
The project structure looks like the following example:



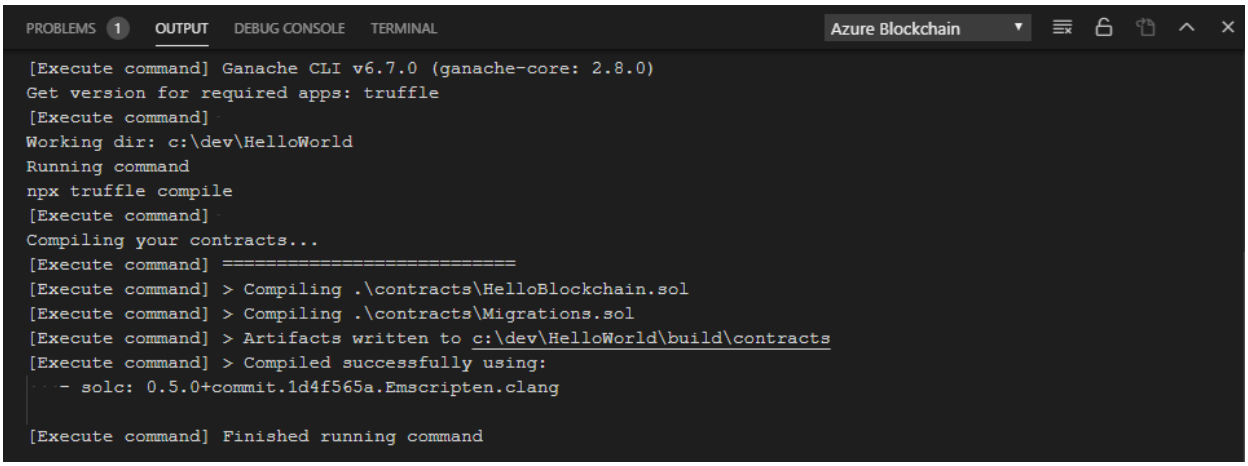
Build a smart contract

Smart contracts are located in the project's **contracts** directory. You compile smart contracts before you deploy them to a blockchain. Use the **Build Contracts** command to compile all the smart contracts in your project.

1. In the VS Code explorer sidebar, expand the **contracts** folder in your project.
2. Right-click **HelloBlockchain.sol** and choose **Build Contracts** from the menu.



Azure Blockchain Development Kit uses Truffle to compile the smart contracts.

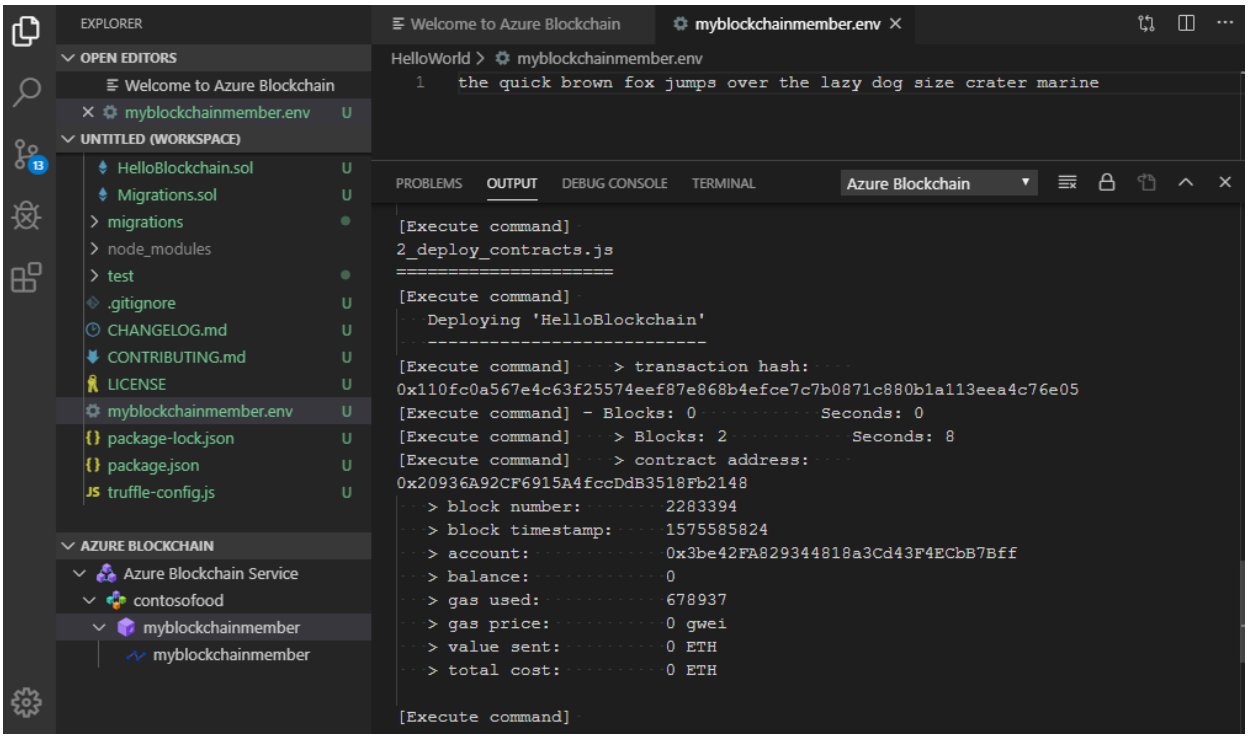


Deploy a smart contract

Truffle uses migration scripts to deploy your contracts to an Ethereum network. Migrations are JavaScript files located in the project's **migrations** directory.

1. To deploy your smart contract, right-click **HelloBlockchain.sol** and choose **Deploy Contracts** from the menu.
2. Choose your Azure Blockchain consortium network in the command palette. The consortium blockchain network was added to the project's Truffle configuration file when you created the project.
3. Choose **Generate mnemonic**. Choose a filename and save the mnemonic file in the project folder. For example, `myblockchainmember.env`. The mnemonic file is used to generate an Ethereum private key for your blockchain member.

Azure Blockchain Development Kit uses Truffle to execute the migration script to deploy the contracts to the blockchain.



Call a contract function

The **HelloBlockchain** contract's **SendRequest** function changes the **RequestMessage** state variable. Changing the state of a blockchain network is done via a transaction. You can create a script to execute the **SendRequest** function via a transaction.

1. Create a new file in the root of your Truffle project and name it `sendrequest.js`. Add the following Web3 JavaScript code to the file.

```
var HelloBlockchain = artifacts.require("HelloBlockchain");

module.exports = function(done) {
  console.log("Getting the deployed version of the HelloBlockchain smart contract")
  HelloBlockchain.deployed().then(function(instance) {
    console.log("Calling SendRequest function for contract ", instance.address);
    return instance.SendRequest("Hello, blockchain!");
  }).then(function(result) {
    console.log("Transaction hash: ", result.tx);
    console.log("Request complete");
    done();
  }).catch(function(e) {
    console.log(e);
    done();
  });
};
```

2. When Azure Blockchain Development Kit creates a project, the Truffle configuration file is generated with your consortium blockchain network endpoint details. Open **truffle-config.js** in your project. The configuration file lists two networks: one named development and one with the same name as the consortium.

3. In VS Code's terminal pane, use Truffle to execute the script on your consortium blockchain network. In the terminal pane menu bar, select the **Terminal** tab and **PowerShell** in the dropdown.

```
truffle exec sendrequest.js --network <blockchain network>
```

Replace <blockchain network> with the name of the blockchain network defined in the **truffle-config.js**.

Truffle executes the script on your blockchain network.

When you execute a contract's function via a transaction, the transaction isn't processed until a block is created. Functions meant to be executed via a transaction return a transaction ID instead of a return value.

Query contract state

Smart contract functions can return the current value of state variables. Let's add a function to return the value of a state variable.

1. In **HelloBlockchain.sol**, add a **getMessage** function to the **HelloBlockchain** smart contract.

   ```
   function getMessage() public view returns (string memory)
   {
       if (State == StateType.Request)
           return RequestMessage;
       else
           return ResponseMessage;
   }
   ```

   The function returns the message stored in a state variable based on the current state of the contract.

2. Right-click **HelloBlockchain.sol** and choose **Build Contracts** from the menu to compile the changes to the smart contract.

3. To deploy, right-click **HelloBlockchain.sol** and choose **Deploy Contracts** from the menu. When prompted, choose your Azure Blockchain consortium network in the command palette.

4. Next, create a script using to call the **getMessage** function. Create a new file in the root of your Truffle project and name it `getmessage.js`. Add the following Web3 JavaScript code to the file.

   ```
   var HelloBlockchain = artifacts.require("HelloBlockchain");

   module.exports = function(done) {
     console.log("Getting the deployed version of the HelloBlockchain smart contract")
     HelloBlockchain.deployed().then(function(instance) {
       console.log("Calling getMessage function for contract ", instance.address);
       return instance.getMessage();
     }).then(function(result) {
       console.log("Request message value: ", result);
       console.log("Request complete");
       done();
     }).catch(function(e) {
       console.log(e);
       done();
     });
   };
   ```

5. In VS Code's terminal pane, use Truffle to execute the script on your blockchain network. In the terminal pane menu bar, select the **Terminal** tab and **PowerShell** in the dropdown.

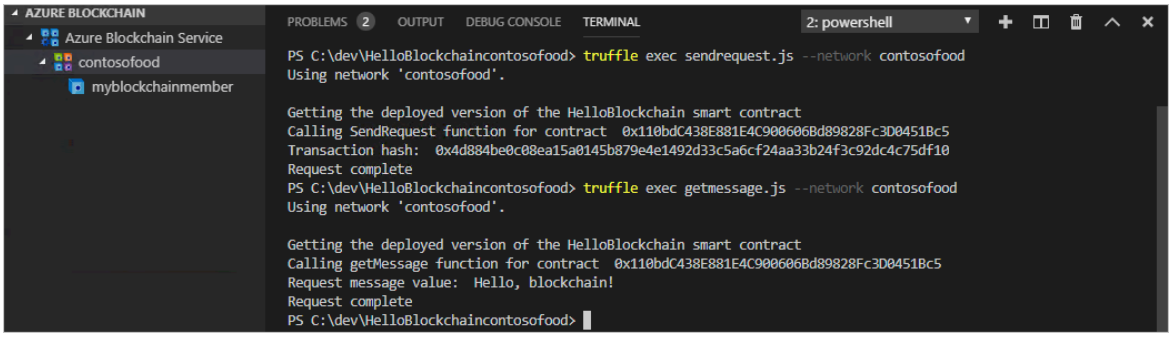
```
truffle exec getmessage.js --network <blockchain network>
```

Replace <blockchain network> with the name of the blockchain network defined in the **truffle-config.js**.

The script queries the smart contract by calling the getMessage function. The current value of the **RequestMessage** state variable is returned.

Notice the value is not **Hello, blockchain!**. Instead, the returned value is a placeholder. When you change and deploy the contract, the changed contract is deployed at a new address and the state variables are assigned values in the smart contract constructor. The Truffle sample **2_deploy_contracts.js** migration script deploys the smart contract and passes a placeholder value as an argument. The constructor sets the **RequestMessage** state variable to the placeholder value and that's what is returned.

1. To set the **RequestMessage** state variable and query the value, run the **sendrequest.js** and **getmessage.js** scripts again.



sendrequest.js sets the **RequestMessage** state variable to **Hello, blockchain!** and **getmessage.js** queries the contract for value of **RequestMessage** state variable and returns **Hello, blockchain!**.

Clean up resources

When no longer needed, you can delete the resources by deleting the `myResourceGroup` resource group you created in the *Create a blockchain member* prerequisite quickstart.

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 created a sample Solidity project using Azure Blockchain Development Kit. You built and deployed a smart contract then called a function via a transaction on a blockchain consortium network hosted on Azure Blockchain Service.

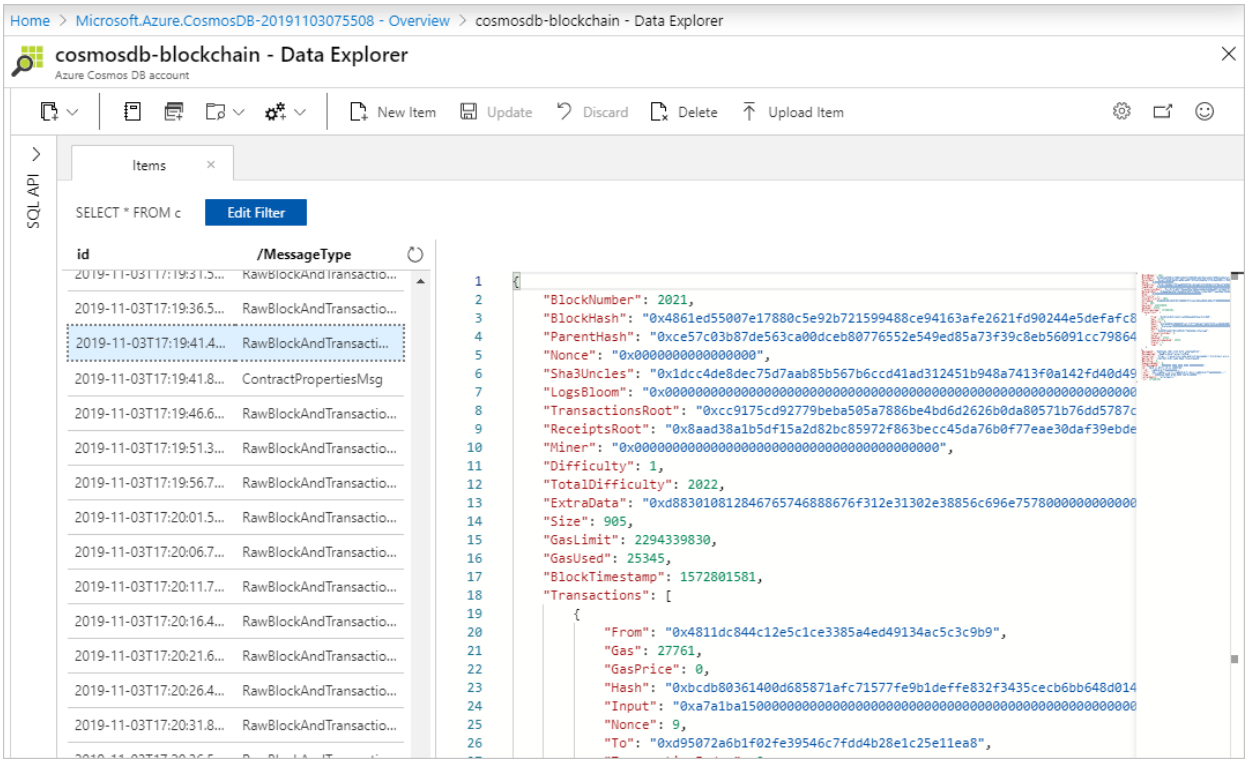
Tutorial: Use Blockchain Data Manager to send data to Azure Cosmos DB

6/16/2021 • 14 minutes to read • Edit Online

In this tutorial, you use Blockchain Data Manager for Azure Blockchain Service to record blockchain transaction data in Azure Cosmos DB.

> **IMPORTANT**
>
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

Blockchain Data Manager captures, transforms, and delivers blockchain ledger data to Azure Event Grid Topics. From Azure Event Grid, you use an Azure Logic App connector to create documents in an Azure Cosmos DB database. When finished with tutorial, you can explore blockchain transaction data in Azure Cosmos DB Data Explorer.



In this tutorial, you:

- Create a Blockchain Data Manager instance
- Add a blockchain application to decode transaction properties and events
- Create an Azure Cosmos DB account and database to store transaction data
- Create an Azure Logic App to connect an Azure Event Grid Topic to Azure Cosmos DB
- Send a transaction to a blockchain ledger
- View the decoded transaction data in Azure Cosmos DB

If you don't have an Azure subscription, create a free account before you begin.

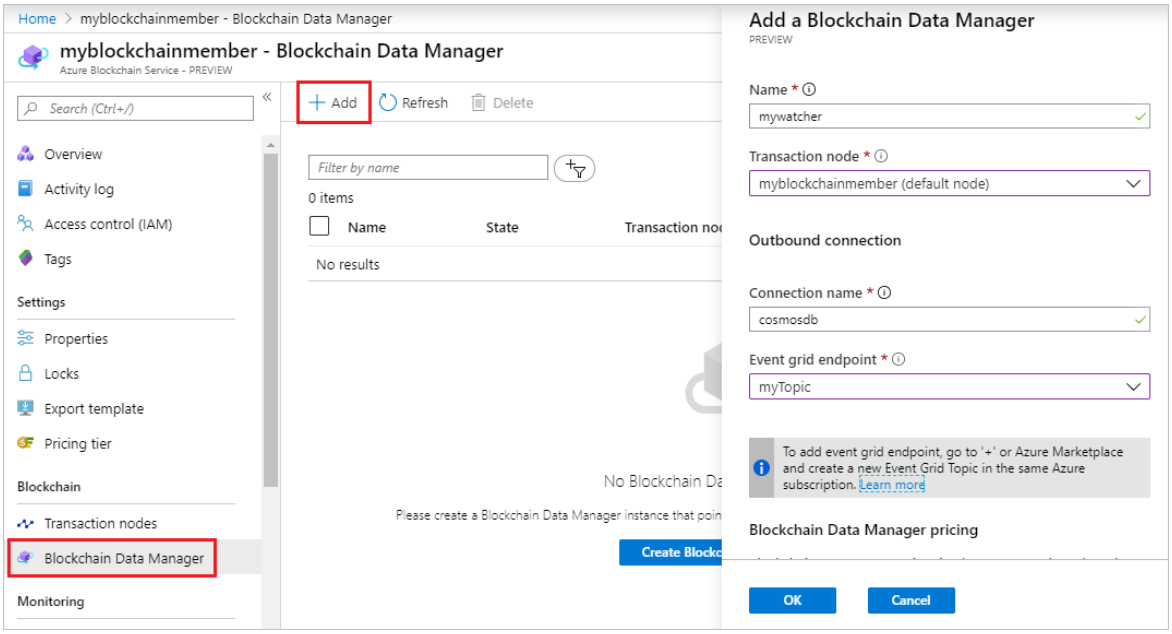
Prerequisites

- Complete Quickstart: Create a blockchain member using the Azure portal or Quickstart: Create an Azure Blockchain Service blockchain member using Azure CLI
- Complete Quickstart: Use Visual Studio Code to connect to an Azure Blockchain Service consortium network. The quickstart guides you though installing Azure Blockchain Development Kit for Ethereum and setting up your blockchain development environment.
- Complete Tutorial: Use Visual Studio Code to create, build, and deploy smart contracts. The tutorial walks through creating a sample smart contract.
- Create an Event Grid Topic
- Learn about Event handlers in Azure Event Grid

Create instance

A Blockchain Data Manager instance connects and monitors an Azure Blockchain Service transaction node. An instance captures all raw block and raw transaction data from the transaction node. An outbound connection sends blockchain data to Azure Event Grid. You configure a single outbound connection when you create the instance.

1. Sign in to the Azure portal.
2. Go to the Azure Blockchain Service member you created in the prerequisite Quickstart: Create a blockchain member using the Azure portal. Select **Blockchain Data Manager**.
3. Select **Add**.



Enter the following details:

SETTING	EXAMPLE	DESCRIPTION
Name	mywatcher	Enter a unique name for a connected Blockchain Data Manager.
Transaction node	myblockchainmember	Choose the default transaction node of the Azure Blockchain Service member you created in the prerequisite.

SETTING	EXAMPLE	DESCRIPTION
Connection name	cosmosdb	Enter a unique name of the outbound connection where blockchain transaction data is sent.
Event grid endpoint	myTopic	Choose an event grid topic you created in the prerequisite. Note: The Blockchain Data Manager instance and the event grid topic must be in the same subscription.

4. Select **OK**.

   It takes less than a minute to create a Blockchain Data Manager instance. After the instance is deployed, it is automatically started. A running Blockchain Data Manager instance captures blockchain events from the transaction node and sends data to event grid.

Add application

Add the **helloblockchain** blockchain application so that Blockchain Data Manager decodes event and property state. Blockchain Data Manager requires the smart contract ABI and bytecode file to add the application.

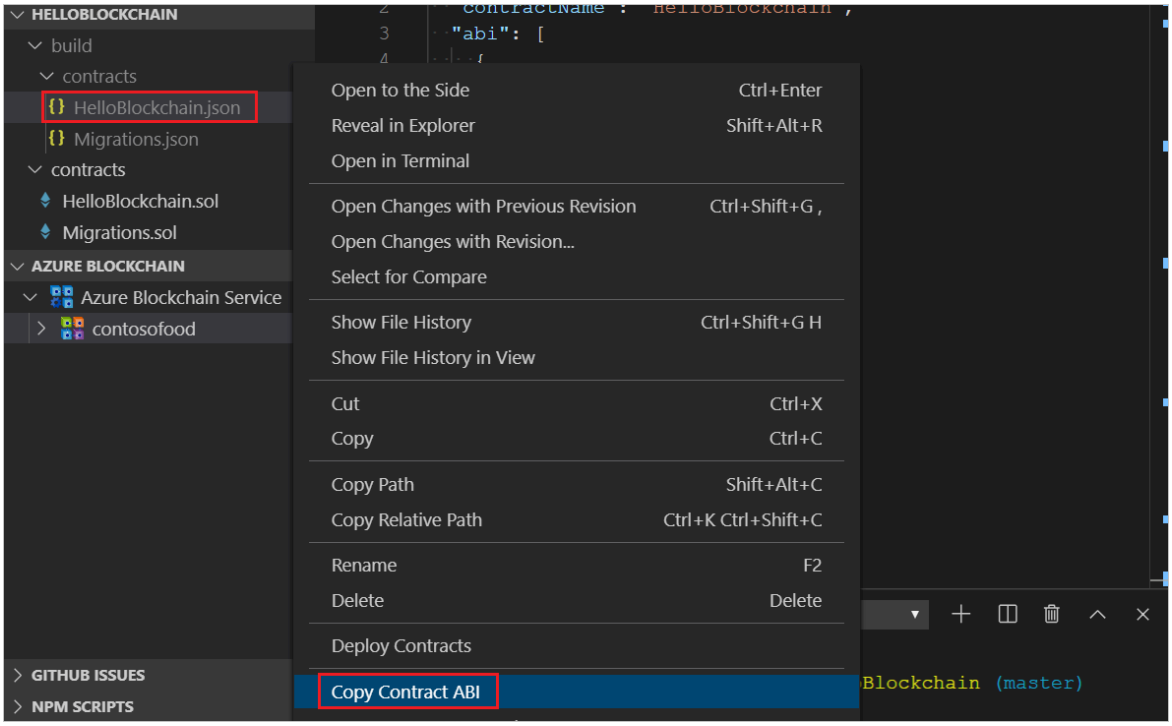
Get contract ABI and bytecode

The contract ABI defines the smart contract interfaces. It describes how to interact with the smart contract. You can use the Azure Blockchain Development Kit for Ethereum extension to copy the contract ABI to the clipboard.

1. In the Visual Studio Code explorer pane, expand the **build/contracts** folder of the **helloblockchain** Solidity project you created in the prerequisite Tutorial: Use Visual Studio Code to create, build, and deploy smart contracts.

2. Right-click the contract metadata JSON file. The file name is the smart contract name followed by the **.json** extension.

3. Select **Copy Contract ABI**.

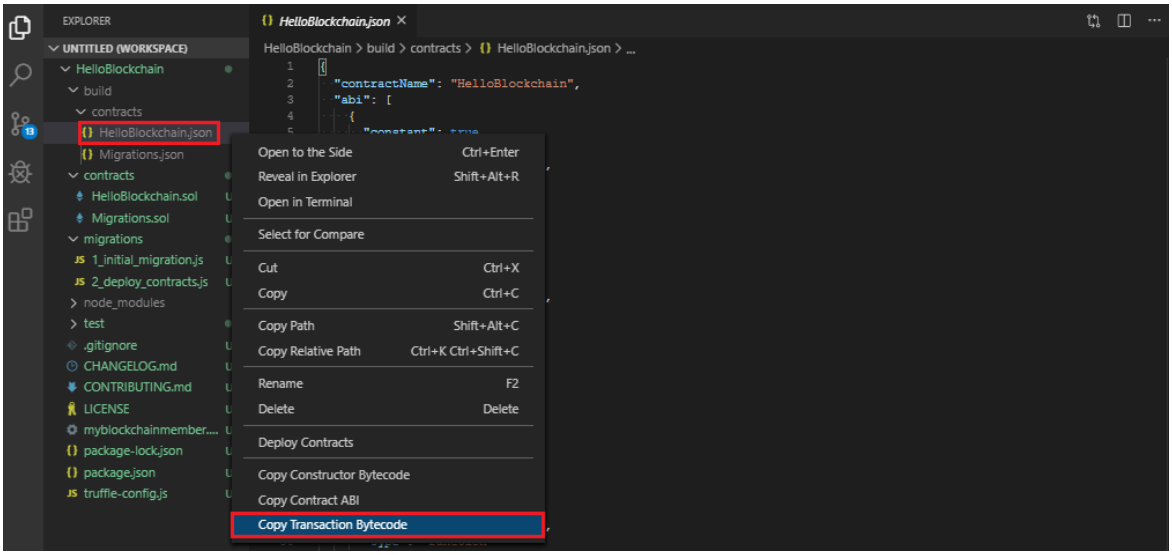


   The contract ABI is copied to the clipboard.

4. Save the **abi** array as a JSON file. For example, *abi.json*. You use the file in a later step.

Blockchain Data Manager requires the deployed bytecode for the smart contract. The deployed bytecode is different than the smart contract bytecode. You use the Azure blockchain development kit extension to copy the bytecode to the clipboard.

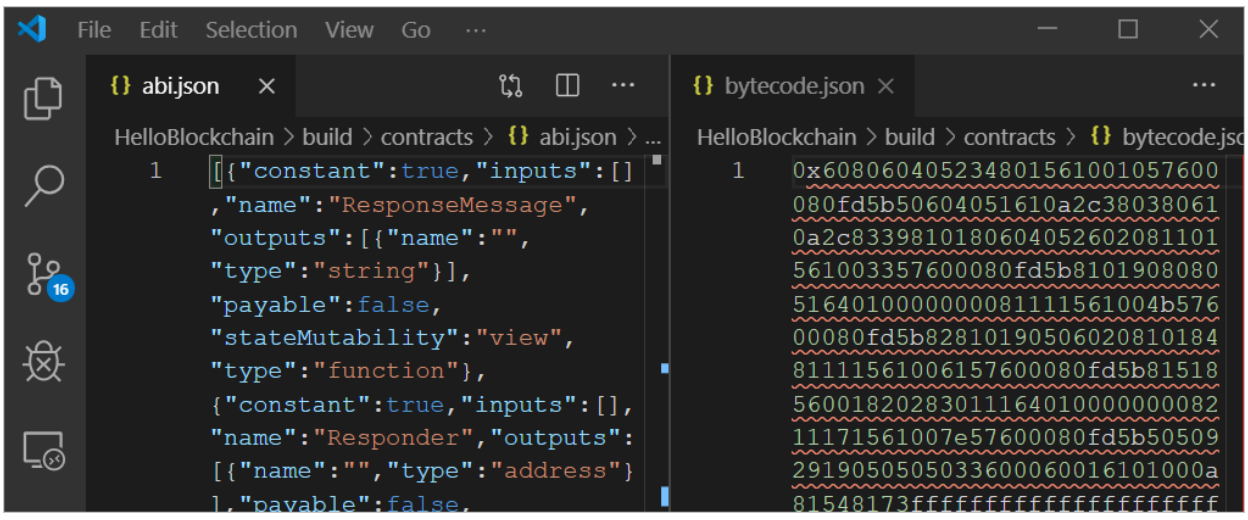
1. In the Visual Studio Code explorer pane, expand the **build/contracts** folder of your Solidity project.
2. Right-click the contract metadata JSON file. The file name is the smart contract name followed by the **.json** extension.
3. Select **Copy Transaction Bytecode**.



The bytecode is copied to the clipboard.

4. Save the **bytecode** value as a JSON file. For example, *bytecode.json*. You use the file in a later step.

The following example shows *abi.json* and *bytecode.json* files open in the VS Code editor. Your files should look similar.



Create contract ABI and bytecode URL

Blockchain Data Manager requires the contract ABI and bytecode files to be accessible by a URL when adding an application. You can use an Azure Storage account to provide a privately accessible URL.

Create storage account

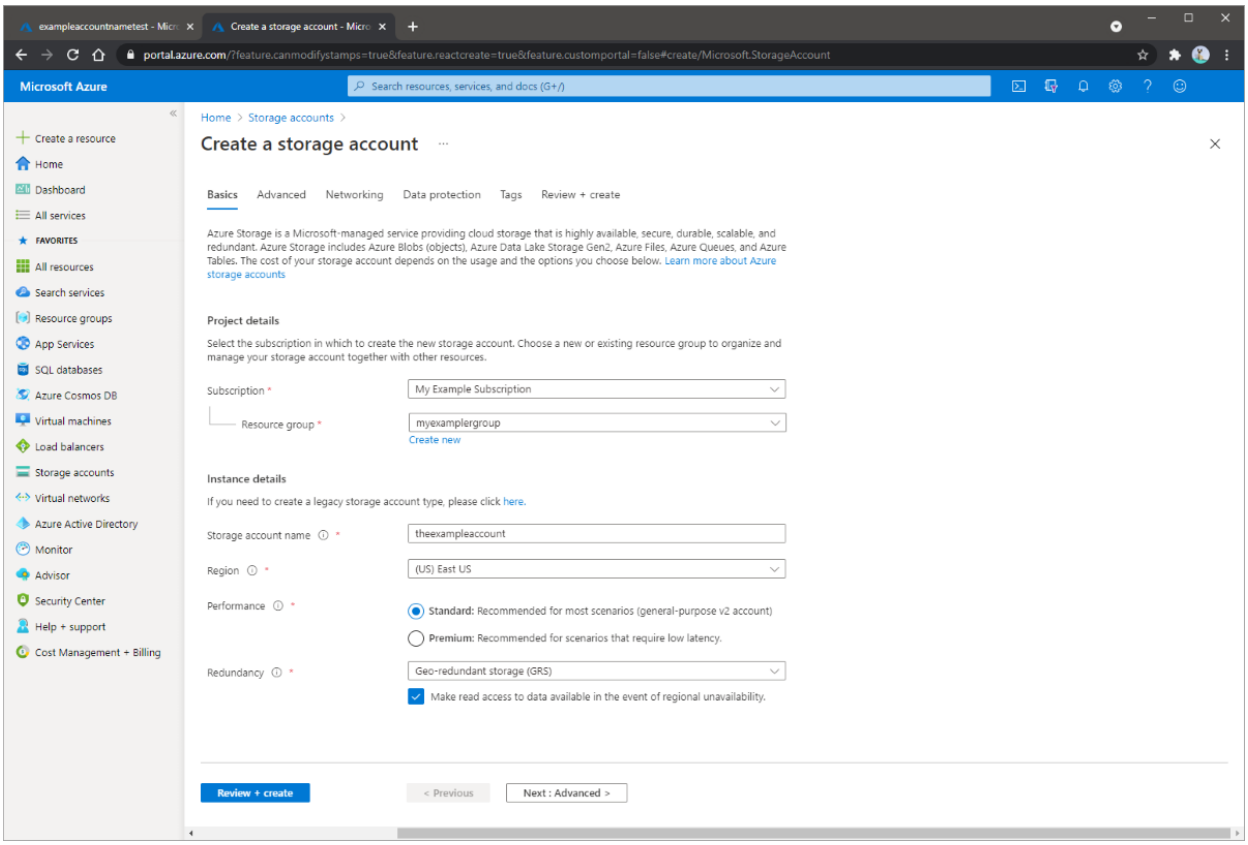
To create a general-purpose v2 storage account in the Azure portal, follow these steps:

1. On the Azure portal menu, select **All services**. In the list of resources, type **Storage Accounts**. As you

begin typing, the list filters based on your input. Select **Storage Accounts**.

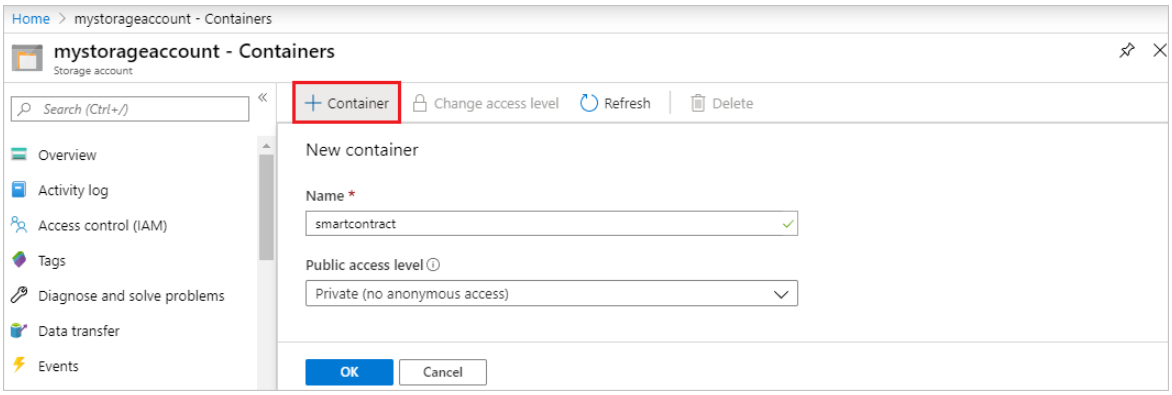
2. On the **Storage Accounts** window that appears, choose **+ New**.
3. On the **Basics** blade, select the subscription in which to create the storage account.
4. Under the **Resource group** field, select your desired resource group, or create a new resource group. For more information on Azure resource groups, see Azure Resource Manager overview.
5. Next, enter a name for your storage account. The name you choose must be unique across Azure. The name also must be between 3 and 24 characters in length, and may include only numbers and lowercase letters.
6. Select a region for your storage account, or use the default region.
7. Select a performance tier. The default tier is *Standard*.
8. Specify how the storage account will be replicated. The default redundancy option is *Geo-redundant storage (GRS)*. For more information about available replication options, see Azure Storage redundancy.
9. Additional options are available on the **Advanced**, **Networking**, **Data protection**, and **Tags** blades. To use Azure Data Lake Storage, choose the **Advanced** blade, and then set **Hierarchical namespace** to **Enabled**. For more information, see Azure Data Lake Storage Gen2 Introduction
10. Select **Review + Create** to review your storage account settings and create the account.
11. Select **Create**.

The following image shows the settings on the **Basics** blade for a new storage account:



Upload contract files

1. Create a new container for the storage account. Select **Containers > Container**.

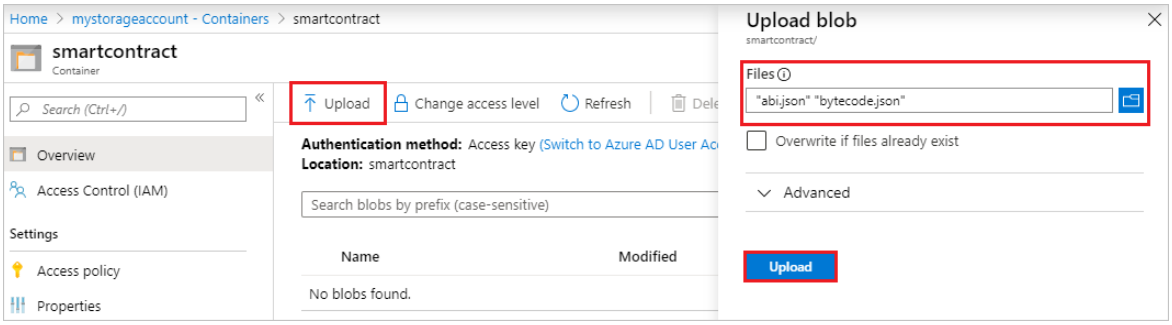


SETTING	DESCRIPTION
Name	Name the container. For example, *smartcontract*
Public access level	Choose *Private (no anonymous access)*

2. Select **OK** to create the container.

3. Select the container then select **Upload**.

4. Choose both JSON files you created in the Get Contract ABI and bytecode section.



Select **Upload**.

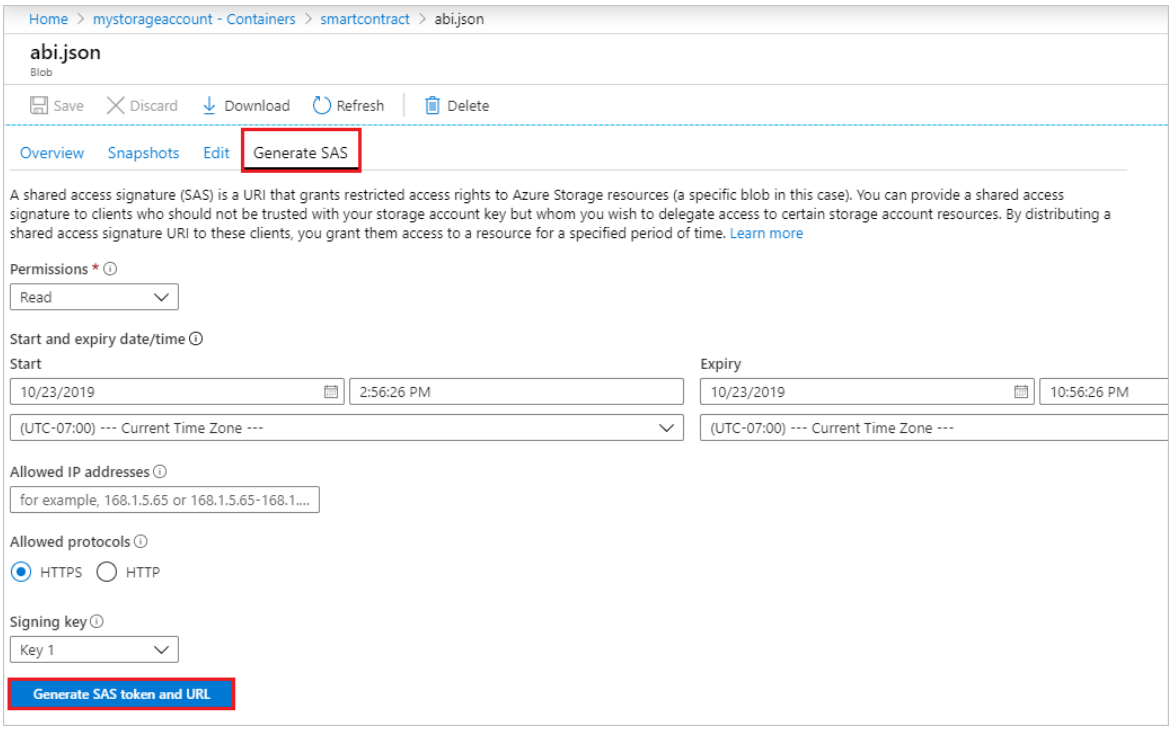
Generate URL

For each blob, generate a shared access signature.

1. Select the ABI JSON blob.

2. Select **Generate SAS**

3. Set desired access signature expiration then select **Generate blob SAS token and URL**.



4. Copy the **Blob SAS URL** and save it for the next section.

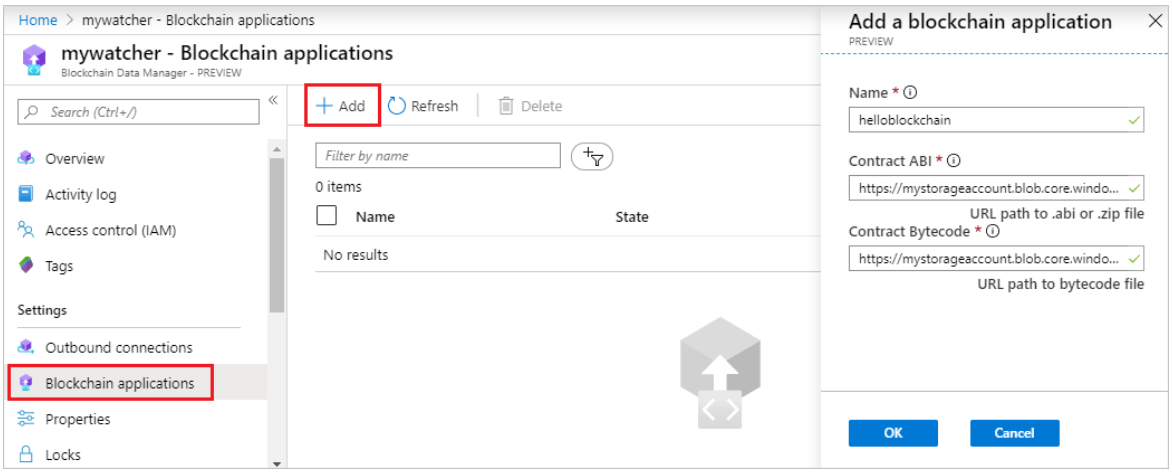
5. Repeat the Generate URL steps for the bytecode JSON blob.

Add helloblockchain application to instance

1. Select your Blockchain Data Manager instance from the instance list.

2. Select **Blockchain applications**.

3. Select **Add**.

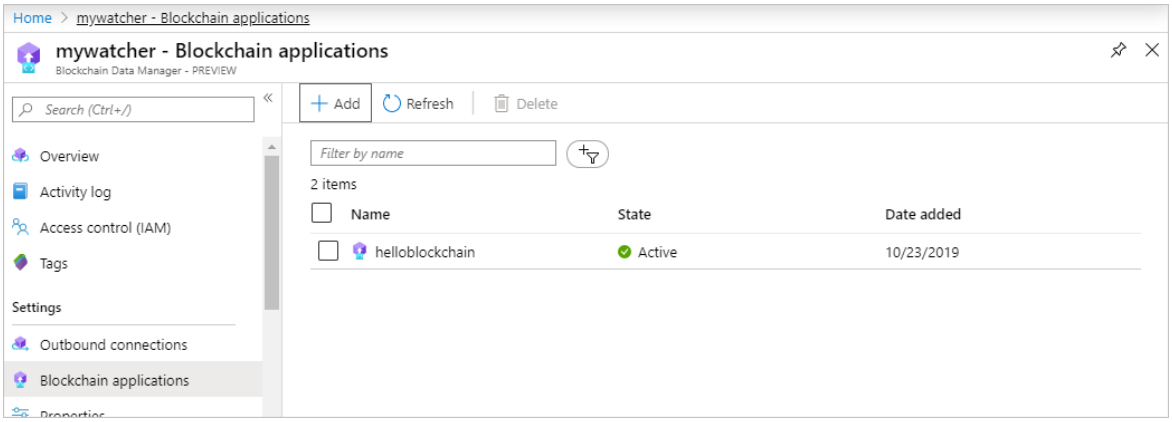


Enter the name of the blockchain application and the smart contract ABI and bytecode URLs.

SETTING	DESCRIPTION
Name	Enter a unique name for the blockchain application to track.
Contract ABI	URL path to the Contract ABI file. For more information, see Create contract ABI and bytecode URL.
Contract Bytecode	URL path to bytecode file. For more information, see Create contract ABI and bytecode URL.

4. Select **OK**.

   Once the application is created, the application appears in the list of blockchain applications.



You can delete the Azure Storage account or use it to configure more blockchain applications. If you wish to delete the Azure Storage account, you can delete the resource group. Deleting the resource group also deletes the associated storage account, and any other resources associated with the resource group.

Create Azure Cosmos DB

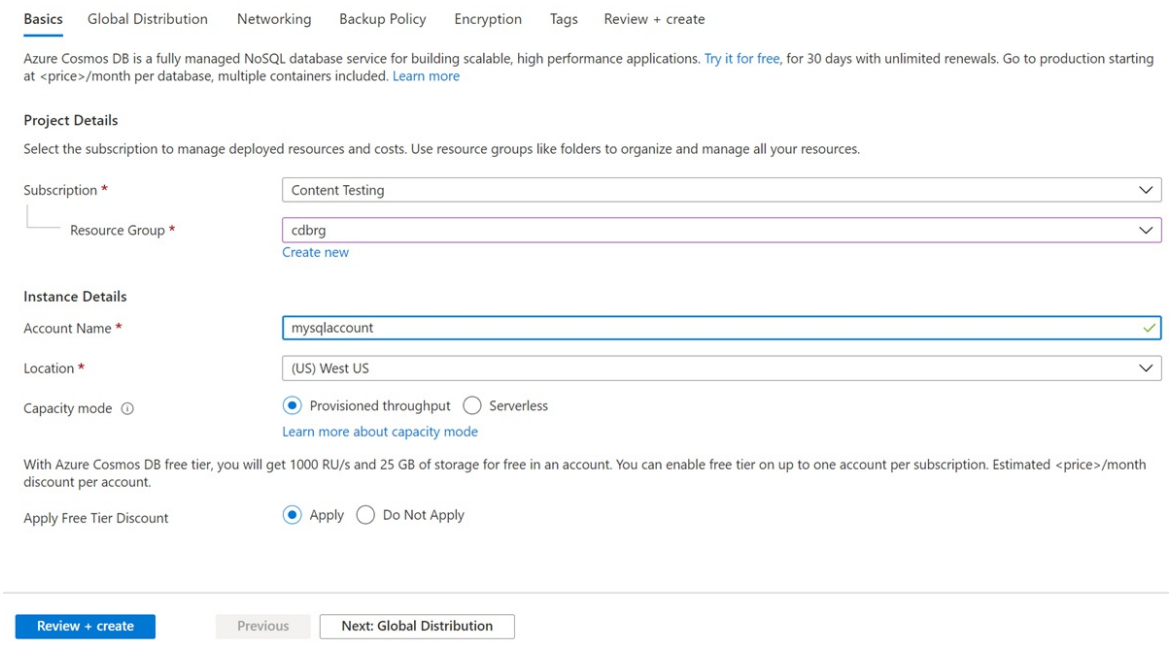
1. From the Azure portal menu or the **Home page**, select **Create a resource**.
2. On the **New** page, search for and select **Azure Cosmos DB**.
3. On the **Azure Cosmos DB** page, select **Create**.
4. In the **Create Azure Cosmos DB Account** page, enter the basic settings for the new Azure Cosmos account.

SETTING	VALUE	DESCRIPTION
Subscription	Subscription name	Select the Azure subscription that you want to use for this Azure Cosmos account.
Resource Group	Resource group name	Select a resource group, or select **Create new**, then enter a unique name for the new resource group.
Account Name	A unique name	Enter a name to identify your Azure Cosmos account. Because *documents.azure.com* is appended to the name that you provide to create your URI, use a unique name. The name can only contain lowercase letters, numbers, and the hyphen (-) character. It must be between 3-44 characters in length.

SETTING	VALUE	DESCRIPTION
API	The type of account to create	Select **Core (SQL)** to create a document database and query by using SQL syntax. The API determines the type of account to create. Azure Cosmos DB provides five APIs: Core (SQL) and MongoDB for document data, Gremlin for graph data, Azure Table, and Cassandra. Currently, you must create a separate account for each API. Learn more about the SQL API.
Location	The region closest to your users	Select a geographic location to host your Azure Cosmos DB account. Use the location that is closest to your users to give them the fastest access to the data.
Capacity mode	Provisioned throughput or Serverless	Select **Provisioned throughput** to create an account in provisioned throughput mode. Select **Serverless** to create an account in serverless mode.
Apply Azure Cosmos DB free tier discount	**Apply** or **Do not apply**	With Azure Cosmos DB free tier, you will get the first 1000 RU/s and 25 GB of storage for free in an account. Learn more about free tier.

NOTE

You can have up to one free tier Azure Cosmos DB account per Azure subscription and must opt-in when creating the account. If you do not see the option to apply the free tier discount, this means another account in the subscription has already been enabled with free tier.



5. In the **Global Distribution** tab, configure the following details. You can leave the default values for the purpose of this quickstart:

SETTING	VALUE	DESCRIPTION
Geo-Redundancy	Disable	Enable or disable global distribution on your account by pairing your region with a pair region. You can add more regions to your account later.
Multi-region Writes	Disable	Multi-region writes capability allows you to take advantage of the provisioned throughput for your databases and containers across the globe.

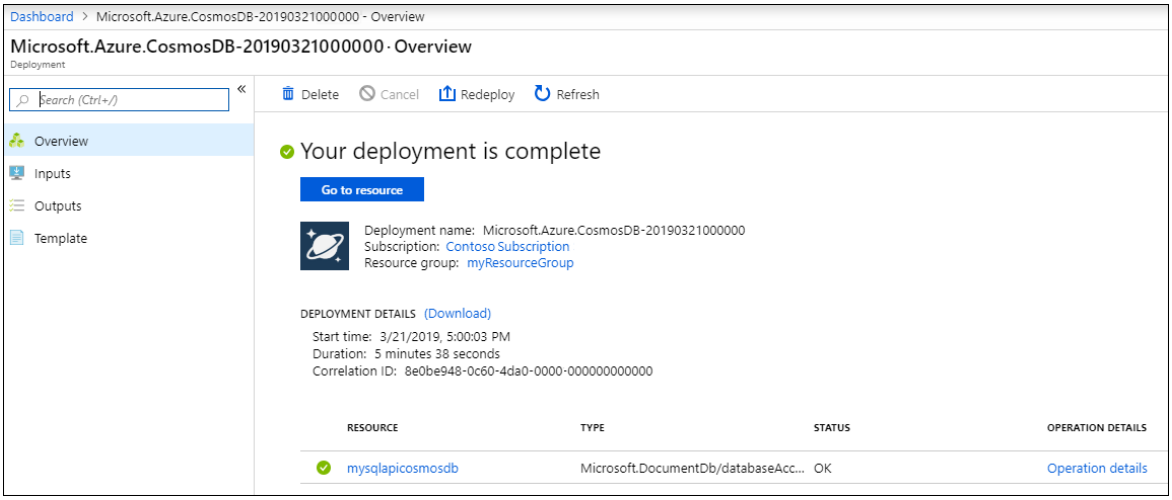
> **NOTE**
>
> The following options are not available if you select **Serverless** as the **Capacity mode**:
>
> - Apply Free Tier Discount
> - Geo-redundancy
> - Multi-region Writes

6. Optionally you can configure additional details in the following tabs:

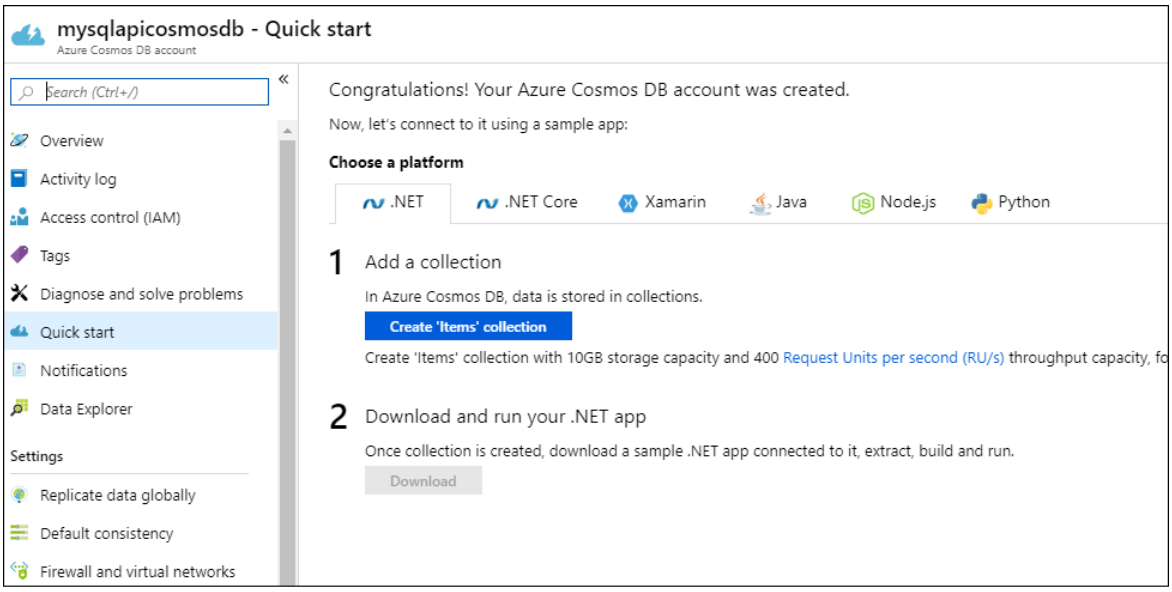
   - **Networking** - Configure access from a virtual network.
   - **Backup Policy** - Configure either periodic or continuous backup policy.
   - **Encryption** - Use either service-managed key or a customer-managed key.
   - **Tags** - Tags are name/value pairs that enable you to categorize resources and view consolidated billing by applying the same tag to multiple resources and resource groups.

7. Select **Review + create**.

8. Review the account settings, and then select **Create**. It takes a few minutes to create the account. Wait for the portal page to display **Your deployment is complete**.



9. Select **Go to resource** to go to the Azure Cosmos DB account page.

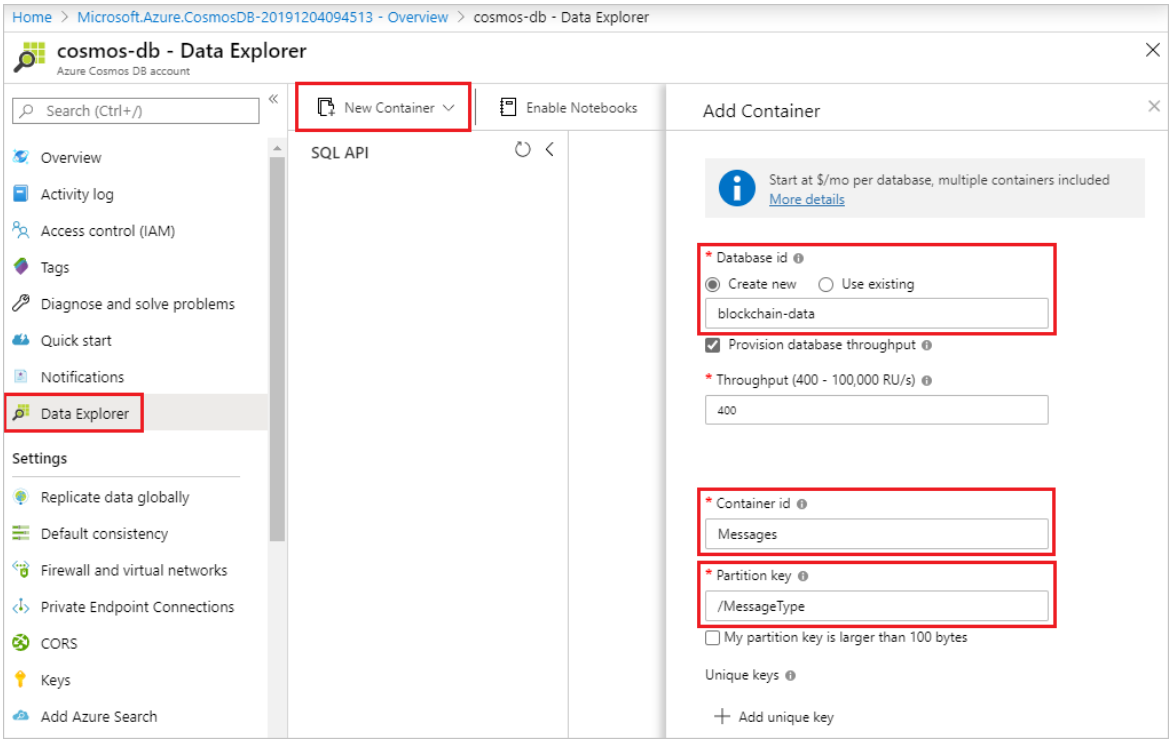


Add a database and container

You can use the Data Explorer in the Azure portal to create a database and container.

1. Select **Data Explorer** from the left navigation on your Azure Cosmos DB account page, and then select **New Container**.

2. In the **Add container** pane, enter the settings for the new container.



SETTING	DESCRIPTION
Database ID	Enter **blockchain-data** as the name for the new database.
Throughput	Leave the throughput at **400** request units per second (RU/s). If you want to reduce latency, you can scale up the throughput later.
Container ID	Enter **Messages** as the name for your new container.

SETTING	DESCRIPTION
Partition key	Use **/MessageType** as the partition key.

3. Select **OK**. The Data Explorer displays the new database and the container that you created.

Create Logic App

Azure Logic Apps helps you schedule and automate business processes and workflows when you need to integrate systems and services. You can use a logic app to connect Event Grid to Azure Cosmos DB.

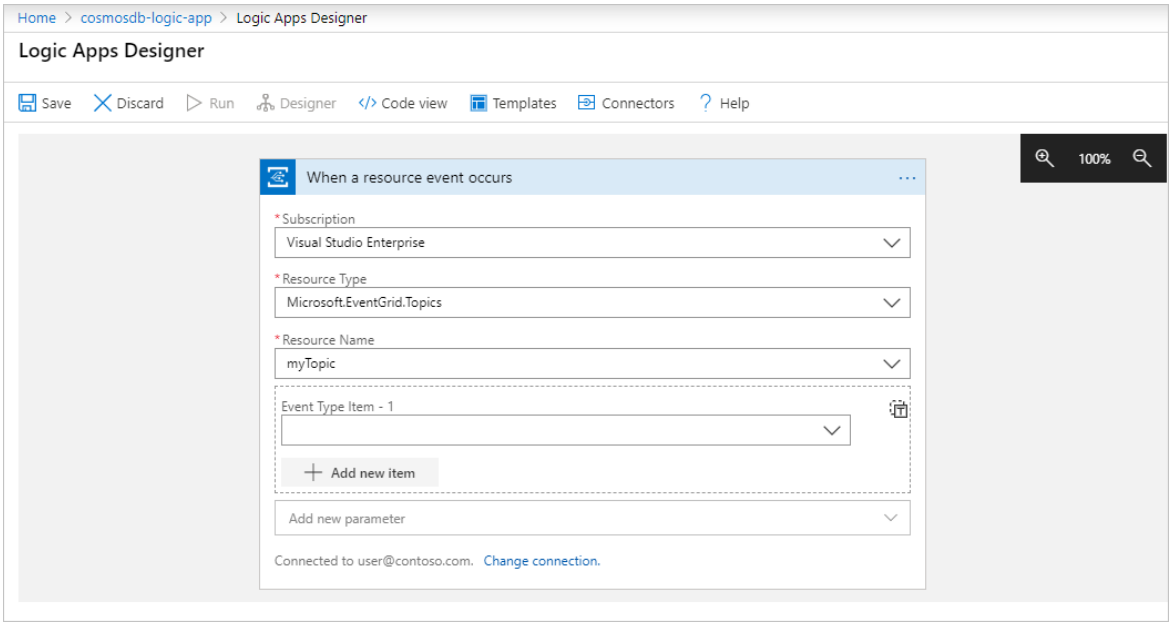
1. In the Azure portal, select **Create a resource** > **Integration** > **Logic App**.
2. Provide details on where to create your logic app. After you're done, select **Create**.

   For more information on creating logic apps, see Create automated workflows with Azure Logic Apps.
3. After Azure deploys your app, select your logic app resource.
4. In the Logic Apps Designer, under **Templates**, select **Blank Logic App**.

Add Event Grid trigger

Every logic app must start with a trigger, which fires when a specific event happens or when a specific condition is met. Each time the trigger fires, the Logic Apps engine creates a logic app instance that starts and runs your workflow. Use an Azure Event Grid trigger to sends blockchain transaction data from Event Grid to Cosmos DB.

1. In the Logic Apps Designer, search for and select the **Azure Event Grid** connector.
2. From the **Triggers** tab, select **When a resource event occurs**.
3. Create an API connection to your Event Grid Topic.



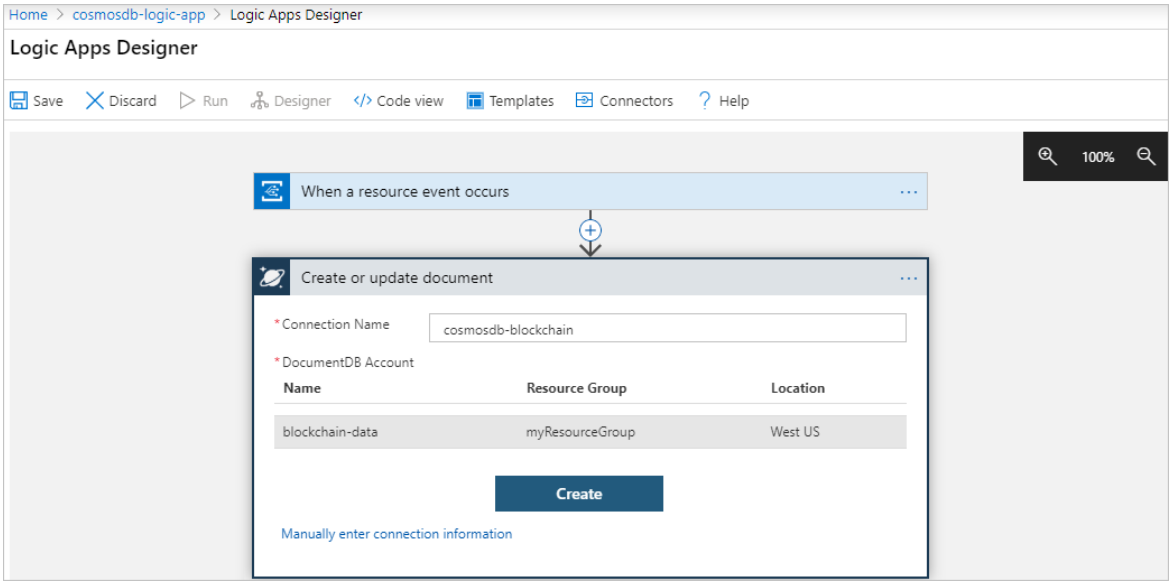
SETTING	DESCRIPTION
Subscription	Choose the subscription that contains the Event Grid Topic.
Resource Type	Choose **Microsoft.EventGrid.Topics**.

SETTING	DESCRIPTION
Resource Name	Choose the name of the Event Grid Topic where Blockchain Data Manager is sending transaction data messages.

Add Cosmos DB action

Add an action to create a document in Cosmos DB for each transaction. Use the transaction message type as the partition key to categorize the messages.

1. Select **New step**.
2. On **Choose an action**, search for **Azure Cosmos DB**.
3. Choose **Azure Cosmos DB > Actions > Create or update document**.
4. Create an API connection to your Cosmos DB database.



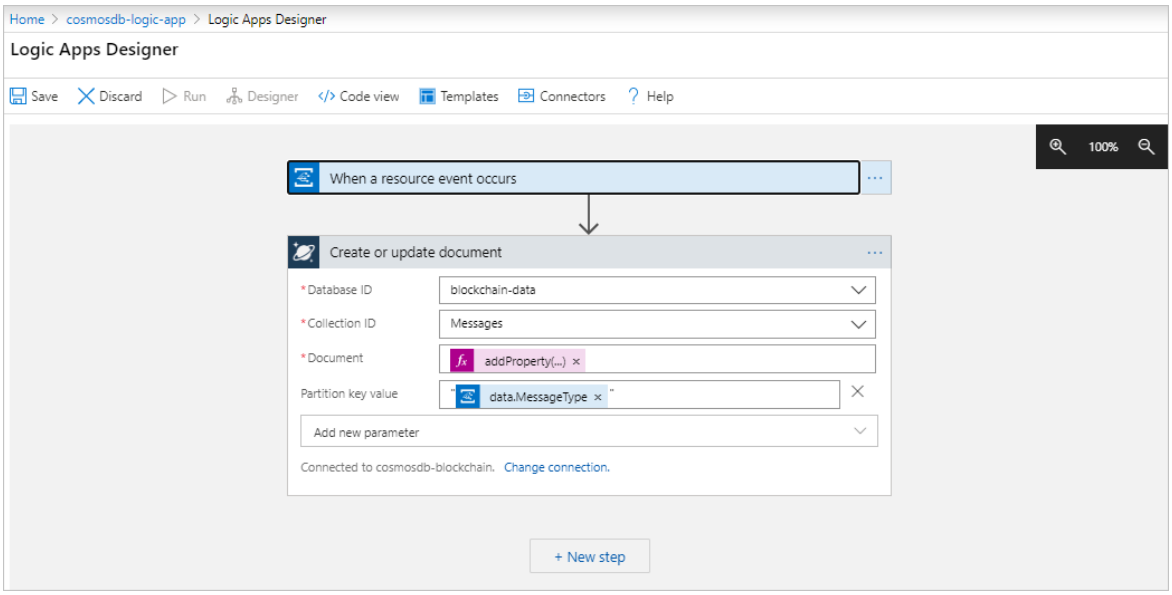
SETTING	DESCRIPTION
Connection Name	Choose the subscription that contains the Event Grid Topic.
DocumentDB Account	Choose the DocumentDB account you created in the Create Azure Cosmos DB account section.

5. Enter the **Database ID** and **Collection ID** for your Azure Cosmos DB that you created previously in the Add a database and container section.
6. Select the **Document** setting. In the *Add dynamic content* pop-out, select **Expression** and copy and paste the following expression:

```
addProperty(triggerBody()?['data'], 'id', utcNow())
```

   The expression gets the data portion of the message and sets the ID to a timestamp value.

7. Select **Add new parameter** and choose **Partition key value**.
8. Set the **Partition key value** to `"@{triggerBody()['data']['MessageType']}"`. The value must be surrounded by double quotes.



The value sets the partition key to the transaction message type.

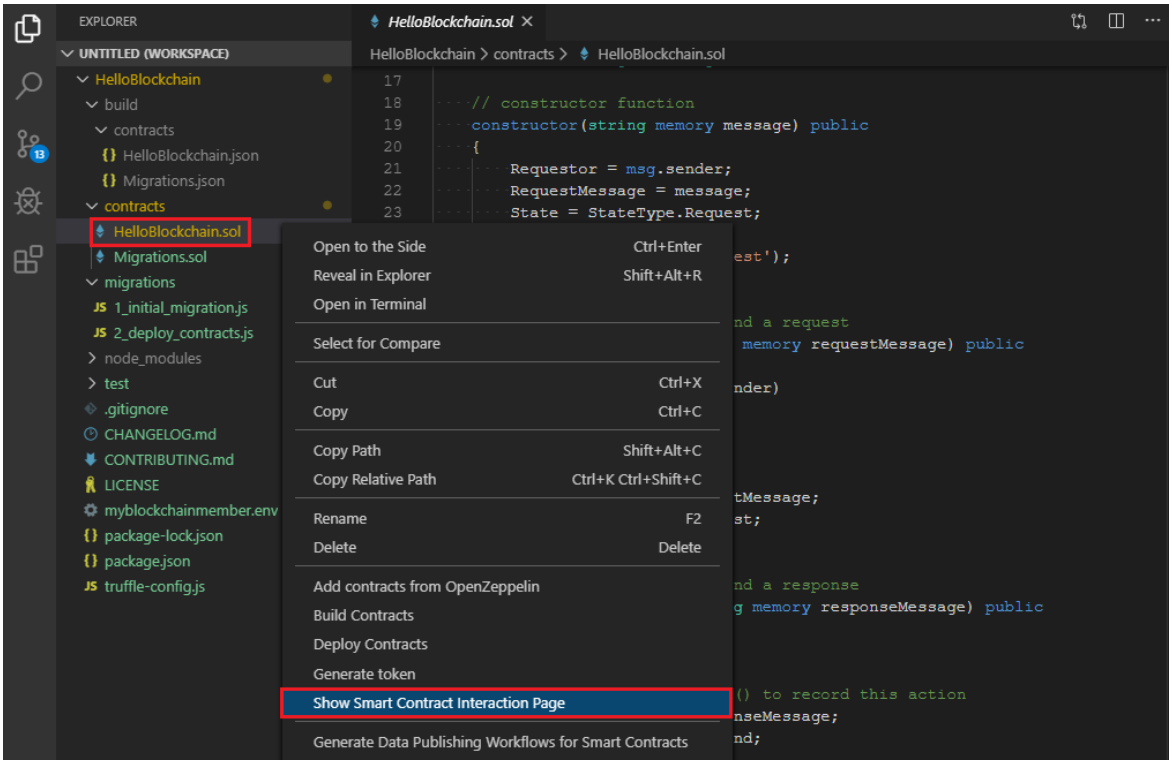
9. Select **Save**.

The logic app monitors the Event Grid Topic. When a new transaction message is sent from Blockchain Data Manager, the logic app creates a document in Cosmos DB.

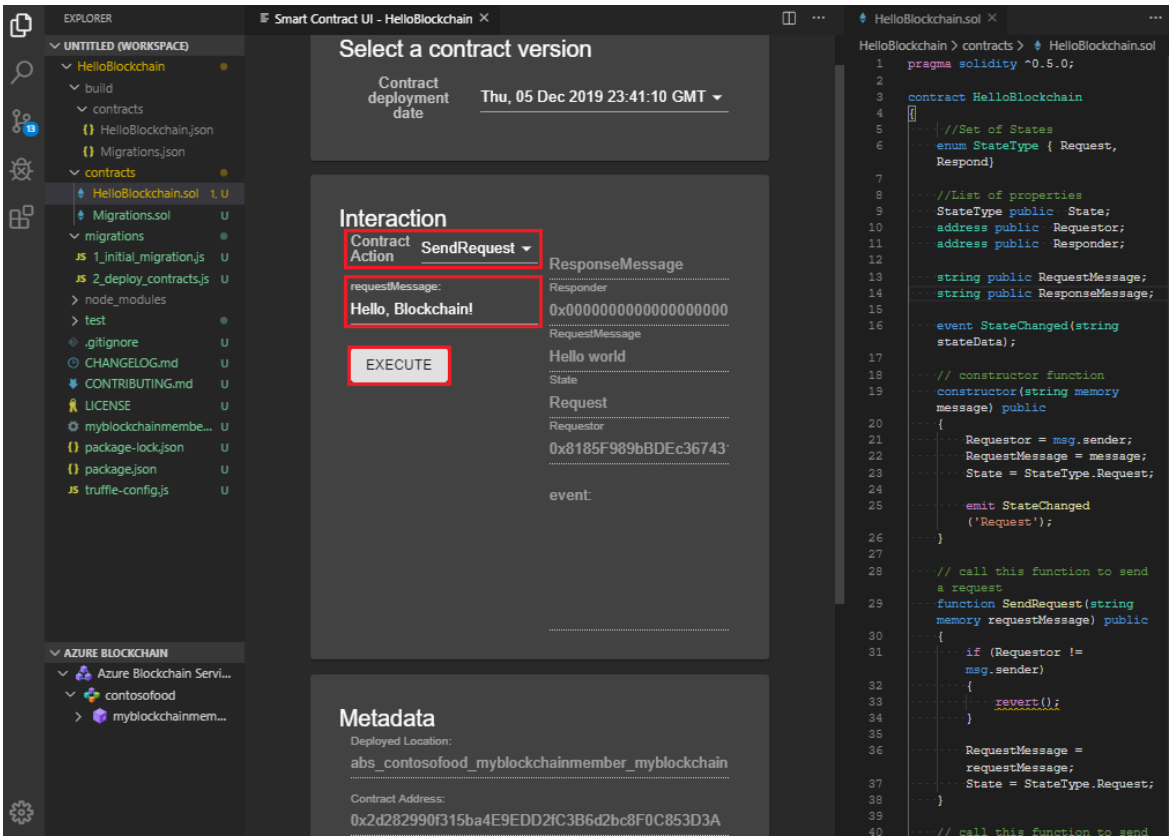
Send a transaction

Next, send a transaction to the blockchain ledger to test what you created. Use the **HelloBlockchain** contract's **SendRequest** function you created in the prerequisite Tutorial: Use Visual Studio Code to create, build, and deploy smart contracts.

1. Use the Azure Blockchain Development Kit smart contract interaction page to call the **SendRequest** function. Right-click **HelloBlockchain.sol** and choose **Show Smart Contract Interaction Page** from the menu.



2. Choose **SendRequest** contract action and enter **Hello, Blockchain!** for the **requestMessage** parameter. Select **Execute** to call the **SendRequest** function via a transaction.

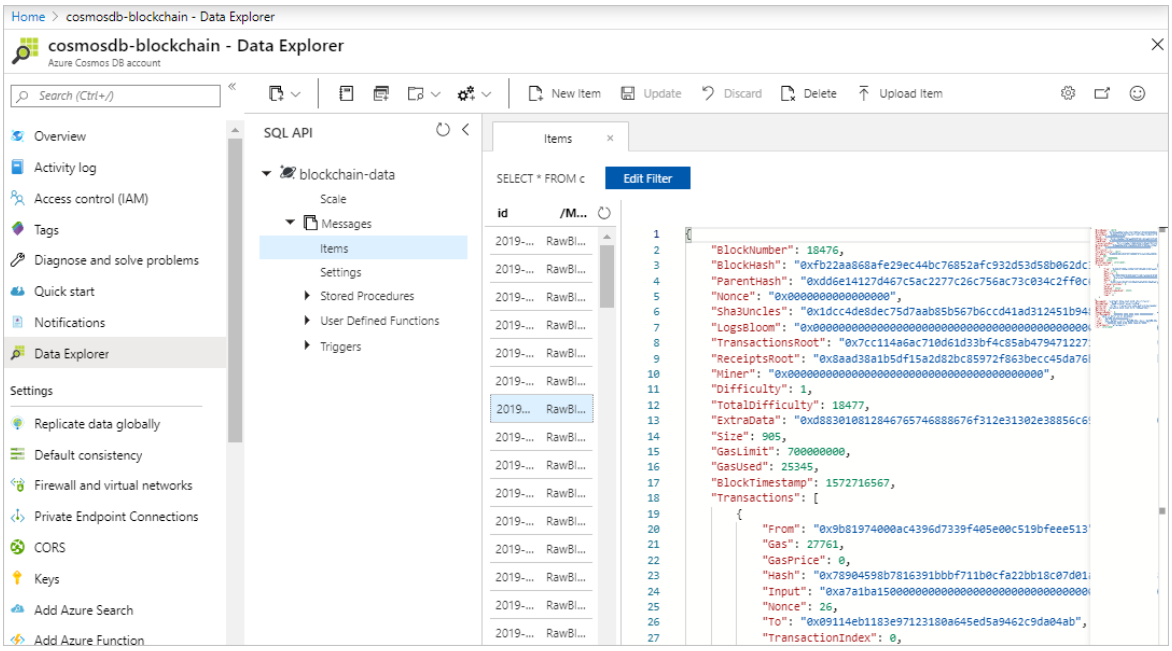


The SendRequest function sets the **RequestMessage** and **State** fields. The current state for **RequestMessage** is the argument you passed **Hello, Blockchain**. The **State** field value remains **Request**.

View transaction data

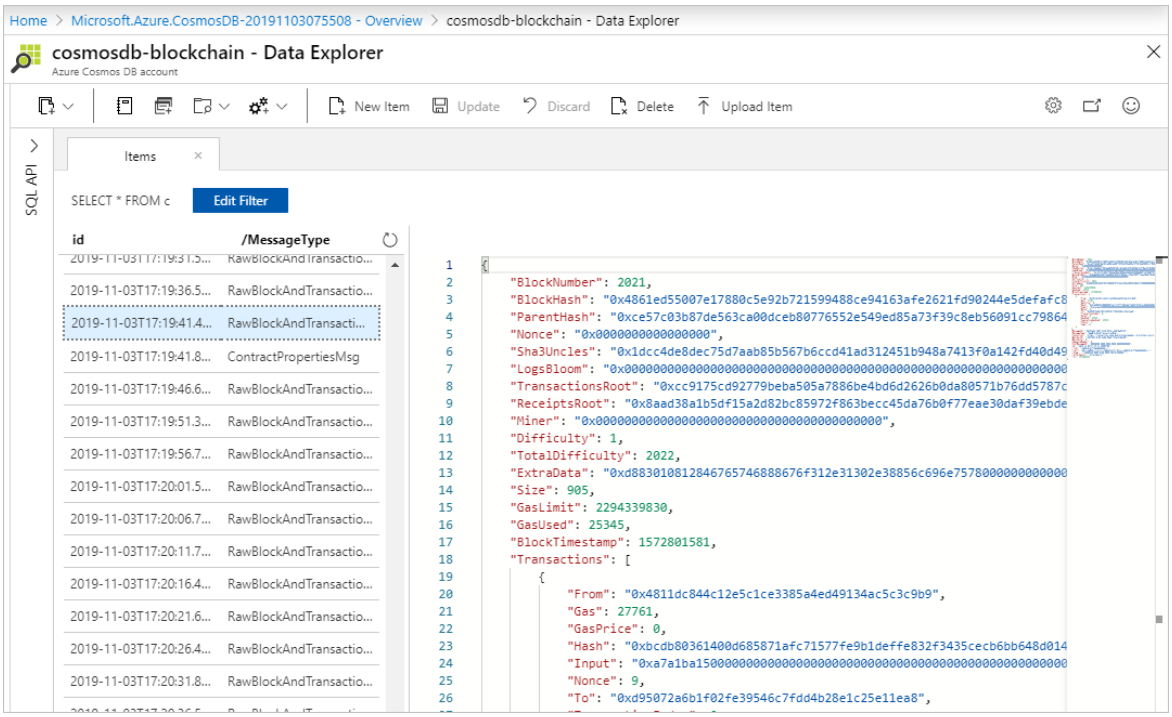
Now that you have connected your Blockchain Data Manager to Azure Cosmos DB, you can view the blockchain transaction messages in Cosmos DB Data Explorer.

1. Go to the Cosmos DB Data Explorer view. For example, **cosmosdb-blockchain > Data Explorer > blockchain-data > Messages > Items**.



Data Explorer lists the blockchain data messages that were created in the Cosmos DB database.

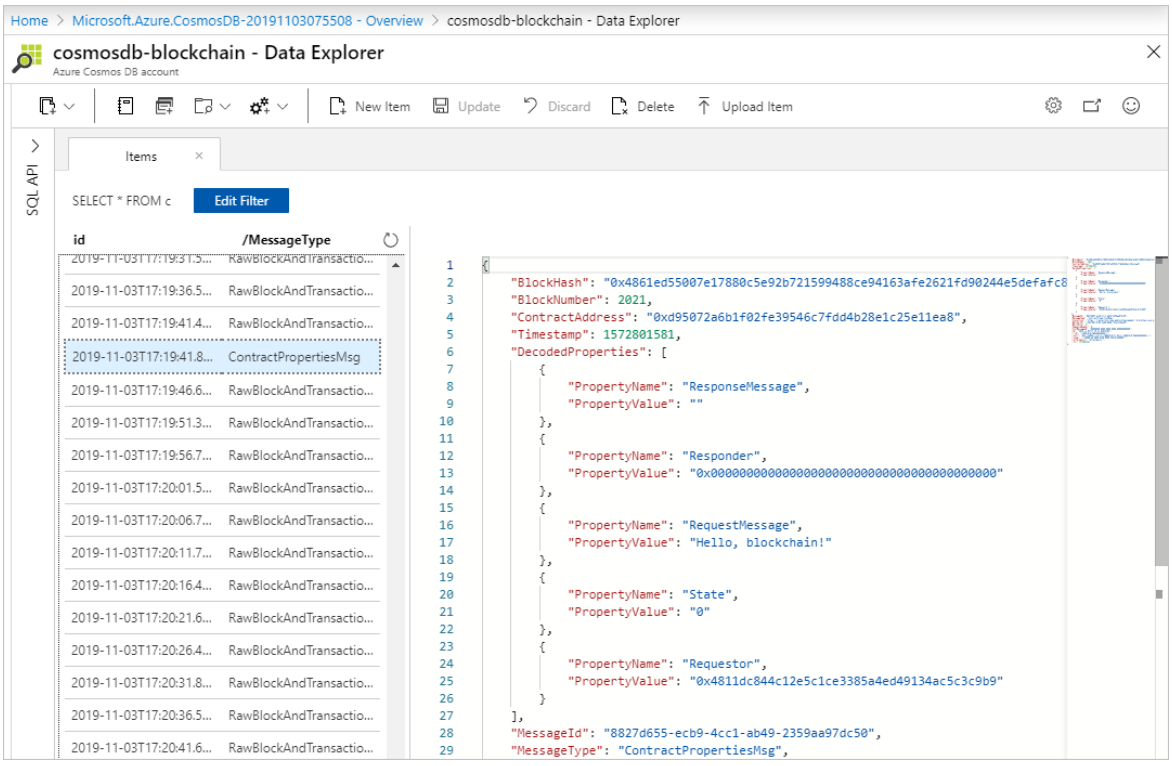
2. Browse through the messages by selecting item ID and find the message with the matching transaction hash.



The raw transaction message contains detail about the transaction. However, the property information is encrypted.

Since you added the HelloBlockchain smart contract to the Blockchain Data Manager instance, a **ContractProperties** message type is also sent that contains decoded property information.

3. Find the **ContractProperties** message for the transaction. It should be the next message in the list.



The **DecodedProperties** array contains the properties of the transaction.

Congratulations! You have successfully created a transaction message explorer using Blockchain Data Manager and Azure Cosmos DB.

Clean up resources

When no longer needed, you can delete the resources and resource groups you used for this tutorial. To delete a

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

Learn more about integrating with blockchain ledgers.

Using the Ethereum Blockchain connector with Azure Logic Apps

Azure Blockchain Service consortium

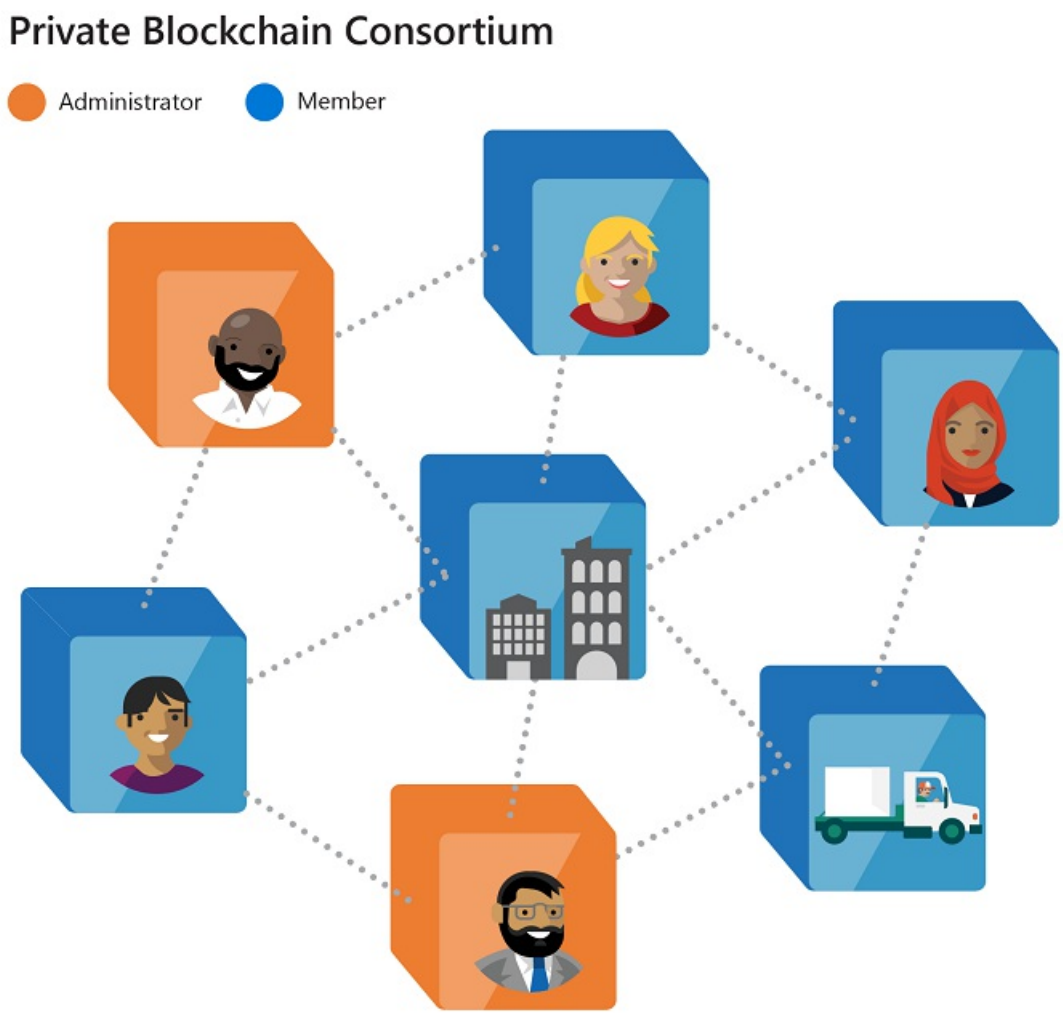
11/24/2019 • 3 minutes to read • Edit Online

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:



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.

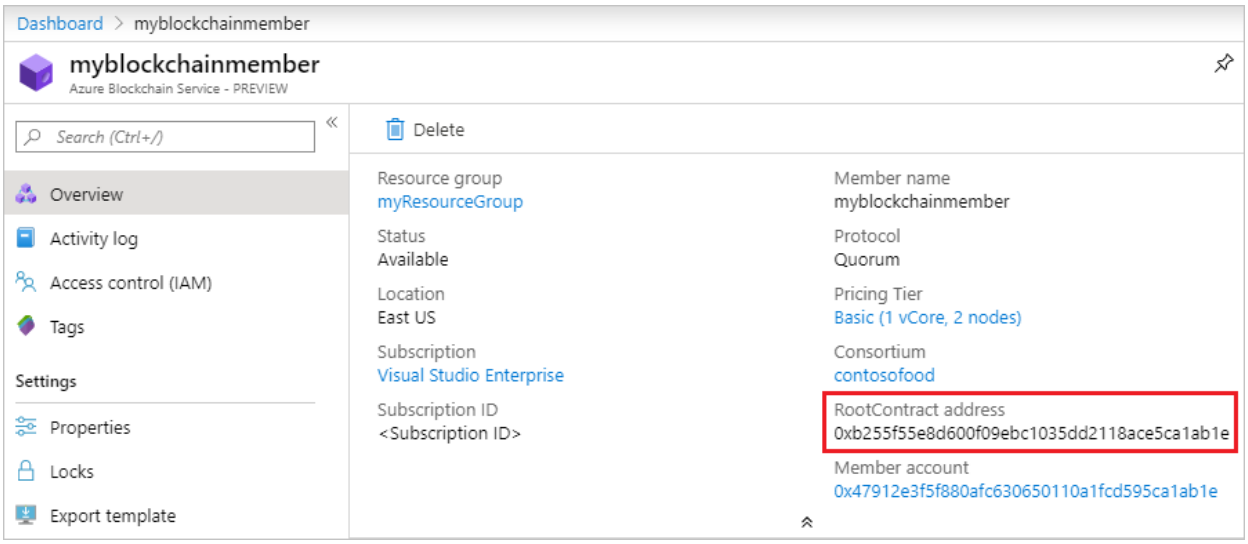
Once invited, participants can join the blockchain consortium by deploying a new member in Azure Blockchain Service. To view and join the invited consortium, you must specify the same Azure subscription ID used in the invite by the network administrator.

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.

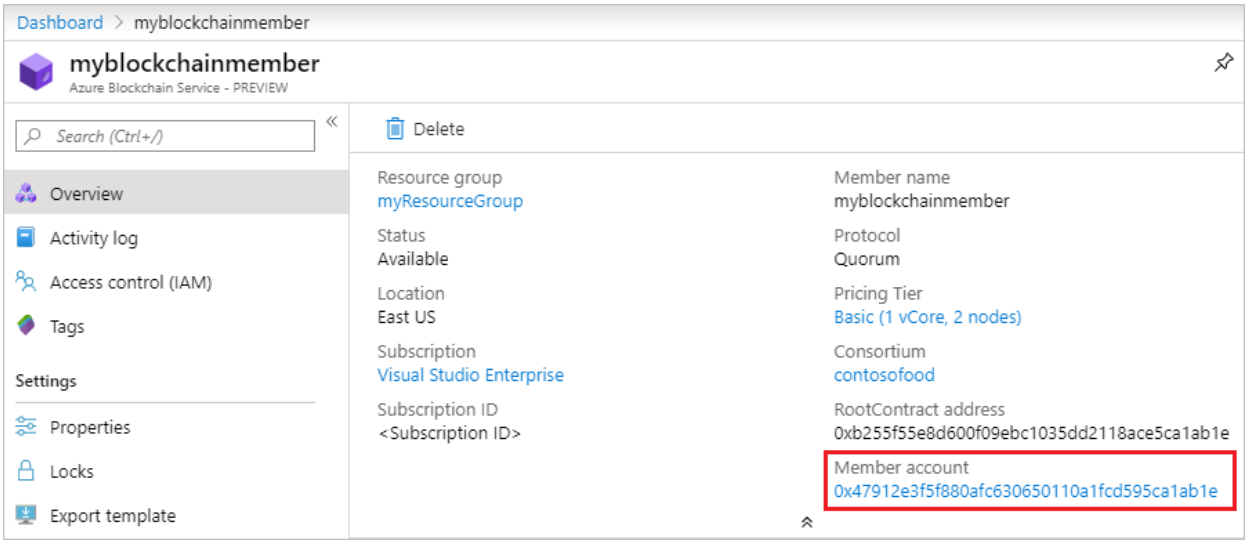


You can interact with the consortium management smart contract using the consortium management PowerShell module, Azure portal, or directly through the smart contract using the Azure Blockchain Service generated Ethereum account.

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.



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

Consortium management actions can be accessed through PowerShell. For more information, see Manage consortium members in Azure Blockchain Service using PowerShell.

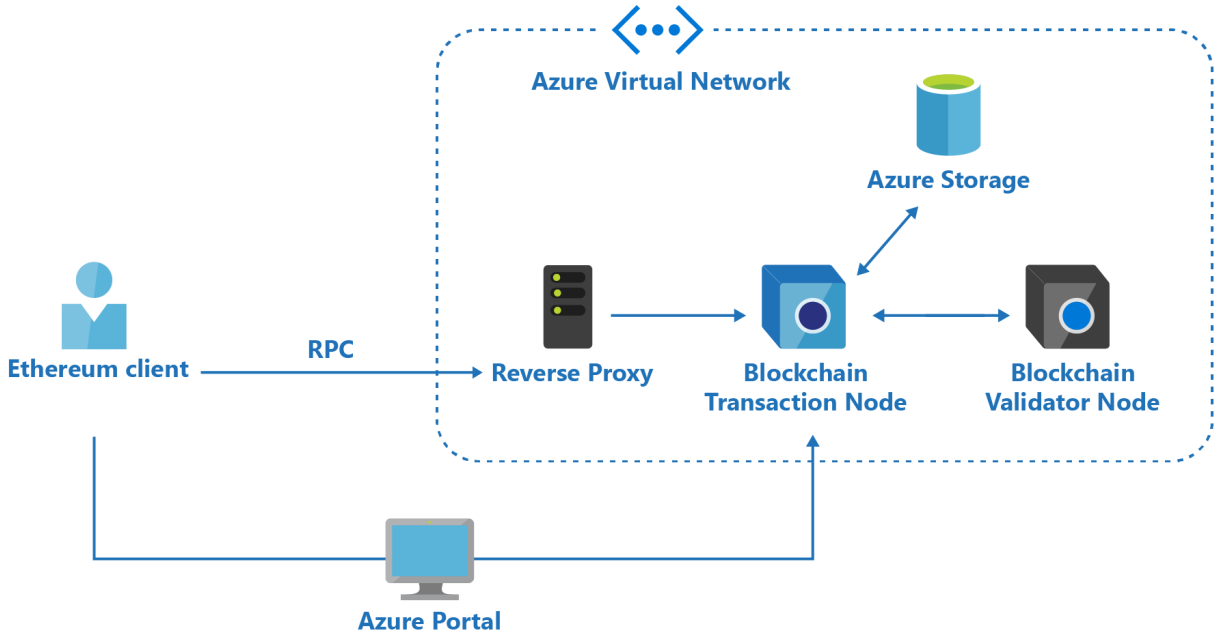
Azure Blockchain Service security

4/7/2020 • 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.

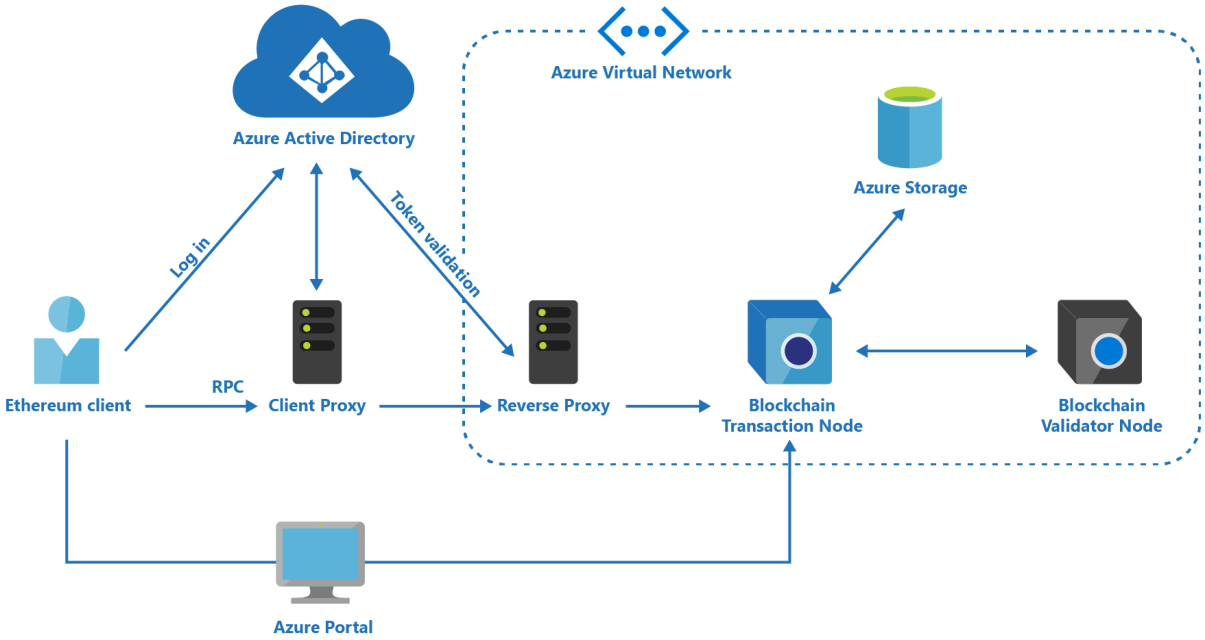


Encryption

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 TLS.



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

See How to configure Azure Active Directory access for Azure Blockchain Service.

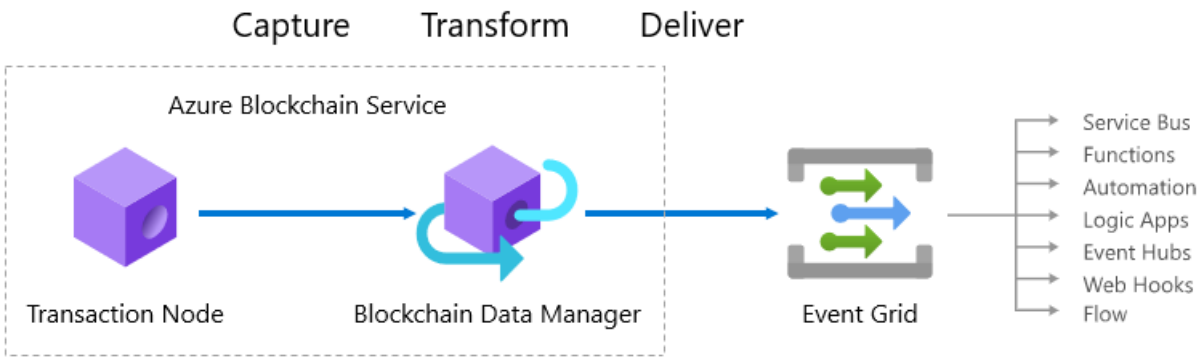
What is Blockchain Data Manager for Azure Blockchain Service?

2/14/2020 • 2 minutes to read • Edit Online

Blockchain Data Manager captures, transforms, and delivers Azure Blockchain Service transaction data to Azure Event Grid Topics providing reliable and scalable blockchain ledger integration with Azure services.

In most enterprise blockchain scenarios, a blockchain ledger is one part of a solution. For example, to transfer an asset from one entity to another, you need a mechanism for submitting the transaction. You then need a mechanism for reading ledger data to ensure the transaction occurred, was accepted, and the resulting state changes are then integrated with your end-to-end solution. In this example, if you write a smart contract to transfer assets, you can use Blockchain Data Manager to integrate off-chain applications and data stores. For the asset transfer example, when an asset is transferred on the blockchain, events and property state changes are delivered by Blockchain Data Manager via Event Grid. You can then use multiple possible event handlers for Event Grid to store blockchain data off-chain or react to state changes in real time.

Blockchain Data Manager performs three main functions: capture, transform, and deliver.



Capture

Each Blockchain Data Manager instance connects to one Azure Blockchain Service member transaction node. Only users with access to the transaction node can create a connection ensuring proper access control to customer data. A Blockchain Data Manager instance reliably captures all raw block and raw transaction data from the transaction node and can scale to support enterprise workloads.

Transform

You can use Blockchain Data Manager to decode event and property state by configuring smart contract applications within Blockchain Data Manager. To add a smart contract, you provide the contract ABI and bytecode. Blockchain Data Manager uses the smart contract artifacts to decode and discover contract addresses. After adding the blockchain application to the instance, Blockchain Data Manager dynamically discovers the smart contract address when the smart contract is deployed to the consortium and sends decoded event and property state to configured destinations.

Deliver

Blockchain Data Manager supports multiple Event Grid Topic outbound connections for any given Blockchain Data Manager instance. You can send blockchain data to a single destination or send blockchain data to multiple

destinations. Using Blockchain Data Manager, you can build a scalable event-based data publishing solution for any blockchain deployment.

Configuration options

You can configure Blockchain Data Manager to meet the needs of your solution. For example, you can provision:

- A single Blockchain Data Manager instance for an Azure Blockchain Service member.
- A Blockchain Data Manager instance per Azure Blockchain Service transaction node. For example, private transaction nodes can have their own Blockchain Data Manager instance to maintain confidentiality.
- A Blockchain Data Manager instance can support multiple output connections. One Blockchain Data Manager instance can be used to manage all data publishing integration points for an Azure Blockchain Service member.

Next steps

Try creating a Blockchain Data Manager instance for an Azure Blockchain Service member.

Azure Blockchain Service development overview

3/27/2020 • 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. The following sections introduce Azure Blockchain Service development for implementing enterprise blockchain solutions.

Connecting 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.

Visual Studio Code

You can connect to consortium members using the Azure Blockchain Development Kit Visual Studio Code extension. Once connected to a consortium, you can compile, build, and deploy smart contracts to an Azure Blockchain Service consortium member.

To develop sophisticated enterprise blockchain solutions, a development framework is needed to connect to different blockchain networks and manage smart contract lifecycles. Most projects interact with at least two blockchain nodes. Developers use a local blockchain during development. When the application is ready for test or release, the developer deploys to a blockchain network. For example, the main public Ethereum network or Azure Blockchain Service. Azure Blockchain Development Kit for Ethereum extension in Visual Studio Code uses Truffle. 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.

For more information, see Quickstart: Use Visual Studio Code to connect to an Azure Blockchain Service consortium network.

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.

For more information, see Quickstart: Use MetaMask to connect and deploy a smart contract

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.

For more information, see Quickstart: Use Geth to attach to an Azure Blockchain Service transaction node.

Ethereum Quorum private transactions

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** - proof-of-work or proof-of-stake consensus is not needed for a permissioned network. Quorum offers multiple consensus mechanisms that are designed for consortium chains such as RAFT and IBFT. Azure Blockchain Service uses the IBFT consensus mechanism.
- **Peer permissioning** - node and peer permissioning using smart contracts ensures only known parties can join the network.
- **Higher Performance** - Quorum offers higher performance than public Geth.

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. The following block explorers work with Azure Blockchain Service:

- Epirus Azure Blockchain Service Explorer from Web3 Labs
- BlockScout

You can also build your own block explorer by using Blockchain Data Manager and Azure Cosmos DB, see Tutorial: Use Blockchain Data Manager to send data to Azure Cosmos DB.

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

Try a quickstart using Azure Blockchain Development Kit for Ethereum to attach to a consortium on Azure Blockchain Service.

Use Visual Studio Code to connect to Azure Blockchain Service

Limits in Azure Blockchain Service

4/6/2020 • 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

Your consortium network should have a least two Azure Blockchain Service standard tier nodes. Standard tier nodes include two validator nodes. Four validator nodes are required to meet Istanbul Byzantine Fault Tolerance consensus.

Use the basic tier is for development, testing, and proof of concepts. Use the standard tier for production grade deployments. You should also use the *Standard* tier if you are using Blockchain Data Manager or sending a high volume of private transactions.

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 1.8 terabytes.

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 resources 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.

Performance

Do not use *eth.estimate* gas function for each transaction submission. The *eth.estimate* function is memory intensive. Calling the function multiple times reduces transactions per second drastically.

If possible, use a conservative gas value for submitting transactions and minimize the use of *eth.estimate*.

Next steps

Learn more about policies regarding systems patching and upgrades - Patching, updates, and versions.

Supported Azure Blockchain Service ledger versions

11/2/2020 • 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 2.6.0 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. For example, if you are running version 2.x, an upgrade to version 3.x is currently not supported. Similarly, if you are running version 2.2, an upgrade to version 2.3 is currently not supported.

How to check Quorum ledger version

You can check the Quorum version on your Azure Blockchain Service member by attaching to your node using geth or viewing diagnostic logs.

Using geth

Attach to your Azure Blockchain Service node using geth. For example,
`geth attach https://myblockchainmember.blockchain.azure.com:3200/<Access key>`.

Once your node is connected, geth reports the Quorum version similar to the following output:

```
instance: Geth/v1.9.7-stable-9339be03(quorum-v2.6.0)/linux-amd64/go1.13.12
```

For more information on using geth, see Quickstart: Use Geth to attach to an Azure Blockchain Service transaction node.

Using diagnostic logs

If you enable diagnostic logs, the Quorum version is reported for transaction nodes. For example, the following node informational log message includes the Quorum version.

```
{"NodeName":"transaction-node","Message":"INFO [06-22|05:31:45.156] Starting peer-to-peer node
instance=Geth/v1.9.7-stable-9339be03(quorum-v2.6.0)/linux-amd64/go1.13.12\n"}
{"NodeName":"transaction-node","Message":"[*] Starting Quorum node with QUORUM_VERSION=2.6.0,
TESSERA_VERSION=0.10.5 and PRIVATE_CONFIG=/working-dir/c/tm.ipc\n"}
111
```

For more information on diagnostic logs, see Monitor Azure Blockchain Service through Azure Monitor.

How to check genesis file content

To check genesis file content of your blockchain node, you can use the following Ethereum JavaScript API:

```
admin.nodeInfo.protocols
```

You can call the API using a geth console or a web3 library. For more information on using geth, see Quickstart: Use Geth to attach to an Azure Blockchain Service transaction node.

Next steps

Limits in Azure Blockchain Service

How to configure Azure Active Directory access for Azure Blockchain Service

5/11/2021 • 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.

> **IMPORTANT**
>
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

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

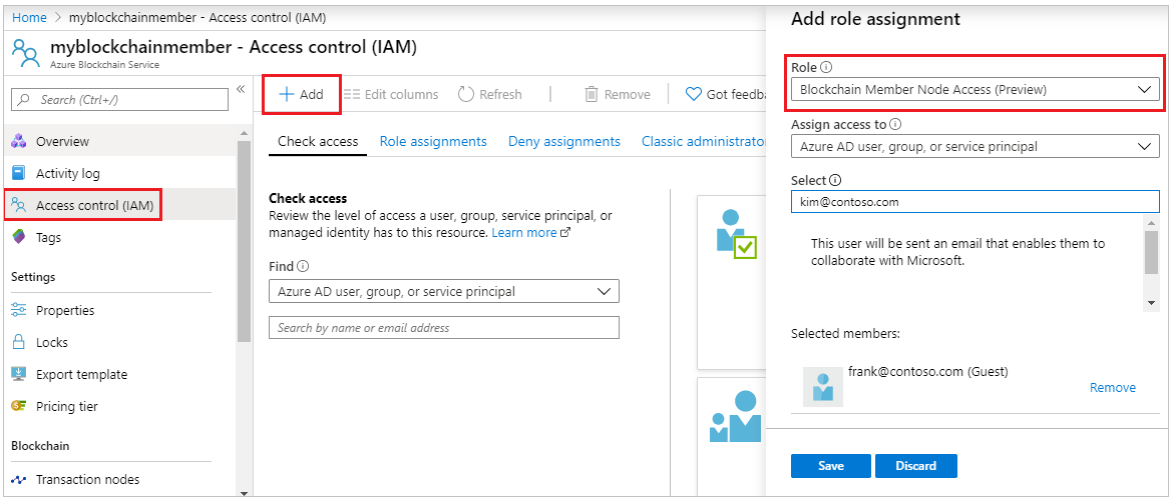
You can grant access at both the member level and the node level. Granting access rights at the member level will in turn grant access to all nodes under the member.

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	`kim@contoso.onmicrosoft.com`
Azure AD group	`sales@contoso.onmicrosoft.com`
Application ID	`13925ab1-4161-4534-8d18-812f5ca1ab1e`



4. Select **Save**.

Grant node level access

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.

Select the Blockchain Member Node Access (Preview) role and add the Azure AD ID object you wish to grant access to.

For more information, see Configure Azure Blockchain Service transaction nodes.

Connect using Azure Blockchain Connector

Download or clone the Azure Blockchain Connector from GitHub.

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

The 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, `yourdomain.onmicrosoft.com`
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 Blockchain Data Manager using the Azure portal

5/11/2021 • 6 minutes to read • Edit Online

Configure Blockchain Data Manager for Azure Blockchain Service to capture blockchain data and send it to an Azure Event Grid Topic.

> **IMPORTANT**
>
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

To configure a Blockchain Data Manager instance, you:

- Create a Blockchain Data Manager instance for an Azure Blockchain Service transaction node
- Add your blockchain applications

Prerequisites

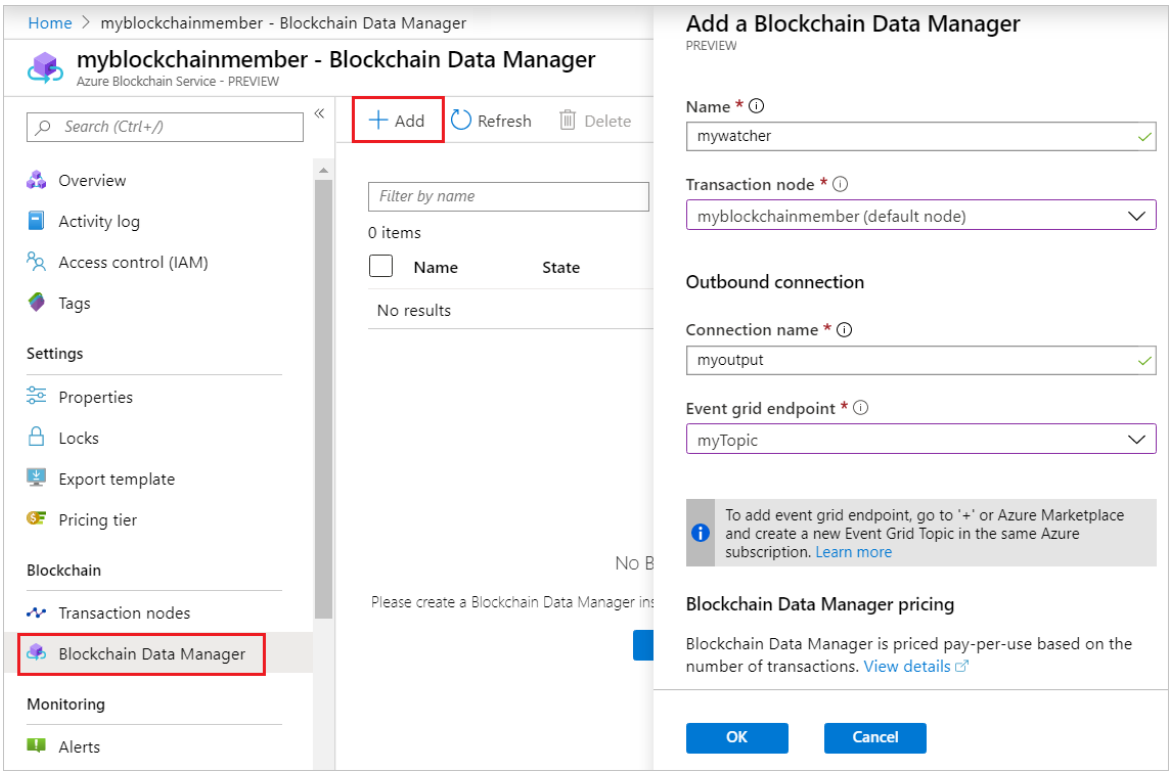
- Complete Quickstart: Create a blockchain member using the Azure portal or Quickstart: Create an Azure Blockchain Service blockchain member using Azure CLI. Azure Blockchain Service *Standard* tier is recommended when using Blockchain Data Manager.
- Create an Event Grid Topic
- Learn about Event handlers in Azure Event Grid

Create instance

A Blockchain Data Manager instance connects and monitors an Azure Blockchain Service transaction node. Only users with access to the transaction node can create a connection. An instance captures all raw block and raw transaction data from the transaction node. Blockchain Data Manager publishes a **RawBlockAndTransactionMsg** message which is a superset of information returned from web3.eth getBlock and getTransaction queries.

An outbound connection sends blockchain data to Azure Event Grid. You configure a single outbound connection when you create the instance. Blockchain Data Manager supports multiple Event Grid Topic outbound connections for any given Blockchain Data Manager instance. You can send blockchain data to a single destination or send blockchain data to multiple destinations. To add another destination, just add additional outbound connections to the instance.

1. Sign in to the Azure portal.
2. Go to the Azure Blockchain Service member you want to connect to Blockchain Data Manager. Select **Blockchain Data Manager**.
3. Select **Add**.



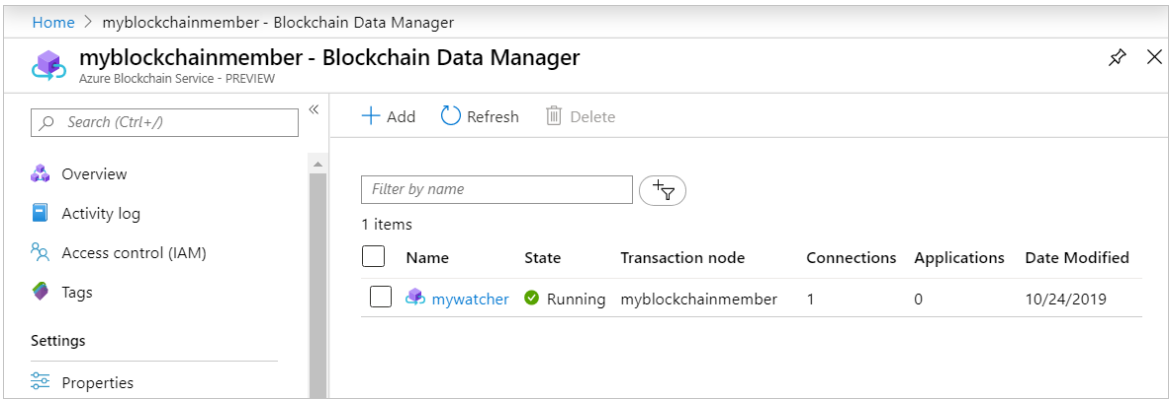
Enter the following details:

SETTING	DESCRIPTION
Name	Enter a unique name for a connected Blockchain Data Manager. The Blockchain Data Manager name can contain lower case letters and numbers and has a maximum length of 20 characters.
Transaction node	Choose a transaction node. Only transaction nodes you have read access are listed.
Connection name	Enter a unique name of the outbound connection where blockchain transaction data is sent.
Event grid endpoint	Choose an event grid topic in the same subscription as the Blockchain Data Manager instance.

4. Select **OK**.

   It takes less than a minute to create a Blockchain Data Manager instance. After the instance is deployed, it is automatically started. A running Blockchain Data Manager instance captures blockchain events from the transaction node and sends data to the outbound connections.

   The new instance appears in the list of Blockchain Data Manager instances for the Azure Blockchain Service member.



Add blockchain application

If you add a blockchain application, Blockchain Data Manager decodes event and property state for the application. Otherwise, only raw block and raw transaction data is sent. Blockchain Data Manager also discovers contract addresses when the contract is deployed. You can add multiple blockchain applications to a Blockchain Data Manager instance.

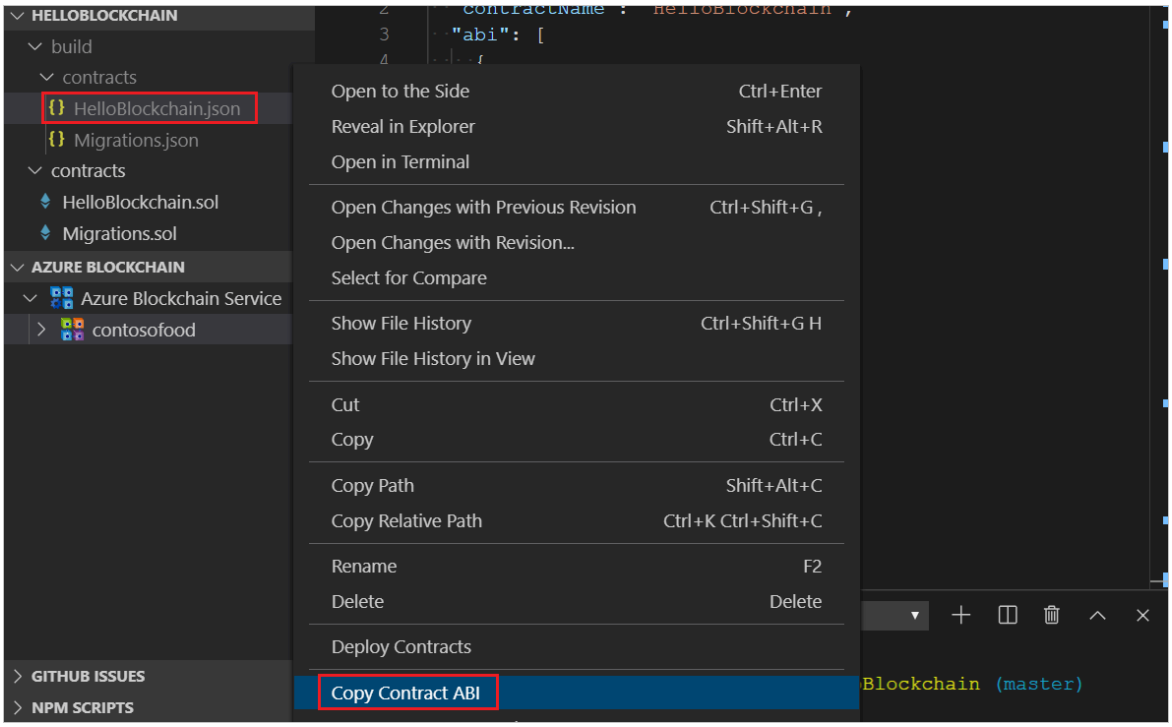
> **IMPORTANT**
> Currently, blockchain applications that declare Solidity array types or mapping types are not fully supported. Properties declared as array or mapping types will not be decoded in *ContractPropertiesMsg* or *DecodedContractEventsMsg* messages.

Blockchain Data Manager requires a smart contract ABI and deployed bytecode file to add the application.

Get Contract ABI and bytecode

The contract ABI defines the smart contract interfaces. It describes how to interact with the smart contract. You can use the Azure Blockchain Development Kit for Ethereum extension to copy the contract ABI to the clipboard.

1. In the Visual Studio Code explorer pane, expand the **build/contracts** folder of your Solidity project.
2. Right-click the contract metadata JSON file. The file name is the smart contract name followed by the **.json** extension.
3. Select **Copy Contract ABI**.

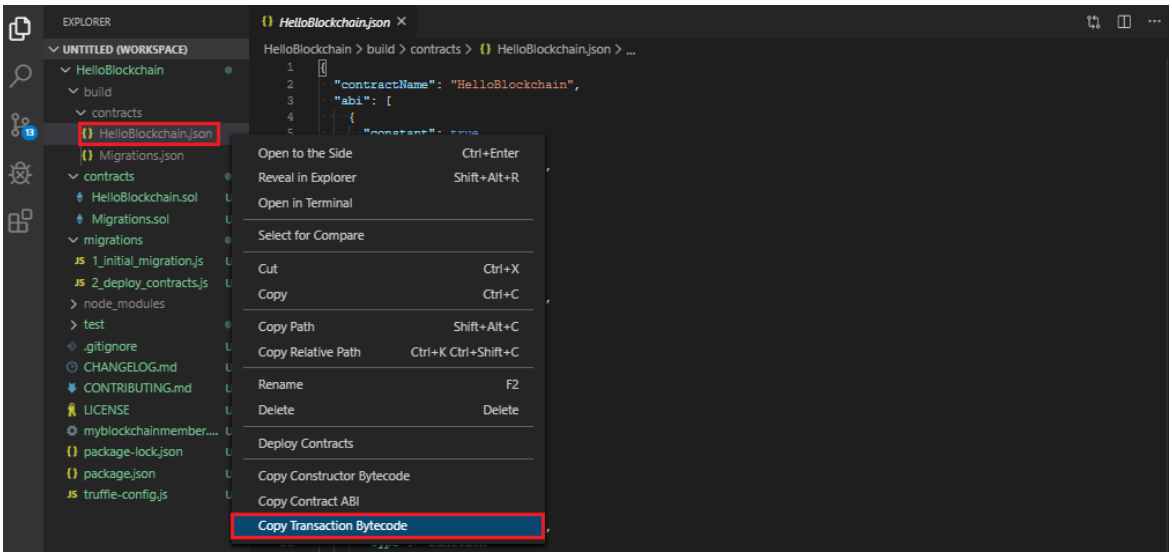


The contract ABI is copied to the clipboard.

4. Save the **abi** array as a JSON file. For example, *abi.json*. You use the file in a later step.

Blockchain Data Manager requires the deployed bytecode for the smart contract. The deployed bytecode is different than the smart contract bytecode. You use the Azure blockchain development kit extension to copy the bytecode to the clipboard.

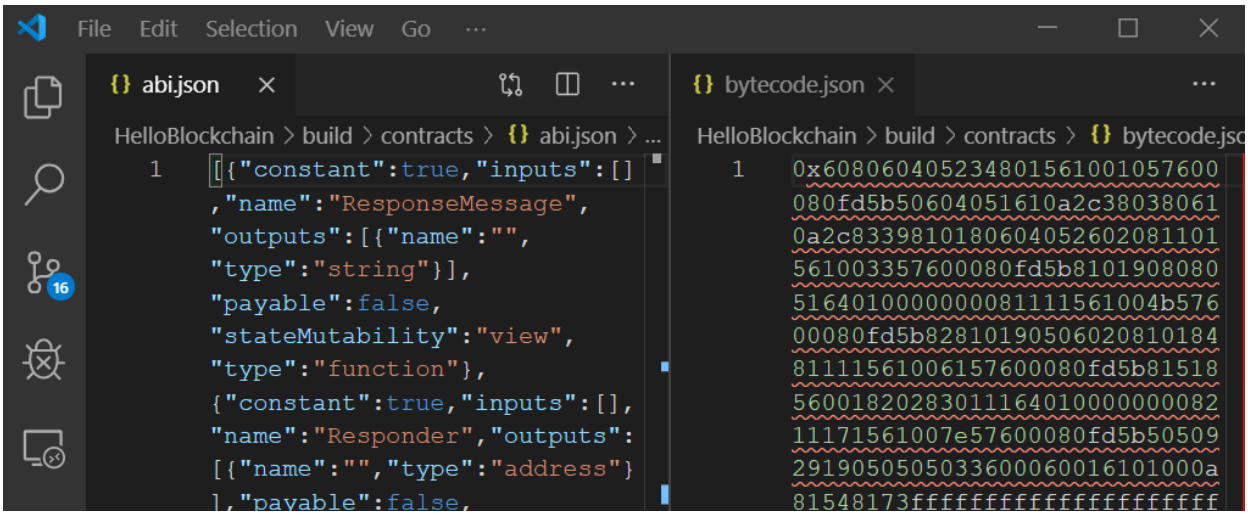
1. In the Visual Studio Code explorer pane, expand the **build/contracts** folder of your Solidity project.
2. Right-click the contract metadata JSON file. The file name is the smart contract name followed by the **.json** extension.
3. Select **Copy Transaction Bytecode**.



The bytecode is copied to the clipboard.

4. Save the **bytecode** value as a JSON file. For example, *bytecode.json*. You use the file in a later step.

The following example shows *abi.json* and *bytecode.json* files open in the VS Code editor. Your files should look similar.



Create contract ABI and bytecode URL

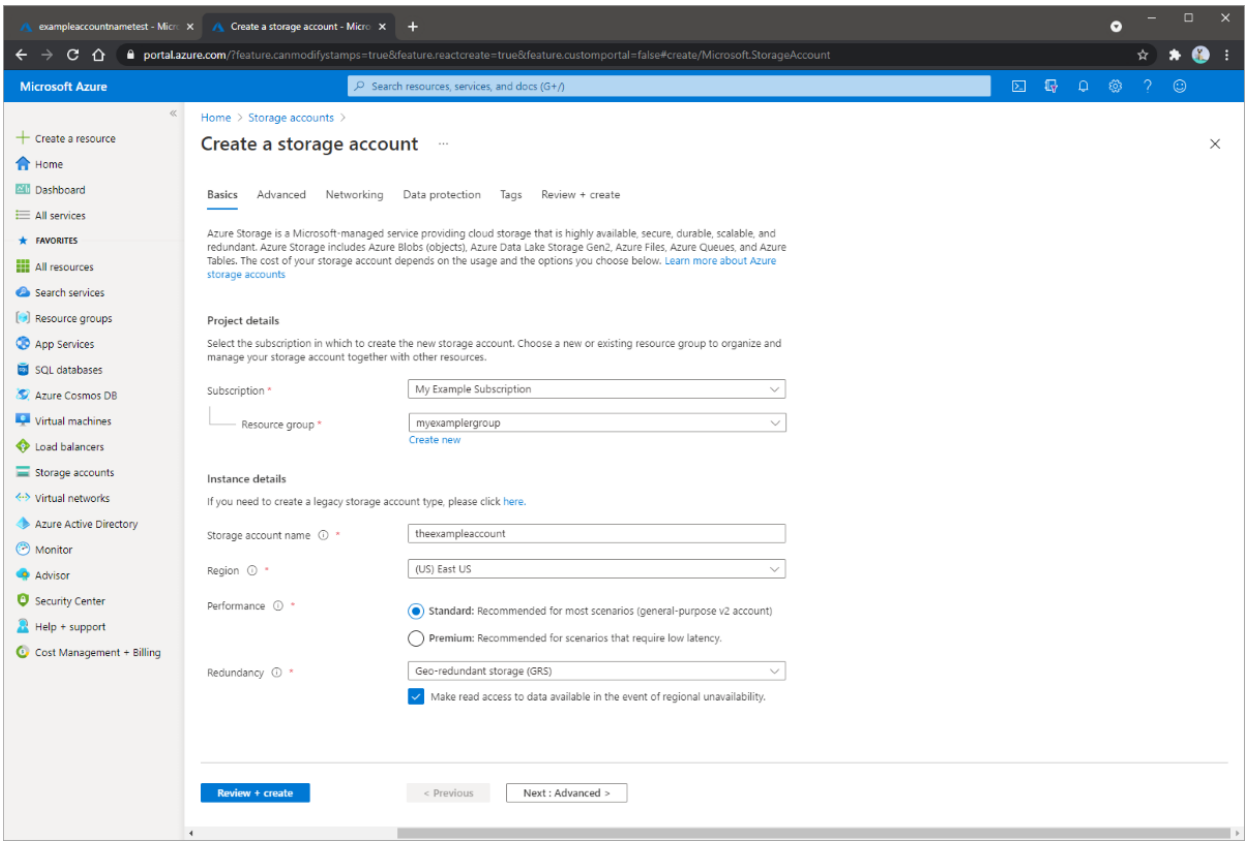
Blockchain Data Manager requires the contract ABI and bytecode files to be accessible by a URL when adding an application. You can use an Azure Storage account to provide a privately accessible URL.

Create storage account

To create a general-purpose v2 storage account in the Azure portal, follow these steps:

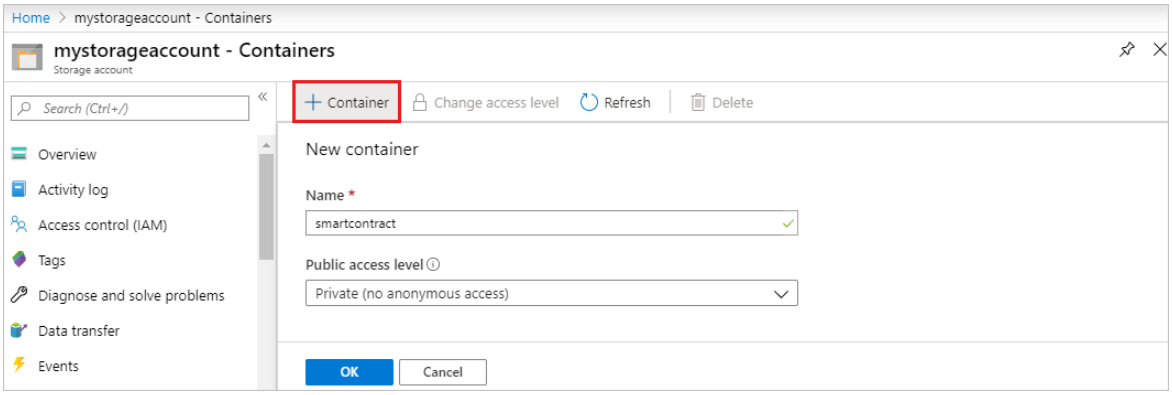
1. On the Azure portal menu, select **All services**. In the list of resources, type **Storage Accounts**. As you begin typing, the list filters based on your input. Select **Storage Accounts**.
2. On the **Storage Accounts** window that appears, choose **+ New**.
3. On the **Basics** blade, select the subscription in which to create the storage account.
4. Under the **Resource group** field, select your desired resource group, or create a new resource group. For more information on Azure resource groups, see Azure Resource Manager overview.
5. Next, enter a name for your storage account. The name you choose must be unique across Azure. The name also must be between 3 and 24 characters in length, and may include only numbers and lowercase letters.
6. Select a region for your storage account, or use the default region.
7. Select a performance tier. The default tier is *Standard*.
8. Specify how the storage account will be replicated. The default redundancy option is *Geo-redundant storage (GRS)*. For more information about available replication options, see Azure Storage redundancy.
9. Additional options are available on the **Advanced**, **Networking**, **Data protection**, and **Tags** blades. To use Azure Data Lake Storage, choose the **Advanced** blade, and then set **Hierarchical namespace** to **Enabled**. For more information, see Azure Data Lake Storage Gen2 Introduction
10. Select **Review + Create** to review your storage account settings and create the account.
11. Select **Create**.

The following image shows the settings on the **Basics** blade for a new storage account:



Upload contract files

1. Create a new container for the storage account. Select **Containers** > **Container**.

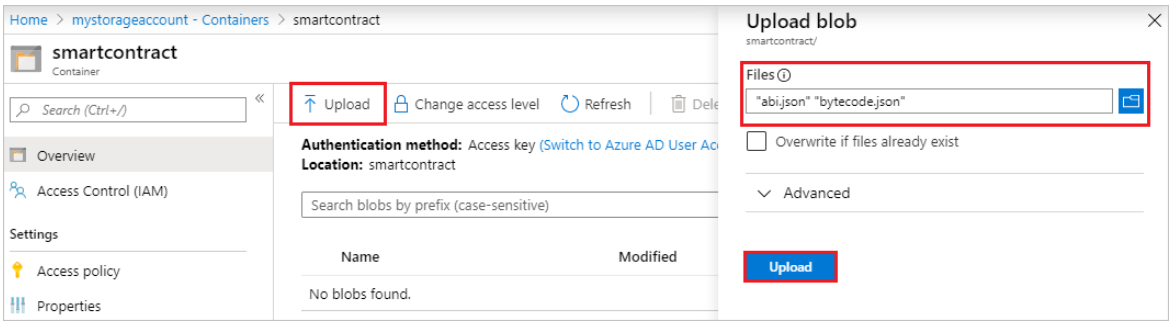


FIELD	DESCRIPTION
Name	Name the container. For example, *smartcontract*
Public access level	Choose *Private (no anonymous access)*

2. Select **OK** to create the container.

3. Select the container then select **Upload**.

4. Choose both JSON files you created in the Get Contract ABI and bytecode section.

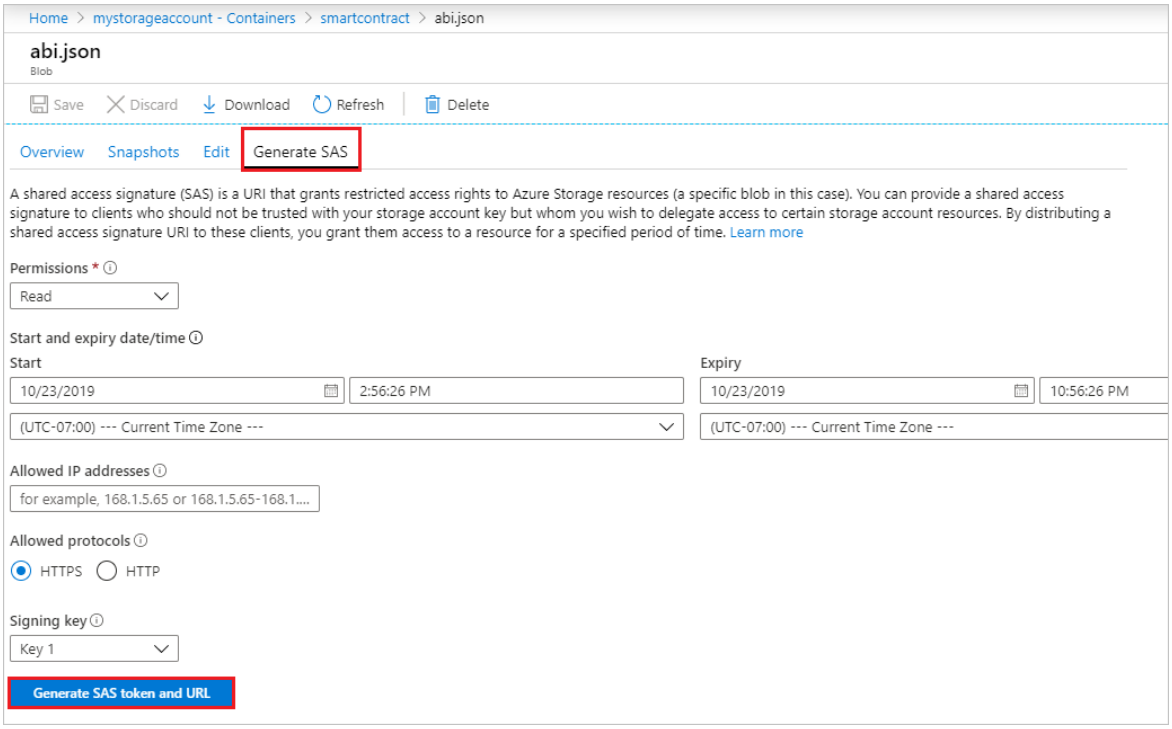


Select **Upload**.

Generate URL

For each blob, generate a shared access signature.

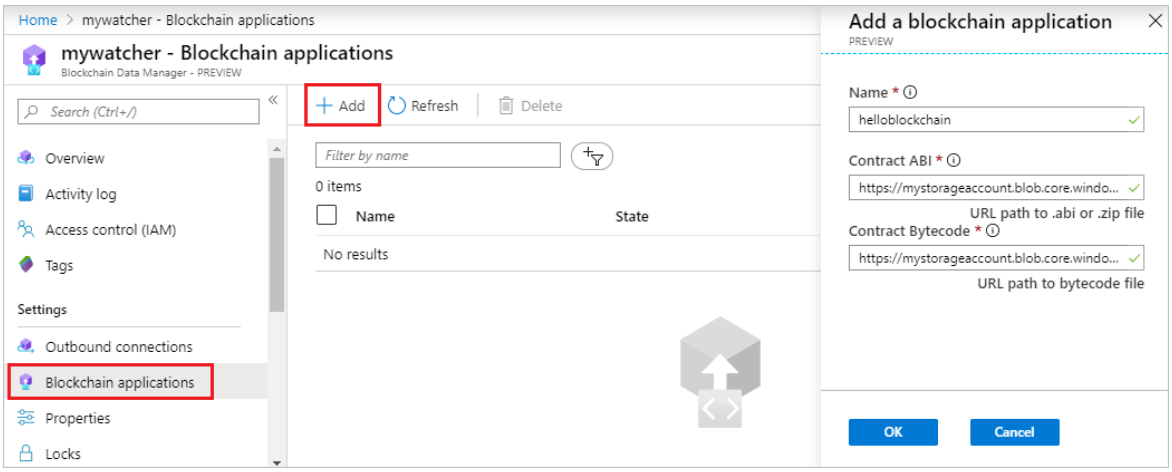
1. Select the ABI JSON blob.
2. Select **Generate SAS**
3. Set desired access signature expiration then select **Generate blob SAS token and URL**.



4. Copy the **Blob SAS URL** and save it for the next section.
5. Repeat the Generate URL steps for the bytecode JSON blob.

Add application to instance

1. Select your Blockchain Data Manager instance from the instance list.
2. Select **Blockchain applications**.
3. Select **Add**.

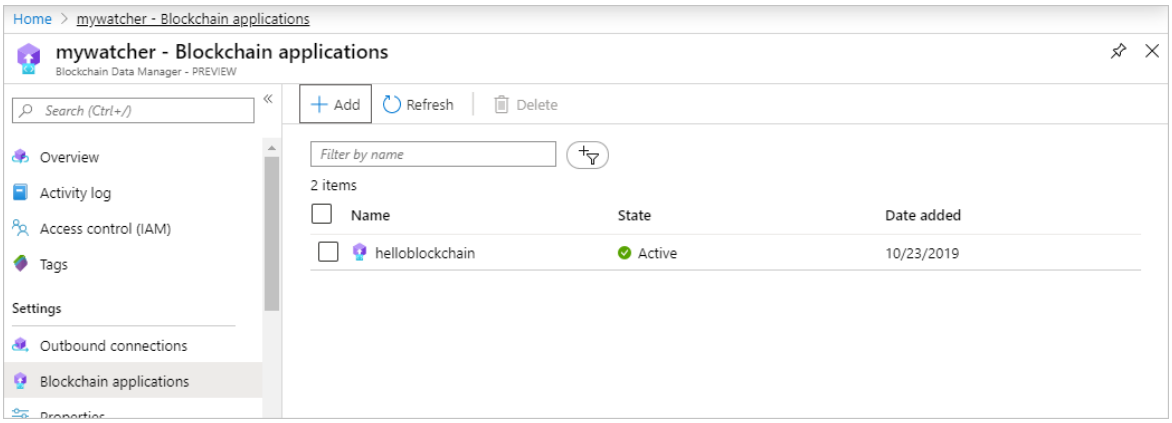


Enter the name of the blockchain application and the smart contract ABI and bytecode URLs.

SETTING	DESCRIPTION
Name	Enter a unique name for the blockchain application to track.
Contract ABI	URL path to the Contract ABI file. For more information, see Create contract ABI and bytecode URL.
Contract Bytecode	URL path to bytecode file. For more information, see Create contract ABI and bytecode URL.

4. Select **OK**.

Once the application is created, the application appears in the list of blockchain applications.

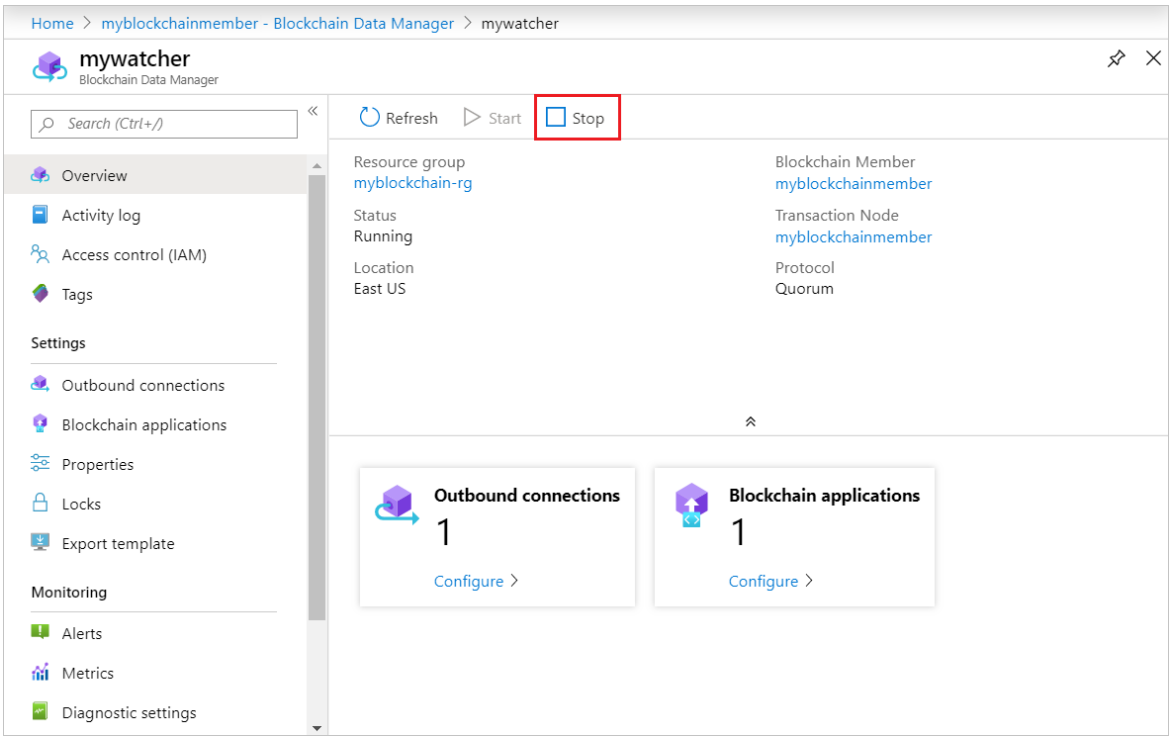


You can delete the Azure Storage account or use it to configure more blockchain applications. If you wish to delete the Azure Storage account, you can delete the resource group. Deleting the resource group also deletes the associated storage account, and any other resources associated with the resource group.

Stop instance

Stop the Blockchain Manager instance when you want to stop capturing blockchain events and sending data to the outbound connections. When the instance is stopped, no charges are incurred for Blockchain Data Manager. For more information, see pricing.

1. Go to **Overview** and select **Stop**.



Next steps

Try the next tutorial creating a blockchain transaction message explorer using Blockchain Data Manager and Azure Cosmos DB.

Use Blockchain Data Manager to send data to Azure Cosmos DB

Configure Blockchain Data Manager using Azure CLI

5/11/2021 • 9 minutes to read • Edit Online

Configure Blockchain Data Manager for Azure Blockchain Service to capture blockchain data send it to an Azure Event Grid Topic.

> **IMPORTANT**
>
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

To configure a Blockchain Data Manager instance, you:

- Create a Blockchain Manager instance
- Create an input to an Azure Blockchain Service transaction node
- Create an output to an Azure Event Grid Topic
- Add a blockchain application
- Start an instance

Prerequisites

- Install the latest Azure CLI and signed in using `az login`.
- Complete Quickstart: Use Visual Studio Code to connect to a Azure Blockchain Service consortium network. Azure Blockchain Service *Standard* tier is recommended when using Blockchain Data Manager.
- Create an Event Grid Topic
- Learn about Event handlers in Azure Event Grid

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 myRG --location eastus
```

Create instance

A Blockchain Data Manager instance monitors an Azure Blockchain Service transaction node. An instance captures all raw block and raw transaction data from the transaction node. Blockchain Data Manager publishes a **RawBlockAndTransactionMsg** message which is a superset of information returned from web3.eth getBlock and getTransaction queries.

```
az resource create \
                    --resource-group <Resource group> \
                    --name <Blockchain Data Manager instance name> \
                    --resource-type Microsoft.blockchain/watchers \
                    --is-full-object \
                    --properties <watcher resource properties>
```

PARAMETER	DESCRIPTION
resource-group	Resource group name where to create the Blockchain Data Manager instance.
name	Name of the Blockchain Data Manager instance.
resource-type	The resource type for a Blockchain Data Manager instance is **Microsoft.blockchain/watchers**.
is-full-object	Indicates properties contain options for the watcher resource.
properties	JSON-formatted string containing properties for the watcher resource. Can be passed as a string or a file.

Create instance examples

JSON configuration example to create a Blockchain Manager instance in the **East US** region.

```
{
    "location": "eastus",
    "properties": {
    }
}
```

ELEMENT	DESCRIPTION
location	Region where to create the watcher resource
properties	Properties to set when creating the watcher resource

Create a Blockchain Data Manager instance named *mywatcher* using a JSON string for configuration.

```
az resource create \
                     --resource-group myRG \
                     --name mywatcher \
                     --resource-type Microsoft.blockchain/watchers \
                      --is-full-object \
                      --properties '{"location":"eastus"}'
```

Create a Blockchain Data Manager instance named *mywatcher* using a JSON configuration file.

```
az resource create \
                    --resource-group myRG \
                    --name mywatcher \
                    --resource-type Microsoft.blockchain/watchers \
                    --is-full-object \
                    --properties @watcher.json
```

Create input

An input connects Blockchain Data Manager to an Azure Blockchain Service transaction node. Only users with access to the transaction node can create a connection.

```
az resource create \
                   --resource-group <Resource group> \
                   --name <Input name> \
                   --namespace Microsoft.Blockchain \
                   --resource-type inputs \
                   --parent watchers/<Watcher name> \
                   --is-full-object \
                   --properties <input resource properties>
```

PARAMETER	DESCRIPTION
resource-group	Resource group name where to create the input resource.
name	Name of the input.
namespace	Use the **Microsoft.Blockchain** provider namespace.
resource-type	The resource type for a Blockchain Data Manager input is **inputs**.
parent	The path to the watcher to which the input is associated. For example, **watchers/mywatcher**.
is-full-object	Indicates properties contain options for the input resource.
properties	JSON-formatted string containing properties for the input resource. Can be passed as a string or a file.

Input examples

Configuration JSON example to create an input resource in the *East US* region that is connected to <Blockchain member>.

```
{
    "location": "eastus",
    "properties": {
        "inputType": "Ethereum",
        "dataSource": {
            "resourceId": "/subscriptions/<Subscription ID>/resourceGroups/<Resource
group>/providers/Microsoft.Blockchain/blockchainMembers/<Blockchain member>/transactionNodes/transaction-
node"
        }
    }
}
```

ELEMENT	DESCRIPTION
location	Region where to create the input resource.
inputType	Ledger type of the Azure Blockchain Service member. Currently, **Ethereum** is supported.
resourceId	Transaction node to which the input is connected. Replace <Subscription ID>, <Resource group>, and <Blockchain member> with the values for the transaction node resource. The input connects to the default transaction node for the Azure Blockchain Service member.

Create an input named *myInput* for *mywatcher* using a JSON string for configuration.

```
az resource create \
                    --resource-group myRG \
                    --name myInput \
                    --namespace Microsoft.Blockchain \
                    --resource-type inputs \
                    --parent watchers/mywatcher \
                    --is-full-object \
                    --properties '{"location":"eastus", "properties":{"inputType":"Ethereum","dataSource":
{"resourceId":"/subscriptions/<Subscription ID>/resourceGroups/<Resource
group>/providers/Microsoft.Blockchain/BlockchainMembers/<Blockchain member>/transactionNodes/transaction-
node"}}}'
```

Create an input named *myInput* for *mywatcher* using a JSON configuration file.

```
az resource create \
                    --resource-group myRG \
                    --name input \
                    --namespace Microsoft.Blockchain \ --resource-type inputs \
                    --parent watchers/mywatcher \
                    --is-full-object \
                    --properties @input.json
```

Create output

An outbound connection sends blockchain data to Azure Event Grid. You can send blockchain data to a single destination or send blockchain data to multiple destinations. Blockchain Data Manager supports multiple Event Grid Topic outbound connections for any given Blockchain Data Manager instance.

```
az resource create \
                    --resource-group <Resource group> \
                    --name <Output name> \
                    --namespace Microsoft.Blockchain \
                    --resource-type outputs \
                    --parent watchers/<Watcher name> \
                    --is-full-object \
                    --properties <output resource properties>
```

PARAMETER	DESCRIPTION
resource-group	Resource group name where to create the output resource.
name	Name of the output.
namespace	Use the **Microsoft.Blockchain** provider namespace.
resource-type	The resource type for a Blockchain Data Manager output is **outputs**.
parent	The path to the watcher to which the output is associated. For example, **watchers/mywatcher**.
is-full-object	Indicates properties contain options for the output resource.
properties	JSON-formatted string containing properties for the output resource. Can be passed as a string or a file.

Output examples

Configuration JSON example to create an output resource in the *East US* region that is connected to an event grid topic named <event grid topic>.

```
{
    "location": "eastus",
    "properties": {
        "outputType": "EventGrid",
        "dataSource": {
            "resourceId": "/subscriptions/<Subscription ID>/resourceGroups/<Resource
group>/providers/Microsoft.EventGrid/topics/<event grid topic>"
        }
    }
}
```

ELEMENT	DESCRIPTION
location	Region where to create the output resource.
outputType	Type of output. Currently, **EventGrid** is supported.
resourceId	Resource to which the output is connected. Replace <Subscription ID>, <Resource group>, and <Blockchain member> with the values for the event grid resource.

Create an output named *myoutput* for *mywatcher* that connects to an event grid topic using a JSON configuration string.

```
az resource create \
                    --resource-group myRG \
                    --name myoutput \
                    --namespace Microsoft.Blockchain \
                    --resource-type outputs \
                    --parent watchers/mywatcher \
                    --is-full-object \
                    --properties '{"location":"eastus","properties":{"outputType":"EventGrid","dataSource":
{"resourceId":"/subscriptions/<Subscription ID>/resourceGroups/<Resource
group>/providers/Microsoft.EventGrid/topics/<event grid topic>"}}}'
```

Create an output named *myoutput* for *mywatcher* that connects to an event grid topic using a JSON configuration file.

```
az resource create \
                    --resource-group myRG \
                    --name myoutput \
                    --namespace Microsoft.Blockchain \
                    --resource-type outputs \
                    --parent watchers/mywatcher \
                    --is-full-object \
                    --properties @output.json
```

Add blockchain application

If you add a blockchain application, Blockchain Data Manager decodes event and property state for the application. Otherwise, only raw block and raw transaction data is sent. Blockchain Data Manager also discovers contract addresses when the contract is deployed. You can add multiple blockchain applications to a Blockchain Data Manager instance.

> **IMPORTANT**
> Currently, blockchain applications that declare Solidity array types or mapping types are not fully supported. Properties declared as array or mapping types will not be decoded in *ContractPropertiesMsg* or *DecodedContractEventsMsg* messages.

```
az resource create \
                    --resource-group <Resource group> \
                    --name <Application name> \
                    --namespace Microsoft.Blockchain \
                    --resource-type artifacts \
                    --parent watchers/<Watcher name> \
                    --is-full-object \
                    --properties <Application resource properties>
```

PARAMETER	DESCRIPTION
resource-group	Resource group name where to create the application resource.
name	Name of the application.
namespace	Use the **Microsoft.Blockchain** provider namespace.

PARAMETER	DESCRIPTION
resource-type	The resource type for a Blockchain Data Manager application is **artifacts**.
parent	The path to the watcher to which the application is associated. For example, **watchers/mywatcher**.
is-full-object	Indicates properties contain options for the application resource.
properties	JSON-formatted string containing properties for the application resource. Can be passed as a string or a file.

Blockchain application examples

Configuration JSON example to create an application resource in the *East US* region that monitors a smart contract defined by the contract ABI and bytecode.

```
{
    "location": "eastus",
    "properties": {
        "artifactType": "EthereumSmartContract",
        "content": {
            "abiFileUrl": "<ABI URL>",
            "bytecodeFileUrl": "<Bytecode URL>",
            "queryTargetTypes": [
                "ContractProperties",
                "ContractEvents"
            ]
        }
    }
}
```

ELEMENT	DESCRIPTION
location	Region where to create the application resource.
artifactType	Type of application. Currently, **EthereumSmartContract** is supported.
abiFileUrl	URL for smart contract ABI JSON file. For more information on obtaining contract ABI and creating a URL, see Get Contract ABI and bytecode and Create contract ABI and bytecode URL.
bytecodeFileUrl	URL for smart contract deployed bytecode JSON file. For more information on obtaining the smart contract deployed bytecode and creating a URL, see Get Contract ABI and bytecode and Create contract ABI and bytecode URL. Note: Blockchain Data Manager requires the **deployed bytecode**.

ELEMENT	DESCRIPTION
queryTargetTypes	Published message types. Specifying **ContractProperties** publishes *ContractPropertiesMsg* message type. Specifying **ContractEvents** publishes *DecodedContractEventsMsg* message type. Note: *RawBlockAndTransactionMsg* and *RawTransactionContractCreationMsg* message types are always published.

Create an application named *myApplication* for *mywatcher* that monitors a smart contract defined by a JSON string.

```
az resource create \
                   --resource-group myRG \
                   --name myApplication \
                   --namespace Microsoft.Blockchain \
                   --resource-type artifacts \
                   --parent watchers/mywatcher \
                   --is-full-object \
                   --properties '{"location":"eastus","properties":
{"artifactType":"EthereumSmartContract","content":{"abiFileUrl":"<ABI URL>","bytecodeFileUrl":"<Bytecode
URL>","queryTargetTypes":["ContractProperties","ContractEvents"]}}}'
```

Create an application named *myApplication* for *mywatcher* that watches a smart contract defined using a JSON configuration file.

```
az resource create \
                   --resource-group myRG \
                   --name myApplication \
                   --namespace Microsoft.Blockchain \
                   --resource-type artifacts \
                   --parent watchers/mywatcher \
                   --is-full-object \
                   --properties @artifact.json
```

Start instance

When running, a Blockchain Manager instance monitors blockchain events from the defined inputs and sends data to the defined outputs.

```
az resource invoke-action \
                         --action start \
                         --ids /subscriptions/<Subscription ID>/resourceGroups/<Resource
group>/providers/Microsoft.Blockchain/watchers/<Watcher name>
```

PARAMETER	DESCRIPTION
action	Use **start** to run the watcher.
ids	Watcher resource ID. Replace <Subscription ID>, <Resource group>, and <Watcher name> with the values for the watcher resource.

Start instance example

Start a Blockchain Data Manager instance named *mywatcher*.

```
az resource invoke-action \
                          --action start \
                          --ids /subscriptions/<Subscription ID>/resourceGroups/<Resource
group>/providers/Microsoft.Blockchain/watchers/mywatcher
```

Stop instance

Stop a Blockchain Data Manager instance.

```
az resource invoke-action \
                          --action stop \
                          --ids /subscriptions/<Subscription ID>/resourceGroups/<Resource
group>/providers/Microsoft.Blockchain/watchers/<Watcher name>
```

PARAMETER	DESCRIPTION
action	Use **stop** to stop the watcher.
ids	Name of the watcher. Replace <Subscription ID>, <Resource group>, and <Watcher name> with the values for the watcher resource.

Stop watcher example

Stop an instance named *mywatcher*.

```
az resource invoke-action \
                          --action stop \
                          --ids /subscriptions/<Subscription ID>/resourceGroups/<Resource
group>/providers/Microsoft.Blockchain/watchers/mywatcher
```

Delete instance

Delete a Blockchain Data Manager instance.

```
az resource delete \
                    --resource-group <Resource group> \
                    --name <Watcher name> \
                    --resource-type Microsoft.Blockchain/watchers
```

PARAMETER	DESCRIPTION
resource-group	Resource group name of the watcher to delete.
name	Name of the watcher to delete.
resource-type	The resource type for a Blockchain Data Manager watcher is **Microsoft.blockchain/watchers**.

Delete instance example

Delete an instance named *mywatcher* in the *myRG* resource group.

```
az resource delete \
                    --resource-group myRG \
                    --name mywatcher \
                    --resource-type Microsoft.blockchain/watchers
```

Next steps

Try the next tutorial creating a blockchain transaction message explorer using Blockchain Data Manager and Azure Cosmos DB.

Use Blockchain Data Manager to send data to Azure Cosmos DB

Configure Azure Blockchain Service transaction nodes

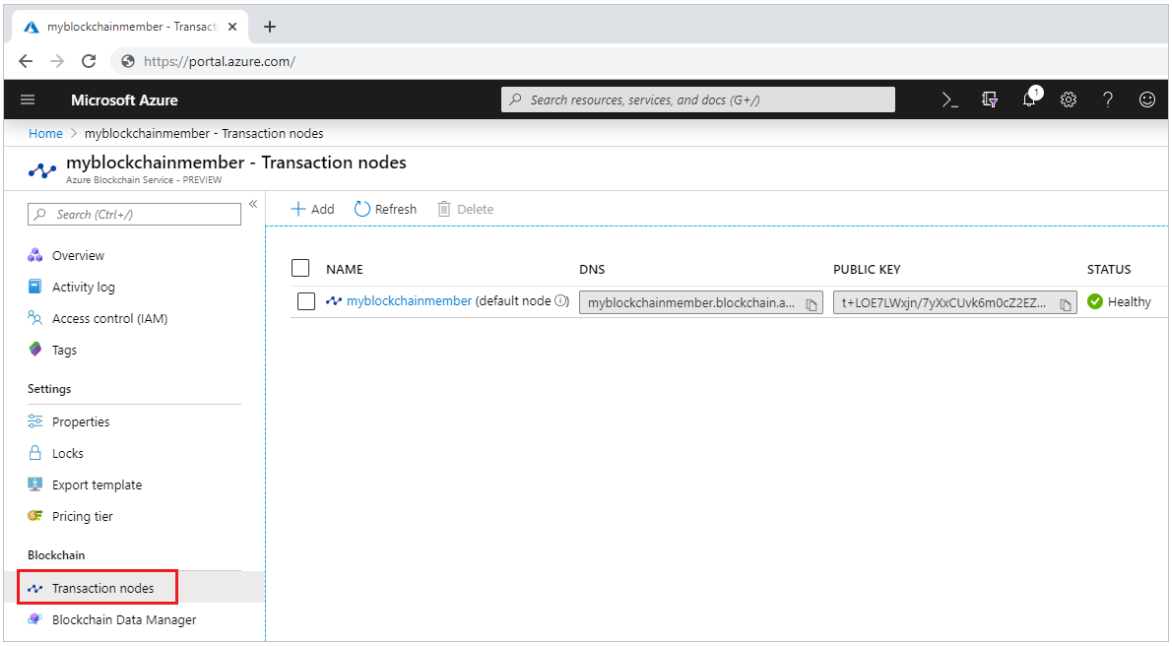
5/11/2021 • 4 minutes to read • Edit Online

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.

> **IMPORTANT**
>
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

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**.



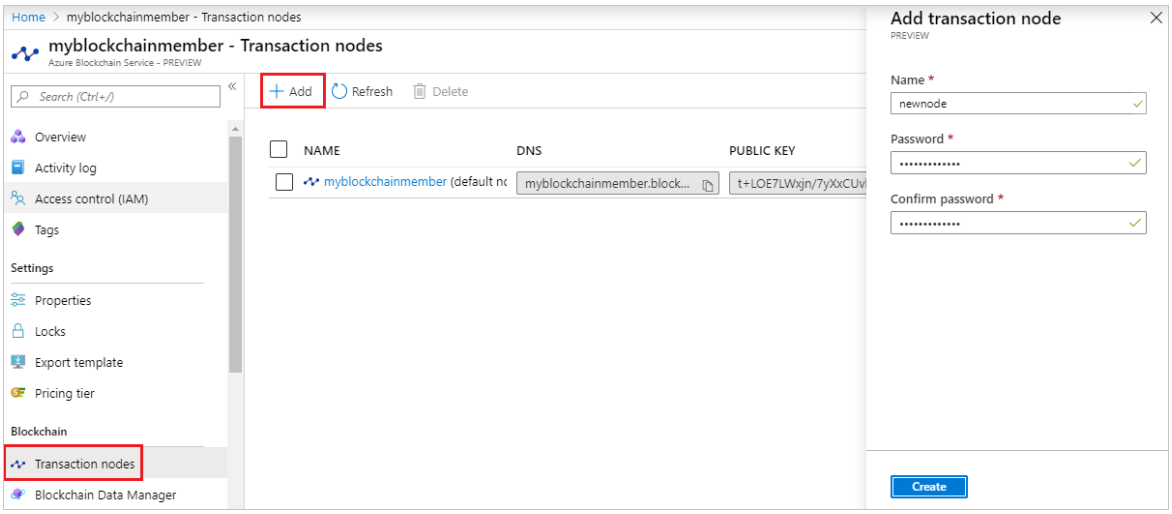
Overview details include public endpoint addresses and public key.

Create transaction node

You can add up to nine additional transaction nodes to your blockchain member, for a total of 10 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, `newnode-myblockchainmember.blockchain.azure.com`. 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**.

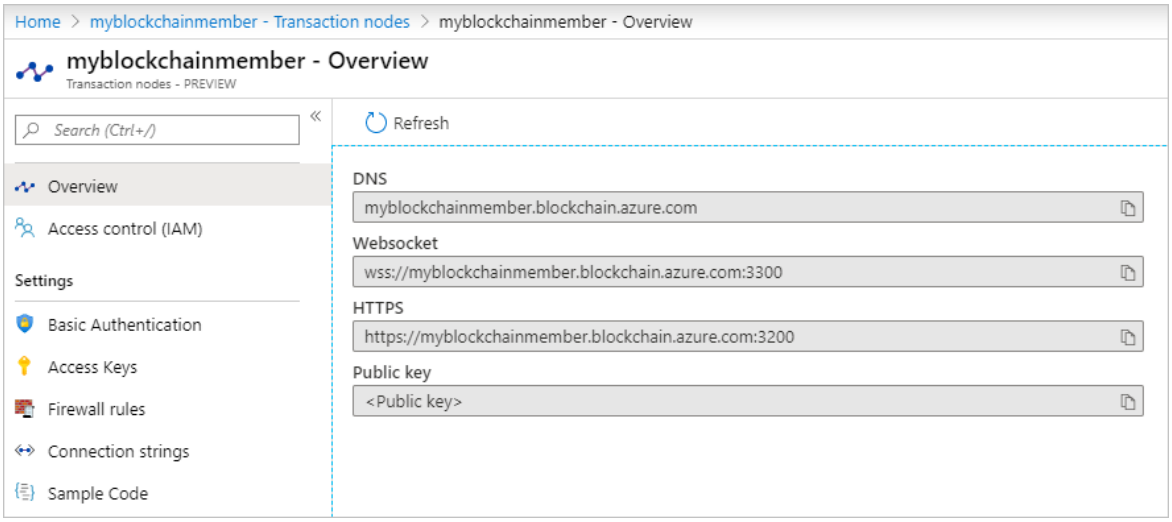
   Provisioning a new transaction node takes about 10 minutes. Additional transaction nodes incur cost. For more information on costs, see Azure pricing.

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 TLS.

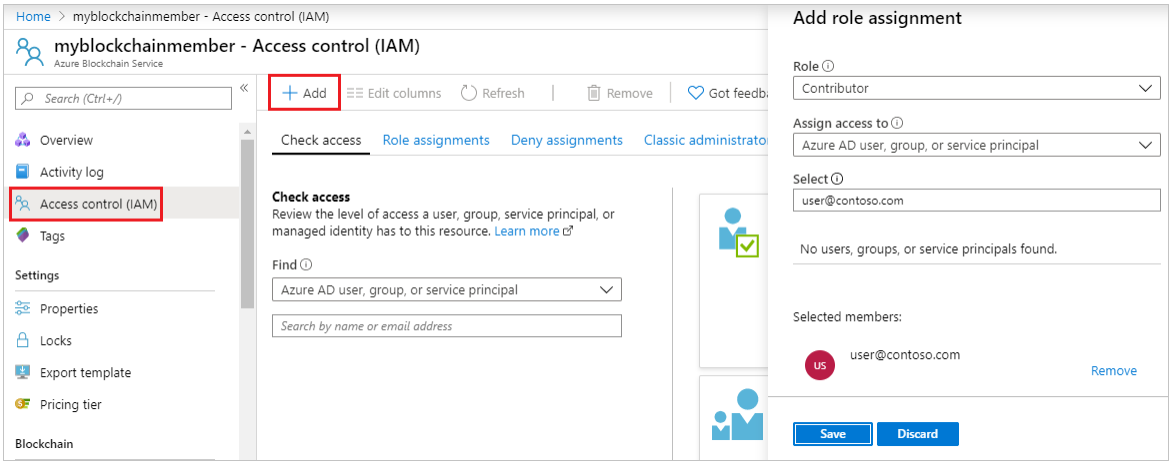
Azure Active Directory access control

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 **Owner**, **Contributor**, or **Reader**.
Assign access to	Select **Azure AD user, group, or service principal**.
Select	Search for the user, group, or service principal you want to add.

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

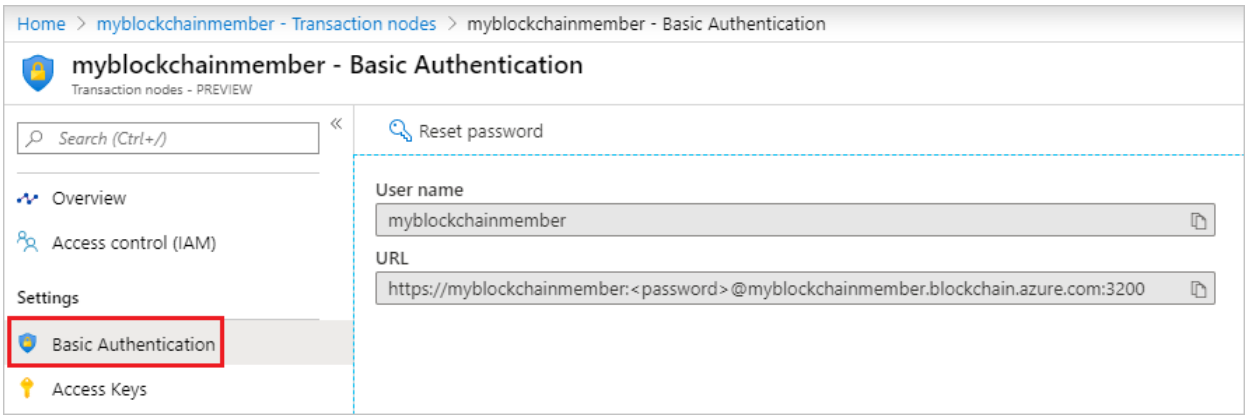
For more information on Azure AD access control, see Assign Azure roles using the Azure portal

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 to change and rotate keys.

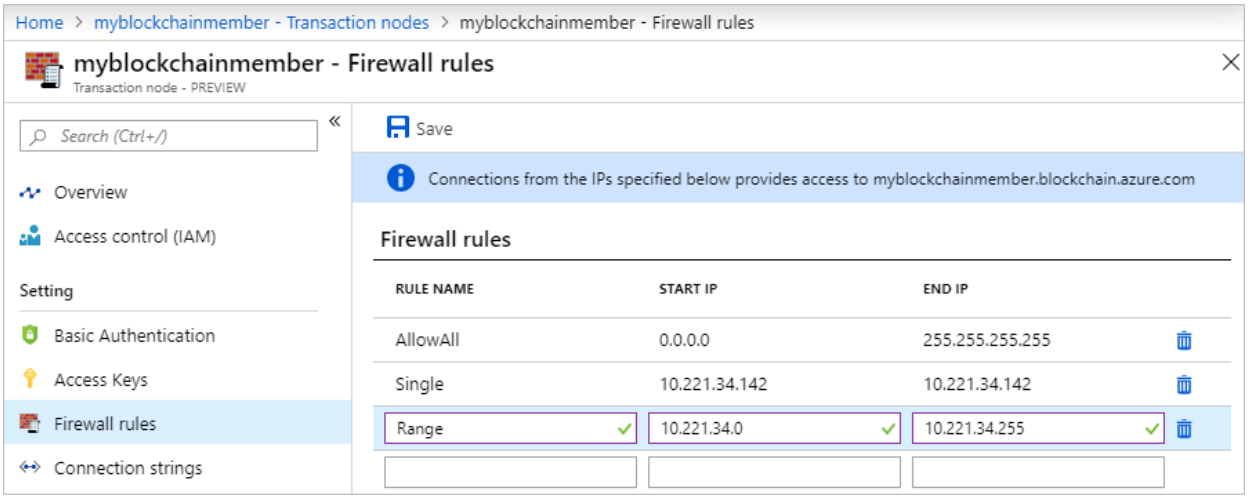
You can view a transaction node's access key details and copy endpoint addresses that include the access keys. Navigate to one of your Azure Blockchain Service member transaction nodes and select **Access Keys** in settings.

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.



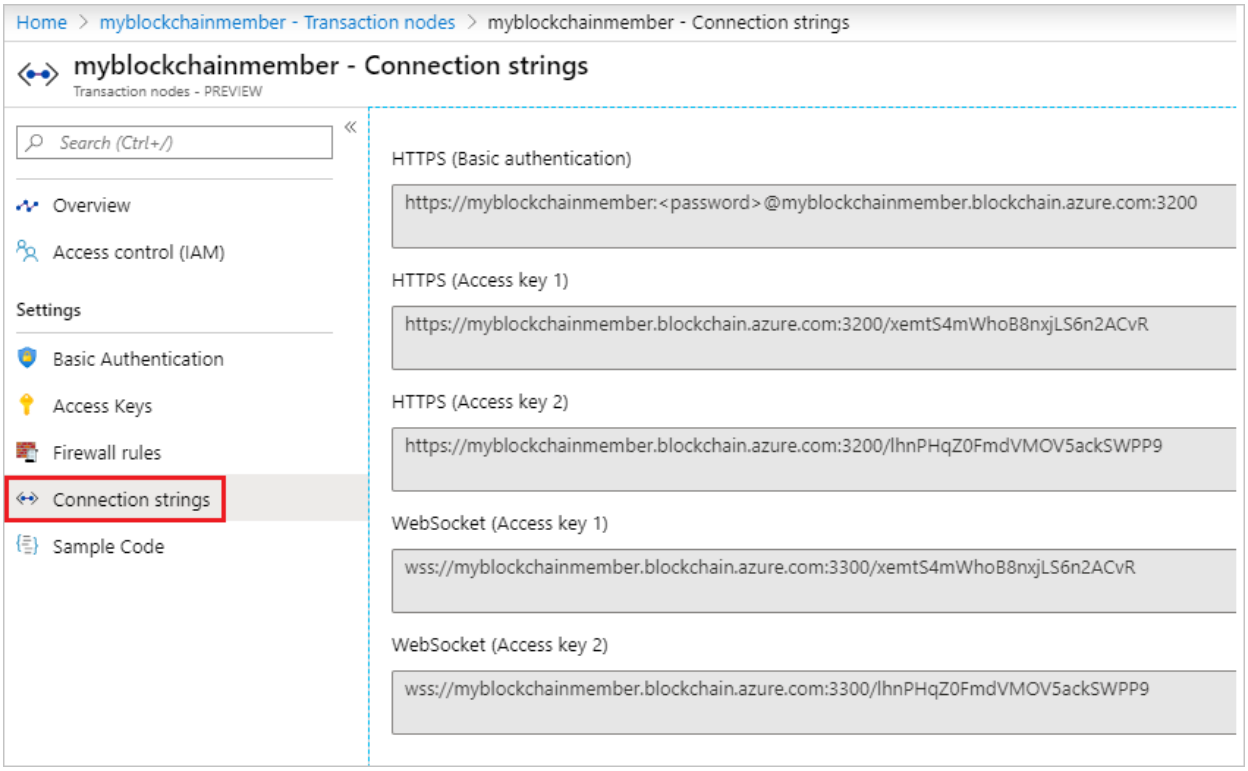
To enable:

- **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.

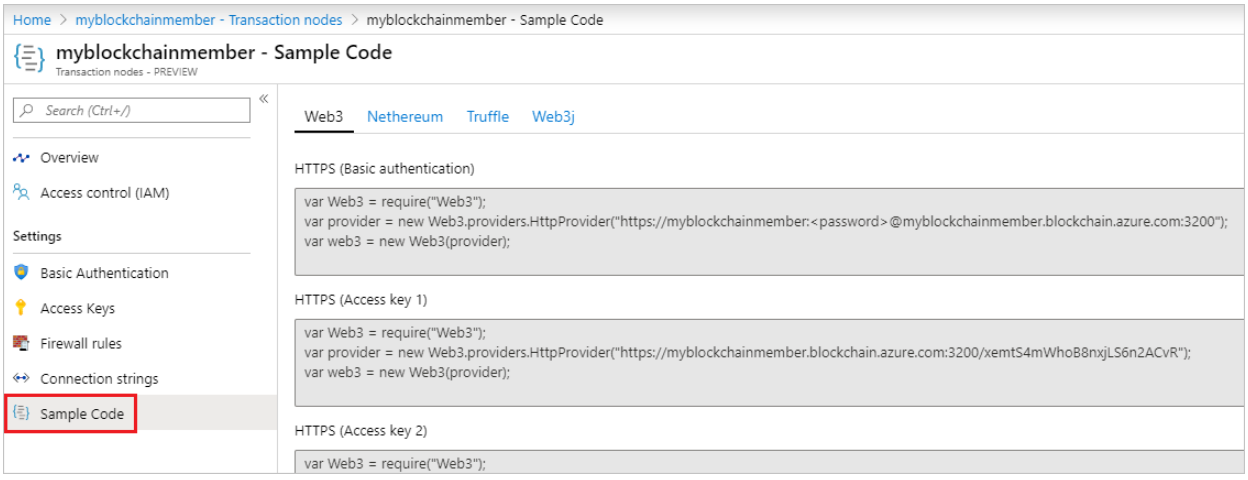


Sample code

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. Go to one of your Azure Blockchain Service member transaction nodes and select **Sample Code** in settings.

Choose the Web3, Nethereum, Truffle, or Web3j tab to view the code sample you want to use.



Next steps

Configure transaction nodes using Azure CLI

Manage Azure Blockchain Service using Azure CLI

5/11/2021 • 7 minutes to read • Edit Online

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

> **IMPORTANT**
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

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, see install Azure CLI.

Prepare your environment

1. Sign in.

   Sign in using the az login command if you're using a local install of the CLI.

   ```
   az login
   ```

   Follow the steps displayed in your terminal to complete the authentication process.

2. Install the Azure CLI extension.

   When working with extension references for the Azure CLI, you must first install the extension. Azure CLI extensions give you access to experimental and pre-release commands that have not yet shipped as part of the core CLI. To learn more about extensions including updating and uninstalling, see Use extensions with Azure CLI.

   Install the extension for Azure Blockchain Service by running the following command:

   ```
   az extension add --name blockchain
   ```

Create blockchain member

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

```
az blockchain member create \
                            --resource-group <myResourceGroup> \
                            --name <myMemberName> \
                            --location <myBlockchainLocation> \
                            --password <strongMemberAccountPassword> \
                            --protocol "Quorum" \
                            --consortium <myConsortiumName> \
                            --consortium-management-account-password <strongConsortiumManagementPassword> \
                            --sku <skuName>
```

PARAMETER	DESCRIPTION
resource-group	Resource group name where Azure Blockchain Service resources are created.
name	A unique name that identifies your Azure Blockchain Service blockchain member. The name is used for the public endpoint address. For example, `myblockchainmember.blockchain.azure.com`.
location	Azure region where the blockchain member is created. For example, `eastus`. Choose the location that is closest to your users or your other Azure applications. Features may not be available in some regions.
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. 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 semicolumn(;)
protocol	Blockchain protocol. Currently, *Quorum* protocol is supported.
consortium	Name of the consortium to join or create. For more information on consortia, see Azure Blockchain Service consortium.
consortium-management-account-password	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.
sku	Tier type. *Standard* or *Basic*. Use the *Basic* tier for development, testing, and proof of concepts. Use the *Standard* tier for production grade deployments. You should also use the *Standard* tier if you are using Blockchain Data Manager or sending a high volume of private transactions. Changing the pricing tier between basic and standard after member creation is not supported.

Change blockchain member passwords or firewall rules

Example updates a blockchain member's password, consortium management password, and firewall rule.

```
az blockchain member update \
                    --resource-group <myResourceGroup> \
                    --name <myMemberName> \
                    --password <strongMemberAccountPassword> \
                    --consortium-management-account-password <strongConsortiumManagementPassword> \
                    --firewall-rules <firewallRules>
```

PARAMETER	DESCRIPTION
resource-group	Resource group name where Azure Blockchain Service resources are created.
name	Name that identifies your Azure Blockchain Service member.
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. 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 semicolumn(;)
consortium-management-account-password	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.
firewall-rules	Start and end IP address for IP allowlist.

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 blockchain transaction-node create \
                    --resource-group <myResourceGroup> \
                    --member-name <myMemberName> \
                    --password <strongTransactionNodePassword> \
                    --name <myTransactionNodeName>
```

PARAMETER	DESCRIPTION
resource-group	Resource group name where Azure Blockchain Service resources are created.
location	Azure region of the blockchain member.
member-name	Name that identifies your Azure Blockchain Service member.

PARAMETER	DESCRIPTION
password	The password for the transaction node. Use the password for basic authentication when connecting to the transaction node public endpoint. 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 semicolumn(;)
name	Transaction node name.

Change transaction node password

Example updates a transaction node password.

```
az blockchain transaction-node update \
                    --resource-group <myResourceGroup> \
                    --member-name <myMemberName> \
                    --password <strongTransactionNodePassword> \
                    --name <myTransactionNodeName>
```

PARAMETER	DESCRIPTION
resource-group	Resource group name where Azure Blockchain Service resources exist.
member-name	Name that identifies your Azure Blockchain Service member.
password	The password for the transaction node. Use the password for basic authentication when connecting to the transaction node public endpoint. 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 semicolumn(;)
name	Transaction node name.

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 blockchain member list-api-key \
                           --resource-group <myResourceGroup> \
                           --name <myMemberName>
```

PARAMETER	DESCRIPTION
resource-group	Resource group name where Azure Blockchain Service resources exist.

PARAMETER	DESCRIPTION
name	Name of the Azure Blockchain Service blockchain member

Regenerate API keys

Use the following command to regenerate your API keys.

```
az blockchain member regenerate-api-key \
                            --resource-group <myResourceGroup> \
                            --name <myMemberName> \
                            [--key-name {<keyValue1>, <keyValue2>}]
```

PARAMETER	DESCRIPTION
resource-group	Resource group name where Azure Blockchain Service resources exist.
name	Name of the Azure Blockchain Service blockchain member.
keyName	Replace <keyValue> with either key1, key2, or both.

Delete a transaction node

Example deletes a blockchain member transaction node.

```
az blockchain transaction-node delete \
                      --resource-group <myResourceGroup> \
                      --member-name <myMemberName> \
                      --name <myTransactionNode>
```

PARAMETER	DESCRIPTION
resource-group	Resource group name where Azure Blockchain Service resources exist.
member-name	Name of the Azure Blockchain Service blockchain member that also includes the transaction node name to be deleted.
name	Transaction node name to delete.

Delete a blockchain member

Example deletes a blockchain member.

```
az blockchain member delete \
                      --resource-group <myResourceGroup> \
                      --name <myMemberName>
```

PARAMETER	DESCRIPTION
resource-group	Resource group name where Azure Blockchain Service resources exist.
name	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/<myMember
Name>
```

PARAMETER	DESCRIPTION
role	Name of the Azure AD role.
assignee	Azure AD user ID. For example, `user@contoso.com`
scope	Scope of the role assignment. Can be either a blockchain member or 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/blockchai
nMembers/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/blockchai
nMembers/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
role	Name of the Azure AD role.
assignee-object-id	Azure AD group ID or application ID.
scope	Scope of the role assignment. Can be either a blockchain member or transaction node.

Example:

Grant node access for **application role**

```
az role assignment create \
                            --role 'myRole' \
                            --assignee-object-id 22222222-2222-2222-2222-222222222222 \
                            --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
role	Name of the Azure AD role.
assignee	Azure AD user ID. For example, `user@contoso.com`
scope	Scope of the role assignment. Can be either a blockchain member or transaction node.

Next steps

Learn how to Configure Azure Blockchain Service transaction nodes with the Azure portal.

Manage consortium members in Azure Blockchain Service using PowerShell

5/11/2021 • 6 minutes to read • Edit Online

You can use PowerShell to manage blockchain consortium members for your Azure Blockchain Service.

> **IMPORTANT**
>
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate 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. 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.

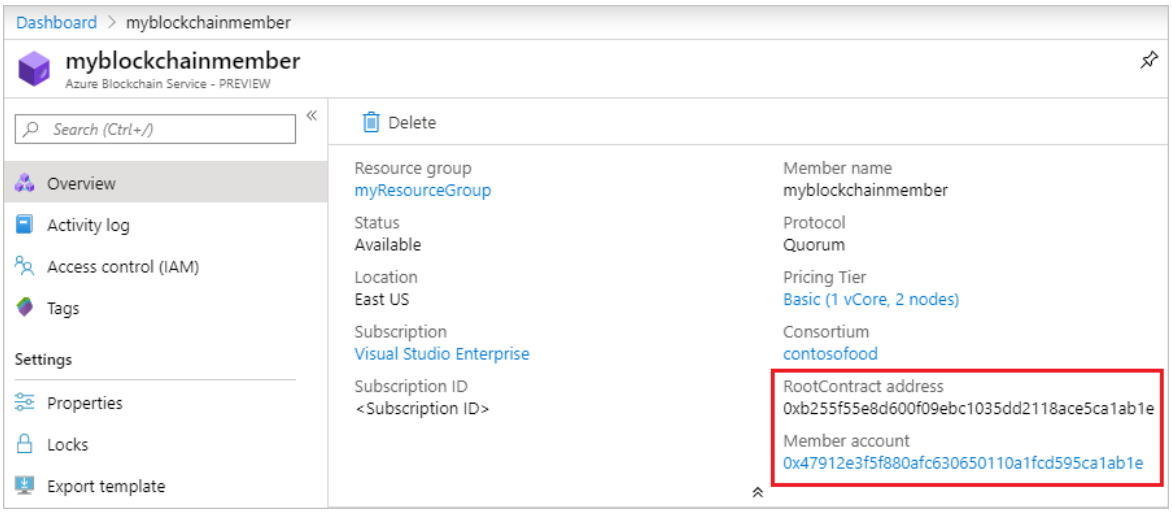
You can use this script to set global variables for calling the consortium management cmdlets.

```
$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.

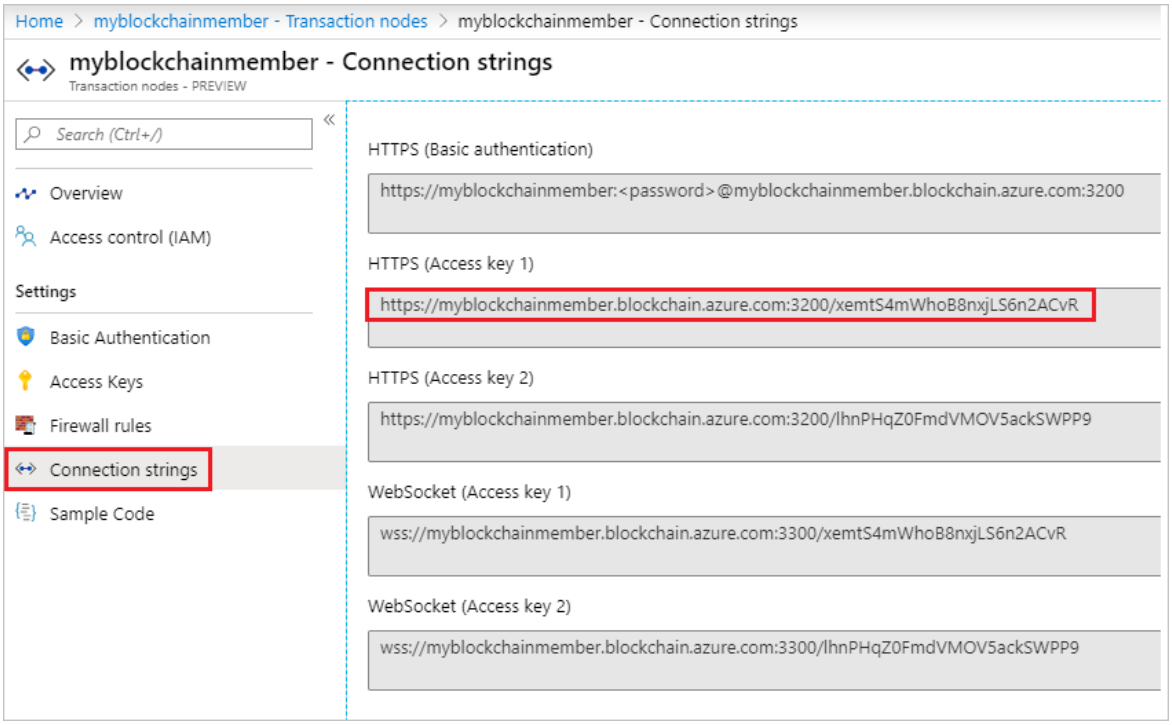
Find the other values in the Azure portal:

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**.



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

Establish a Web3 connection to set the $ContractConnection variable.

```
$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

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

Example

Establish a Web3 connection to set the $ContractConnection and $MemberAccount variables.

```
$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

Establish a Web3 connection to set the $ContractConnection and $MemberAccount variables.

```
$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
Web3Account	Web3Account object obtained from Import-Web3Account	Yes
Web3Client	Web3Client object obtained from New-Web3Connection	Yes

Example

Establish a Web3 connection to set the $ContractConnection and $MemberAccount variables.

```
$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

Establish a Web3 connection to set the $ContractConnection variable.

```
$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
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

Establish a Web3 connection to set the $ContractConnection and $MemberAccount variables.

```
$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 **USER** or **ADMIN**.	Yes
Members	Members object obtained from Import-ConsortiumManagementContracts	Yes
Web3Account	Web3Account object obtained from Import-Web3Account	Yes

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

Example

Establish a Web3 connection to set the $ContractConnection and $MemberAccount variables.

```
$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

Migrate Azure Blockchain Service

8/31/2021 • 9 minutes to read • Edit Online

You can migrate ledger data from Azure Blockchain Service to an alternate offering.

> **IMPORTANT**
> On **September 10, 2021**, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation.

Evaluate alternatives

The first step when planning a migration is to evaluate alternative offerings. Evaluate the following alternatives based on your development status of being in production or evaluation.

Production or pilot phase

If you have already deployed and developed a blockchain solution that is in the production or pilot phase, consider the following alternatives.

Quorum Blockchain Service

Quorum Blockchain Service is a managed offering by ConsenSys on Azure that supports Quorum as ledger technology.

- **Managed offering** - Quorum Blockchain Service has no extra management overhead compared to Azure Blockchain Service.
- **Ledger technology** - Based on ConsenSys Quorum which is an enhanced version of the GoQuorum Ledger technology used in Azure Blockchain Service. No new learning is required. For more information, see the Consensys Quorum FAQ.
- **Continuity** - You can migrate your existing data on to Quorum Blockchain Service by ConsenSys. For more information, see Export data from Azure Blockchain Service

For more information, see Quorum Blockchain Service.

Azure VM-based deployment

There are several blockchain resource management templates you can use to deploy blockchain on IaaS VMs.

- **Ledger technology** - You can continue to use Quorum ledger technology including the new ConsenSys Quorum.
- **Self-management** - Once deployed, you manage the infrastructure and blockchain stack.

New deployment or evaluation phase

If you are starting to develop a new solution or are in an evaluation phase, consider the following alternatives based on your scenario requirements.

- Quorum template from Azure Marketplace
- Besu template from Azure Marketplace

How to migrate to an alternative

To migrate a production workload, first export your data from Azure Blockchain Service. Once you have a copy of your data, you can transition this data to your preferred alternative.

The recommended migration destination is ConsenSys Quorum Blockchain Service. To onboard to this service,

register at the Quorum Blockchain Service page.

To self-manage your blockchain solution using virtual machines in Azure, see Azure VM-based Quorum guidance to set up transaction and validator nodes.

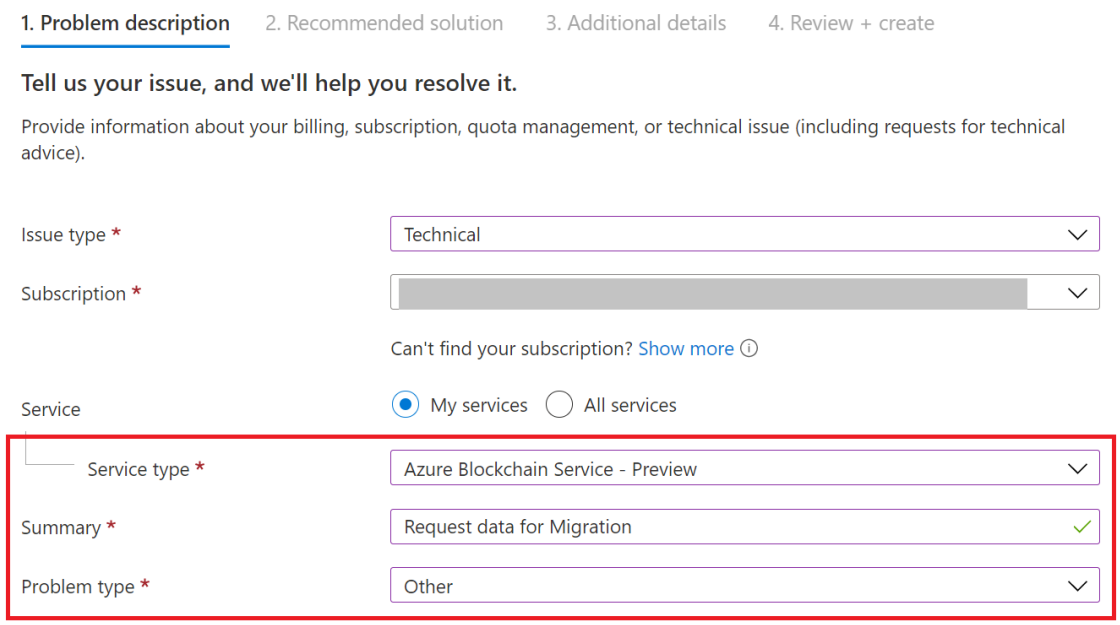
Export data from Azure Blockchain Service

Based on your current development state, you can either opt to use existing ledger data on Azure Blockchain Service or start a new network and use the solution of your choice. We recommend creating a new consortium based on a solution of your choice in all scenarios where you do not need or intend to use existing ledger data on Azure Blockchain Service.

Open support case

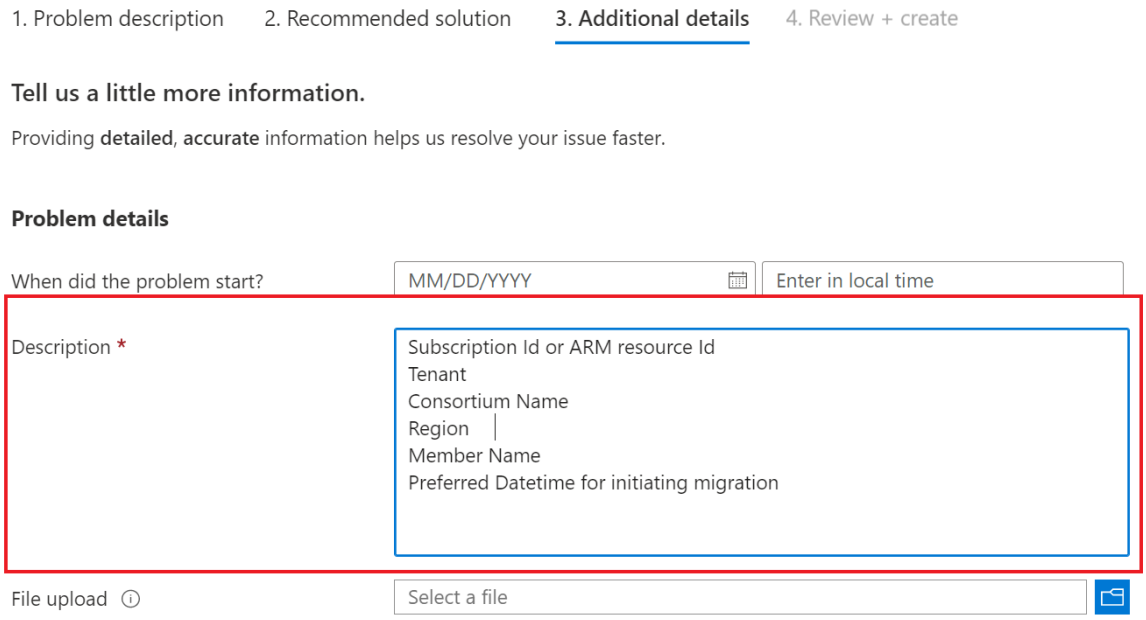
If you have a paid support plan, open a Microsoft Support ticket to pause the consortium and export your blockchain data.

1. Use the Azure portal to open a support ticket. In *Problem description*, enter the following details:



FIELD	RESPONSE
Issue type	Technical
Service	Azure Blockchain Service - Preview
Summary	Request data for migration
Problem type	other

2. In *Additional details*, include the following details:



- Subscription ID or Azure Resource Manager resource ID
- Tenant
- Consortium name
- Region
- Member name
- Preferred Datetime for initiating migration

If your consortium has multiple members, each member is required to open a separate support ticket for respective member data.

Pause consortium

You are required to coordinate with members of consortium to data export since the consortium will be paused for data export and transactions during this time will fail.

Azure Blockchain Service team pauses the consortium, exports a snapshot of data, and makes the data available through short-lived SAS URL for download in an encrypted format. The consortium is resumed after taking the snapshot.

> **IMPORTANT**
> You should stop all applications initiating new blockchain transactions on to the network. Active applications may lead to data loss or your original and migrated networks being out of sync.

Download data

Data format v1

Download the data using the Microsoft Support provided short-lived SAS URL link.

> **IMPORTANT**
> You are required to download your data within seven days.

Decrypt the data using the API access key. You can get the key from the Azure portal or through the REST API.

Caution

Only the default transaction node API access key 1 is used to encrypt all the nodes data of that member.

Do not reset the API access key in between of the migration.

Data format v2

In this version, the SAS token is encrypted instead of the data, resulting in faster snapshot creation. *If* you choose to migrate to ConsenSys Quorum Blockchain Service, importing to Quorum Blockchain Service is also faster.

After the SAS token is decrypted, data can be downloaded as normal. The data itself won't have an additional layer of encryption.

> **IMPORTANT**
> Creating a snapshot in data format v2 is about 8-10 times faster, so you have less downtime.

Caution

The default transaction node API access key 1 is used to encrypt the SAS token.

Do not reset the API access key between or during migration.

You can use the data with either ConsenSys Quorum Blockchain service or your IaaS VM-based deployment.

For ConsenSys Quorum Blockchain Service migration, contact ConsenSys at qbsmigration@consensys.net.

For using the data with your IaaS VM-based deployment, follow the steps in the Azure VM based Quorum guidance section of this article.

Delete resources

Once you have completed your data copy, it is recommended that you delete the Azure Blockchain member resources. You will continue to get billed while these resources exist.

Azure VM-based Quorum guidance

Use the following the steps to create transaction nodes and validator nodes.

Transaction node

A transaction node has two components. Tessera is used for the private transactions and Geth is used for the Quorum application. Validator nodes require only the Geth component.

Tessera

1. Install Java 11. For example, `apt install default-jre`.
2. Update paths in `tessera-config.json`. Change all references of `/working-dir/**` to `/opt/blockchain/data/working-dir/**`.
3. Update the IP address of other transaction nodes as per new IP address. HTTPS won't work since it is not enabled in the Tessera configuration. For information on how to configure TLS, see the Tessera configure TLS article.
4. Update NSG rules to allow inbound connections to port 9000.
5. Run Tessera using the following command:

```
java -Xms512M -Xmx1731M -Dlogback.configurationFile=/tessera/logback-tessera.xml -jar tessera.jar -
configfile /opt/blockchain/data/working-dir/tessera-config.json > tessera.log 2>&1 &
```

Geth

1. Update IPs in enode addresses in `/opt/blockchain/data/working-dir/dd/static-nodes.json`. Public IP address is allowed.

2. Make the same IP address changes under StaticNodes key in `/geth/config.toml`.
3. Update NSG rules to allow inbound connections to port 30303.
4. Run Geth using the following commands:

```
export NETWORK_ID='' # Get network ID from metadata. The network ID is the same for consortium.

PRIVATE_CONFIG=tm.ipc geth --config /geth/config.toml --datadir /opt/blockchain/data/working-dir/dd --networkid $NETWORK_ID --istanbul.blockperiod 5 --nodiscover --nousb --allow-insecure-unlock --verbosity 3 --txpool.globalslots 80000 --txpool.globalqueue 80000 --txpool.accountqueue 50000 --txpool.accountslots 50000 --targetgaslimit 700000000 --miner.gaslimit 800000000 --syncmode full --rpc --rpcaddr 0.0.0.0 --rpcport 3100 --rpccorsdomain '*' --rpcapi admin,db,eth,debug,net,shh,txpool,personal,web3,quorum,istanbul --ws --wsaddr 0.0.0.0 --wsport 3000 --wsorigins '*' --wsapi admin,db,eth,debug,net,shh,txpool,personal,web3,quorum,istanbul
```

Validator Node

Validator node steps are similar to the transaction node except that Geth startup command will have the additional flag `-mine`. Tessera is not started on a validator node. To run Geth without a paired Tessera, you pass `PRIVATE_CONFIG=ignore` in the Geth command. Run Geth using the following commands:

```
export NETWORK_ID=`j q '.APP_SETTINGS | fromjson | ."network-id"' env.json`

PRIVATE_CONFIG=ignore geth --config /geth/config.toml --datadir /opt/blockchain/data/working-dir/dd --networkid $NETWORK_ID --istanbul.blockperiod 5 --nodiscover --nousb --allow-insecure-unlock --verbosity 3 --txpool.globalslots 80000 --txpool.globalqueue 80000 --txpool.accountqueue 50000 --txpool.accountslots 50000 --targetgaslimit 700000000 --miner.gaslimit 800000000 --syncmode full --rpc --rpcaddr 0.0.0.0 --rpcport 3100 --rpccorsdomain '*' --rpcapi admin,db,eth,debug,net,shh,txpool,personal,web3,quorum,istanbul --ws --wsaddr 0.0.0.0 --wsport 3000 --wsorigins '*' --wsapi admin,db,eth,debug,net,shh,txpool,personal,web3,quorum,istanbul –mine
```

Upgrading Quorum

Azure Blockchain Service may be in running one of the following listed versions of Quorum. You can choose to use the same Quorum version or follow the below steps to use latest version of ConsenSys Quorum.

Upgrade Quorum version 2.6.0 or 2.7.0 to ConsenSys 21.1.0

Upgrading from Quorum version 2.6 or 2.7 version is straightforward. Download and update using the following links.

1. Download ConsenSys Quorum and related binaries v21.1.0.
2. Download the latest version of Tessera tessera-app-21.1.0-app.jar.

Upgrade Quorum version 2.5.0 to ConsenSys 21.1.0

1. Download ConsenSys Quorum and related binaries v21.1.0.
2. Download the latest version of Tessera tessera-app-21.1.0-app.jar. For versions 2.5.0, there are some minor genesis file changes. Make the following changes in the genesis file.
3. The value `byzantiumBlock` was set to 1 and it cannot be less than `constantinopleBlock` which is 0. Set the `byzantiumBlock` value to 0.
4. Set `petersburgBlock`, `istanbulBlock` to a future block. This value should be same across all nodes.
5. This step is optional. `ceil2Nby3Block` was incorrectly placed in Azure Blockchain Service Quorum 2.5.0 version. This needs to be inside the istanbul config and set the value future block. This value should be same across all nodes.

6. Run geth to reinitialize genesis block using following command:

```
geth --datadir "Data Directory Path" init "genesis file path"
```

7. Run Geth.

Exported data reference

This section describes the metadata, and folder structure to help import the data into your IaaS VM deployment.

Metadata info

NAME	SAMPLE	DESCRIPTION
consortium_name	<ConsortiumName>	Consortium name (unique across Azure Blockchain Service).
Consortium_Member_Count		Number of members in the consortium
member_name	<memberName>	Blockchain member name (unique across Azure Blockchain Service).
node_name	transaction-node	Node name (each member has multiple nodes).
network_id	543	Geth network ID.
is_miner	False	Is_Miner == true (Validator Node), Is_Miner == false (Transaction node)
quorum_version	2.7.0	Version of Quorum
tessera_version	0.10.5	Tessera version
java_version	java-11-openjdk-amd64	Java version Tessera uses
CurrentBlockNumber		Current block number for the blockchain network

Migrated Data Folder structure

At the top level, there are folders that correspond to each of the nodes of the members.

- **Standard SKU** - Two validator nodes (validator-node-0 and validator-node-1)
- **Basic SKU** - One validator node (validator-node-0)
- **Transaction Node** - Default transaction node named transaction-node.

Other transaction node folders are named after the transaction node name.

Node level folder structure

Each node level folder contains a zip file that is encrypted using the encryption key. For details on the obtaining the encryption key, see the Download data section of this article.

DIRECTORY/FILE	DESCRIPTION
/config/config.toml	Geth parameters. Command line parameters take precedence
/config/genesis.json	Genesis file
/config/logback-tessera.xml	Logback configuration for Tessera
/config/static-nodes.json	Static nodes. Bootstrap nodes are removed and auto-discovery is disabled.
/config/tessera-config.json	Tessera configuration
/data/c/	Tessera DB
/data/dd/	Geth data directory
/env/env	Metadata
/keys/	Tessera keys
/scripts/	Startup scripts (provided for reference only)

Frequently asked questions

What does service retirement mean for existing customers?

The existing Azure Blockchain Service deployments cannot be continued beyond September 10, 2021. Start evaluating alternatives suggested in this article before retirement based on your requirements.

What happens to existing deployments after the announcement of retirement?

Existing deployments are supported until September 10, 2021. Evaluate the suggested alternatives, migrate the data to the alternate offering, operate your requirement on the alternative offering, and start migrating from the deployment on Azure Blockchain Service.

How long will the existing deployments be supported on Azure Blockchain Service?

Existing deployments are supported until September 10, 2021.

Will I be allowed to create new Azure Blockchain members while in retirement phase?

After May 10, 2021, no new member creation or deployments are supported.

Use the Ethereum Blockchain connector with Azure Logic Apps

5/11/2021 • 11 minutes to read • Edit Online

Use the Ethereum Blockchain connector with Azure Logic Apps to perform smart contract actions and respond to smart contract events.

> **IMPORTANT**
>
> On September 10, 2021, Azure Blockchain will be retired. Please migrate ledger data from Azure Blockchain Service to an alternative offering based on your development status in production or evaluation. For more information on evaluating alternatives, see Migrate Azure Blockchain Service.

This article explains how you might use the Ethereum Blockchain connector to send blockchain information to another service or call a blockchain function. For example, let's say you want to create a REST-based microservice that returns information from a blockchain ledger. By using a logic app, you can accept HTTP requests that query information stored in a blockchain ledger.

Prerequisites

- Complete the optional prerequisite Quickstart: Use Visual Studio Code to connect to an Azure Blockchain Service consortium network. The quickstart guides you though installing Azure Blockchain Development Kit for Ethereum and setting up your blockchain development environment.
- If you are new to Azure Logic Apps, consider reviewing the Microsoft Learn modules Introduction to Azure Logic Apps and Call an API from a Logic Apps workflow using a custom connector.

Create a logic app

Azure Logic Apps helps you schedule and automate business processes and workflows when you need to integrate systems and services. First, you create a logic that uses the Ethereum Blockchain connector.

1. In the Azure portal, select **Create a resource** > **Integration** > **Logic App**.
2. Under **Create logic app**, provide details on where to create your logic app. After you're done, select **Create**.

   For more information on creating logic apps, see Create automated workflows with Azure Logic Apps.
3. After Azure deploys your app, select your logic app resource.
4. In the Logic Apps Designer, under **Templates**, select **Blank Logic App**.

Every logic app must start with a trigger, which fires when a specific event happens or when a specific condition is met. Each time the trigger fires, the Logic Apps engine creates a logic app instance that starts and runs your workflow.

The Ethereum Blockchain connector has one trigger and several actions. Which trigger or action you use depends on your scenario. Follow the section in this article that best matches your scenario.

If your workflow:

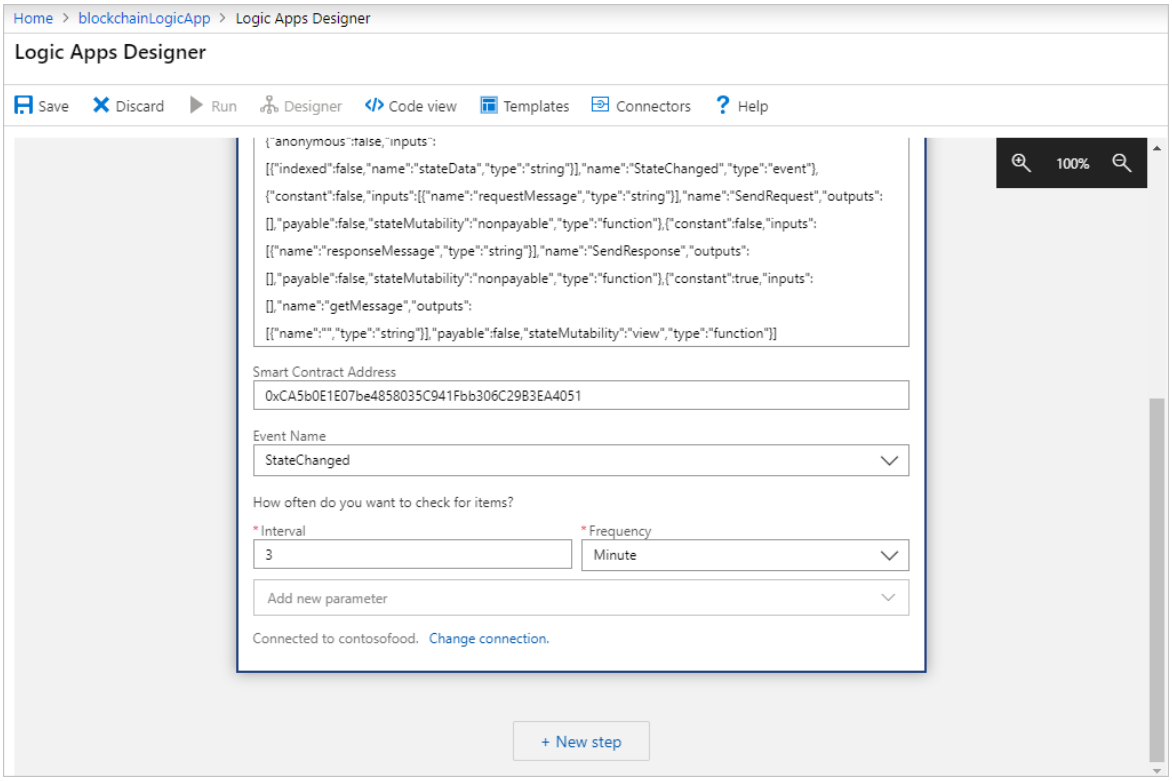
- Triggers when an event occurs on the blockchain, Use the event trigger.

- Queries or deploys a smart contract, Use actions.
- Follows a common scenario, Generate a workflow by using the developer kit.

Use the event trigger

Use Ethereum Blockchain event triggers when you want a logic app to run after a smart contract event occurs. For example, you want to send an email when a smart contract function is called.

1. In the Logic Apps Designer, select the Ethereum Blockchain connector.
2. From the **Triggers** tab, select **When a smart contract event occurs**.
3. Change or create an API connection to Azure Blockchain Service.
4. Enter the details about the smart contract that you want to check for events.



PROPERTY	DESCRIPTION
Contract ABI	The contract application binary interface (ABI) defines the smart contract interfaces. For more information, see Get the contract ABI.
Smart contract address	The contract address is the smart contract destination address on the Ethereum blockchain. For more information, see Get the contract address.
Event name	Select a smart contract event to check. The event triggers the logic app.
Interval and **Frequency**	Select how often you want to check for the event.

5. Select **Save**.

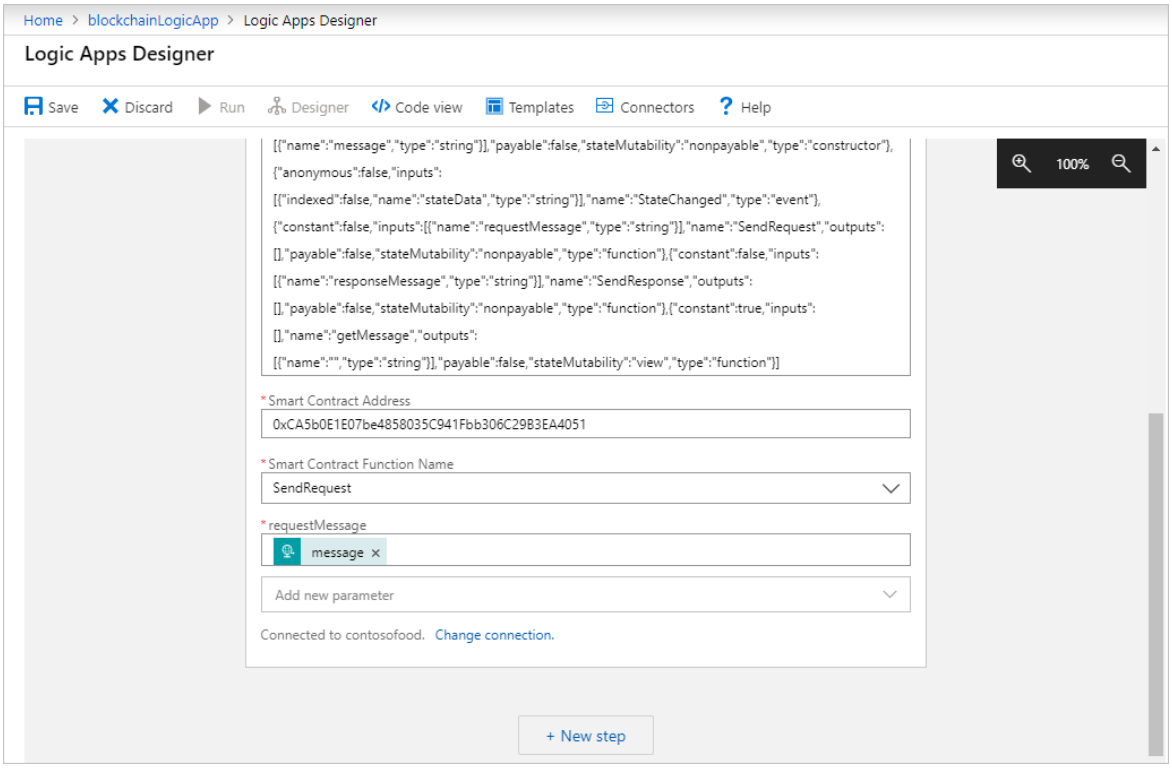
To complete your logic app, you can add a new step that performs an action based on the Ethereum Blockchain event trigger. For example, send an email.

Use actions

Use the Ethereum Blockchain actions when you want a logic app to perform an action on the blockchain ledger. For example, you want to create a REST-based microservice that calls a smart contract function when an HTTP request is made to a logic app.

Connector actions require a trigger. You can use an Ethereum Blockchain connector action as the next step after a trigger, such as an HTTP request trigger for a microservice.

1. In the Logic Apps Designer, select **New step** following a trigger.
2. Select the Ethereum Blockchain connector.
3. From the **Actions** tab, select one of the available actions.



4. Change or create an API connection to Azure Blockchain Service.
5. Depending on the action you chose, provide the following details about your smart contract function.

PROPERTY	DESCRIPTION
Contract ABI	The contract ABI defines the smart contract interfaces. For more information, see Get the contract ABI.
Contract bytecode	The compiled smart contract bytecode. For more information, see Get the contract bytecode.
Smart contract address	The contract address is the smart contract destination address on the Ethereum blockchain. For more information, see Get the contract address.
Smart contract function name	Select the smart contract function name for the action. The list is populated from the details in the contract ABI.

After selecting a smart contract function name, you might see required fields for function parameters. Enter the values or dynamic content required for your scenario.

You can now use your logic app. When the logic app event is triggered, the Ethereum Blockchain action runs. For example, an HTTP request trigger runs an Ethereum blockchain action to query a smart contract state value. This query results in an HTTP response that returns the value.

Generate a workflow

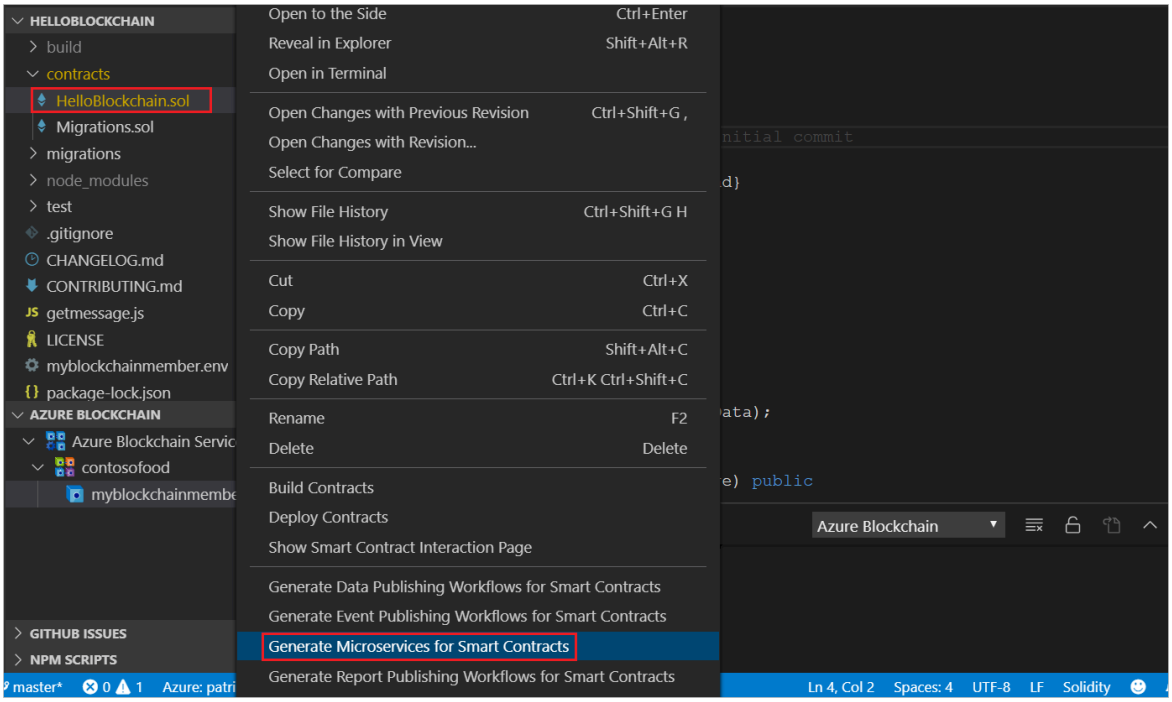
The Azure Blockchain Development Kit for Ethereum Visual Studio Code extension can generate logic app workflows for common scenarios. Four scenarios are available:

- Data publishing to an Azure SQL Database instance
- Event publishing to an instance of Azure Event Grid or Azure Service Bus
- Report publishing
- REST-based microservice

The Azure Blockchain Development Kit uses Truffle to simplify blockchain development. To generate a logic app based on a smart contract, you need a Truffle solution for the smart contract. You also need a connection to your Azure Blockchain Service consortium network. For more information, see Use Visual Studio Code to connect to an Azure Blockchain Service consortium network quickstart.

For example, the following steps generate a REST-based microservice logic app based on the quickstart **HelloBlockchain** smart contract:

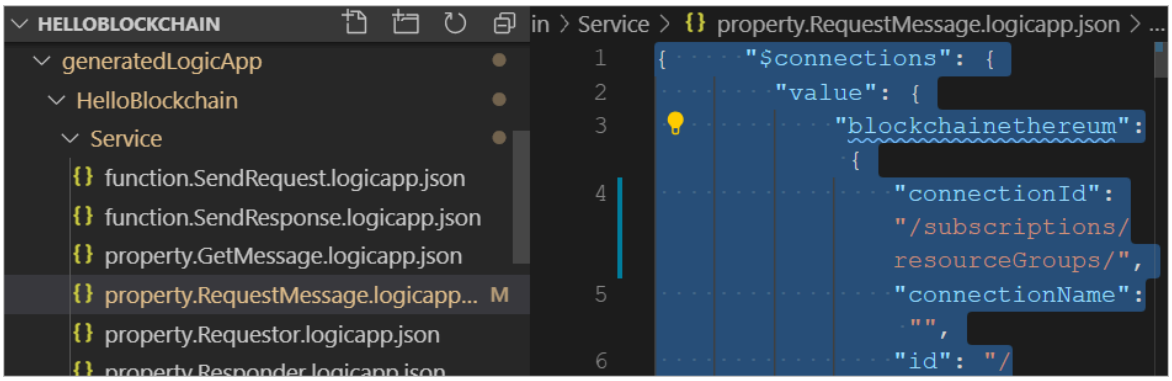
1. In the Visual Studio Code explorer sidebar, expand the **contracts** folder in your solution.
2. Right-click **HelloBlockchain.sol** and select **Generate Microservices for Smart Contracts** from the menu.



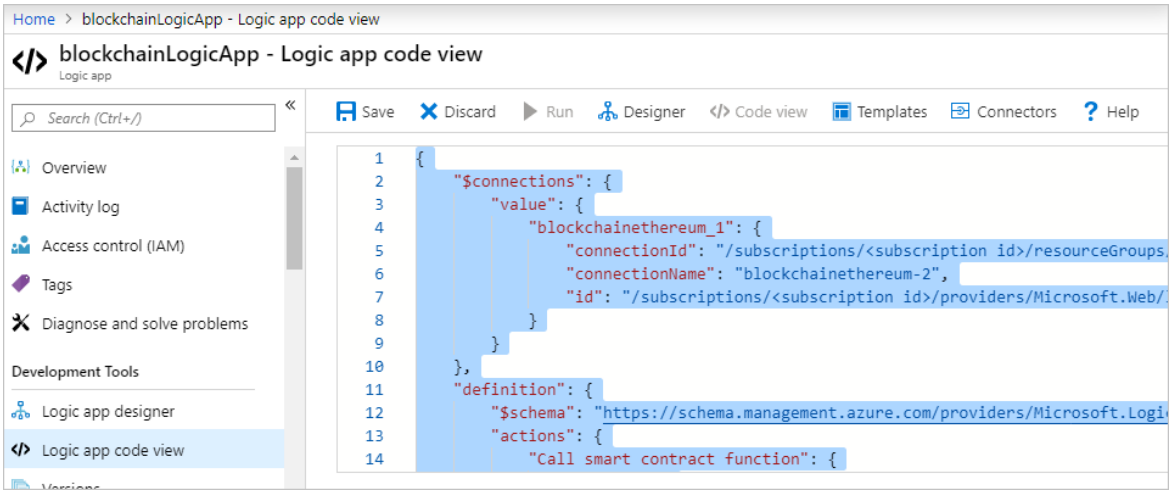
3. In the command palette, select **Logic App**.
4. Enter the **contract address**. For more information, see Get the contract address.
5. Select the Azure subscription and resource group for the logic app.

   The logic app configuration and code files are generated in the **generatedLogicApp** directory.
6. View the **generatedLogicApp/HelloBlockchain** directory. There's a logic app JSON file for each smart contract function, event, and property.

7. Open the **generatedLogicApp/HelloBlockchain/Service/property.RequestMessage.logicapp.json** file and copy the contents.



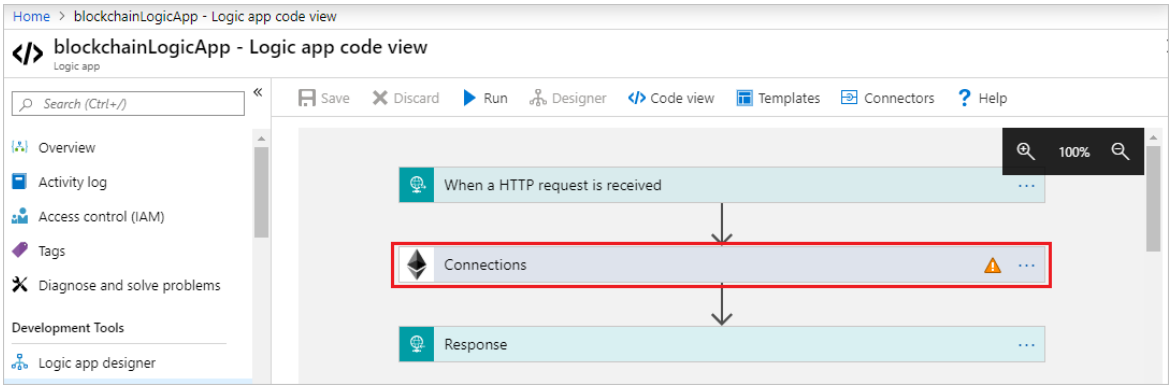
8. In your logic app, select **Logic app code view**. Replace the existing JSON with the generated logic app JSON.



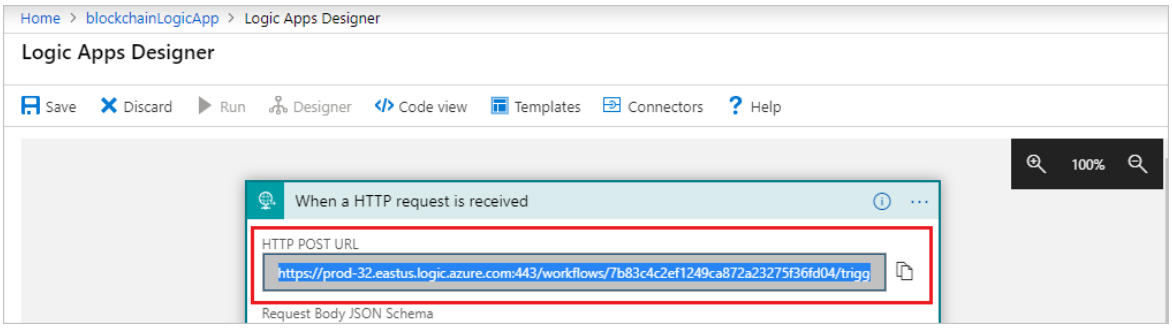
9. Select **Designer** to switch to the designer view.

10. The logic app includes the basic steps for the scenario. However, you need to update the configuration details for the Ethereum Blockchain connector.

11. Select the **Connections** step and change or create an API connection to Azure Blockchain Service.



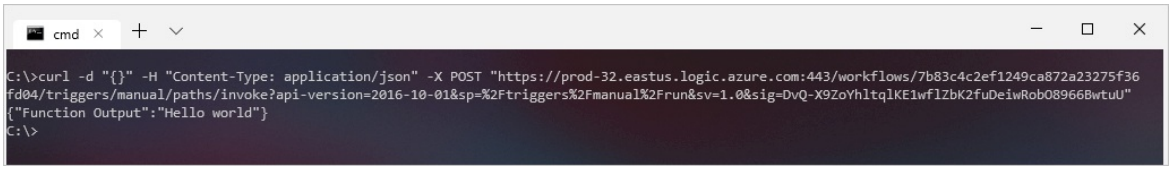
12. You can now use your logic app. To test the REST-based microservice, issue an HTTP POST request to the logic app request URL. Copy the **HTTP POST URL** contents from the **When an HTTP request is received** step.



13. Use cURL to create an HTTP POST request. Replace the placeholder text *<HTTP POST URL>* with the URL from the previous step.

```
curl -d "{}" -H "Content-Type: application/json" -X POST "<HTTP POST URL>"
```

The cURL command returns a response from the logic app. In this case, the response is the output from the **RequestMessage** smart contract function.



For more information about using the development kit, see the Azure Blockchain Development Kit for Ethereum wiki page.

Create an API connection

An API connection to a blockchain is required for the Ethereum Blockchain connector. You can use the API connector for multiple logic apps. Some properties are required and others depend on your scenario.

> **IMPORTANT**
> A private key or account address and password are required for creating transactions on a blockchain. Only one form of authentication is needed. You don't need to provide both the private key and account details. Querying contracts does not require a transaction. If you are using actions that query contract state, the private key or account address and password are not required.

To help you set up a connection to an Azure Blockchain Service member, the following list has possible properties you might need depending on your scenario.

PROPERTY	DESCRIPTION
Connection name	Name of the API connection. Required.
Ethereum RPC endpoint	HTTP address of the Azure Blockchain Service transaction node. Required. For more information, see Get the RPC endpoint.
Private key	Ethereum account private key. Private key or account address and password are required for transactions. For more information, see Get the private key.

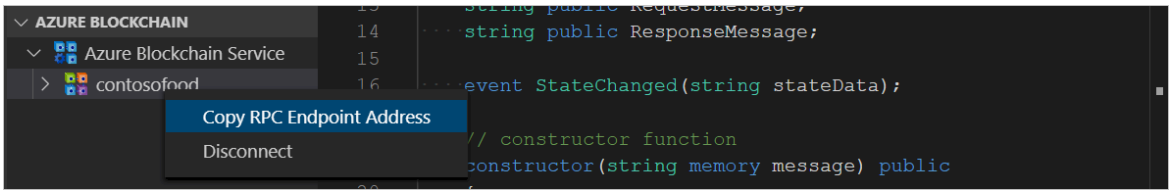
PROPERTY	DESCRIPTION
Account address	Azure Blockchain Service member account address. Private key or account address and password are required for transactions. For more information, see Get the account address.
Account password	The account password is set when you create the member. For information on resetting the password, see Ethereum account.

Get the RPC endpoint

The Azure Blockchain Service RPC endpoint address is required to connect to a blockchain network. You can get the endpoint address by using the Azure Blockchain Development Kit for Ethereum or the Azure portal.

To use the development kit:

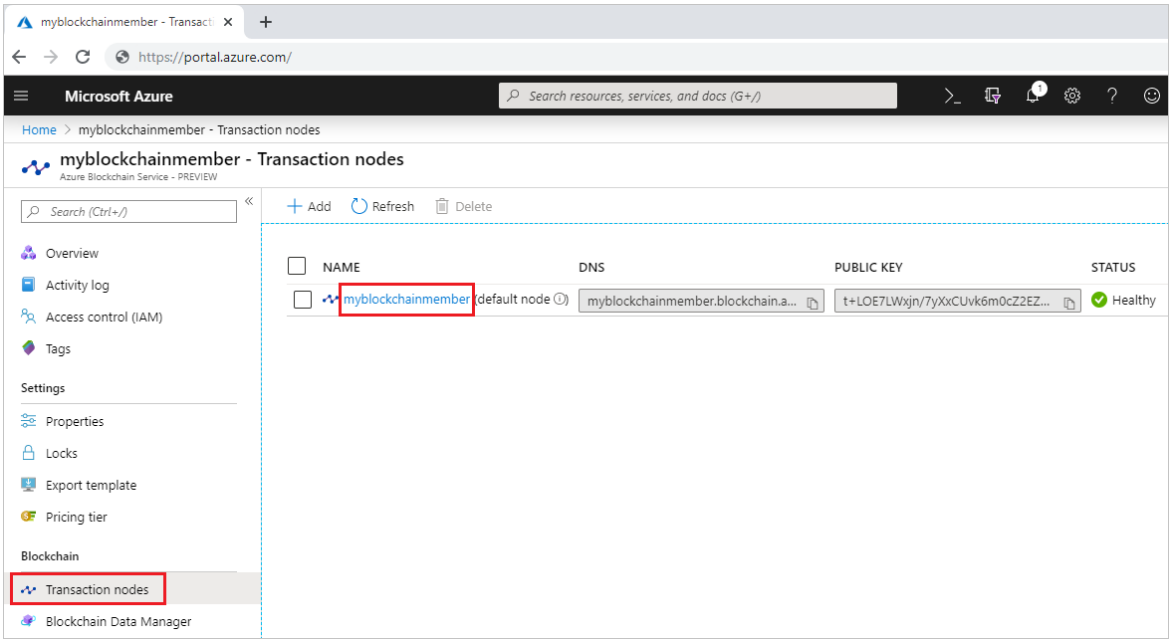
1. Under **Azure Blockchain Service** in Visual Studio Code, right-click the consortium.
2. Select **Copy RPC Endpoint Address**.



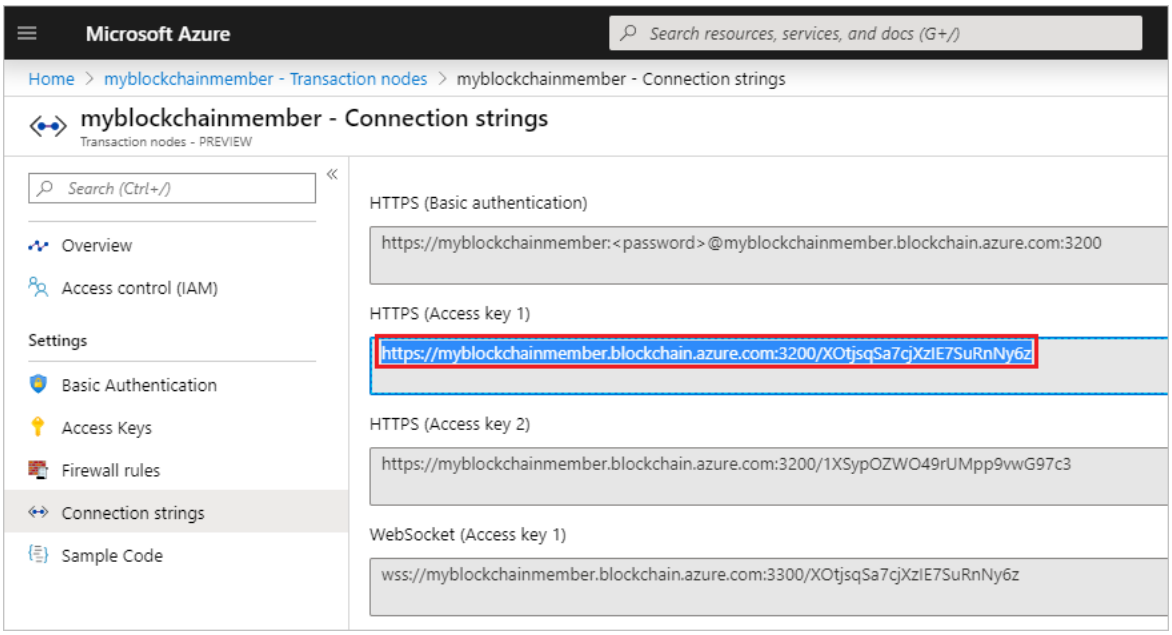
The RPC endpoint is copied to your clipboard.

To use the Azure portal:

1. Sign in to the Azure portal.
2. Go 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)** or **HTTPS (Access key 2)**.

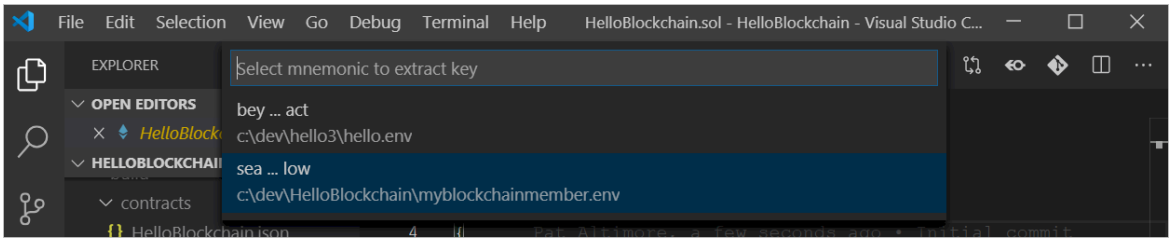


The RPC endpoint is the HTTPS URL, which includes the address and access key of your Azure Blockchain Service member transaction node.

Get the private key

You can use the Ethereum account's private key to authenticate when sending a transaction to the blockchain. Your Ethereum account's public and private keys are generated from a 12-word mnemonic. The Azure Blockchain Development Kit for Ethereum generates a mnemonic when you connect to an Azure Blockchain Service consortium member. You can get the endpoint address by using the development kit extension.

1. In Visual Studio Code, open the command palette (F1).
2. Select **Blockchain: Retrieve private key**.
3. Select the mnemonic you saved when connecting to the consortium member.

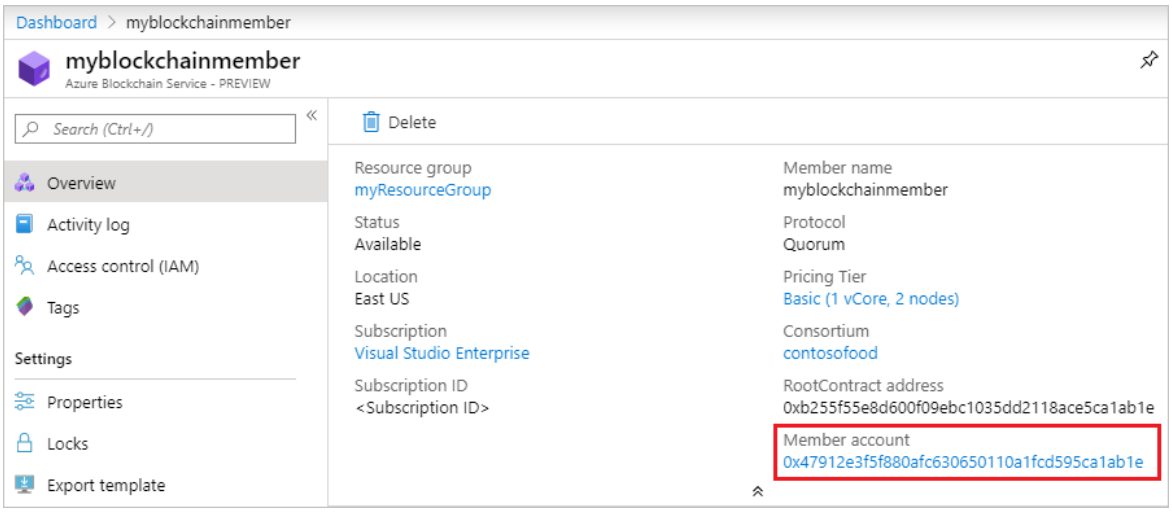


The private key is copied to your clipboard.

Get the account address

You can use the member account and password to authenticate when you send a transaction to the blockchain. The password is set when you create the member.

1. In the Azure portal, go to your Azure Blockchain Service overview page.
2. Copy the **Member account** address.



For more information on the account address and password, see Ethereum account.

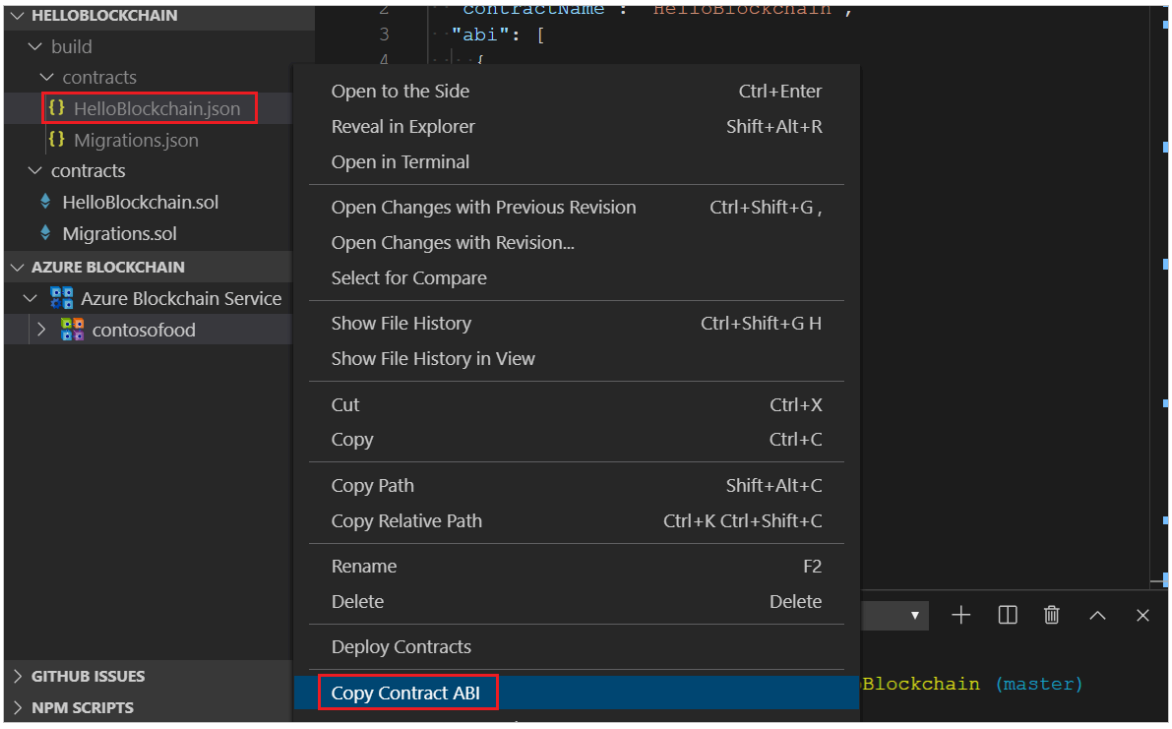
Get the contract ABI

The contract ABI defines the smart contract interfaces. It describes how to interact with the smart contract. You can get the contract ABI by using the Azure Blockchain Development Kit for Ethereum. You can also get it from the contract metadata file created by the Solidity compiler.

To use the development kit:

If you used the development kit or Truffle to build your smart contract, you can use the extension to copy the contract ABI to the clipboard.

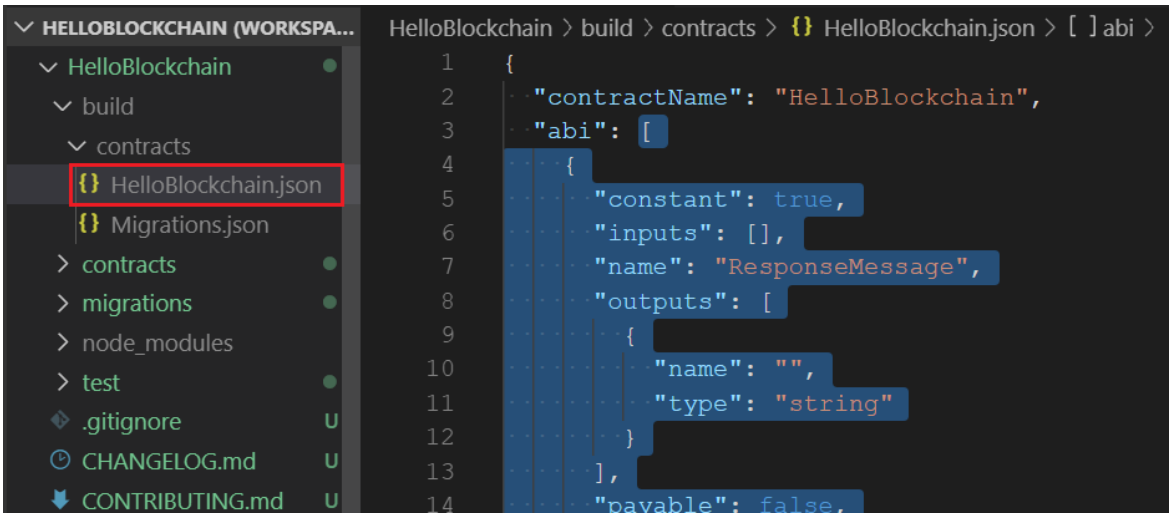
1. In the Visual Studio Code explorer pane, expand the **build/contracts** folder of your Solidity project.
2. Right-click the contract metadata JSON file. The file name is the smart contract name followed by the **.json** extension.
3. Select **Copy Contract ABI**.



The contract ABI is copied to the clipboard.

To use the contract metadata file:

1. Open the contract metadata file contained in the **build/contracts** folder of your Solidity project. The file name is the smart contract name followed by the **.json** extension.
2. Find the **abi** section in the JSON file.
3. Copy the **abi** JSON array.



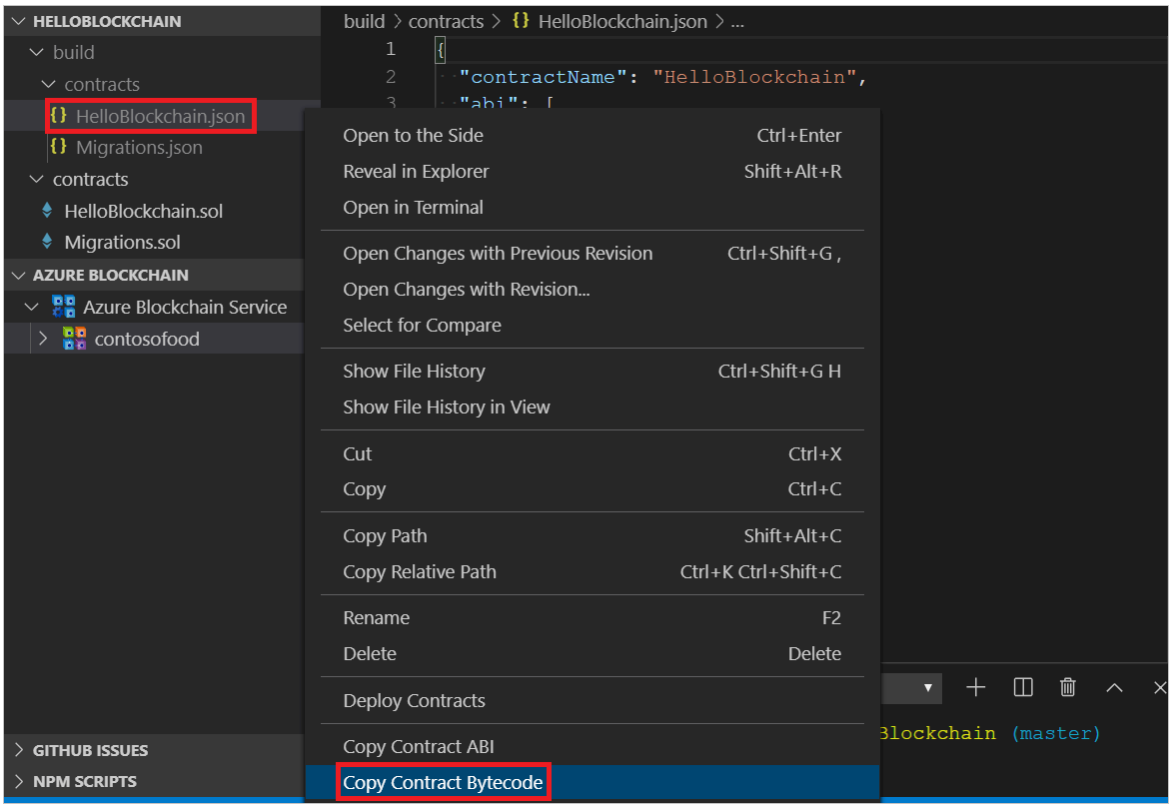
Get the contract bytecode

The contract bytecode is the compiled smart contract executed by the Ethereum virtual machine. You can get the contract bytecode by using the Azure Blockchain Development Kit for Ethereum. You can also get it from the Solidity compiler.

To use the development kit:

If you used the development kit or Truffle to build your smart contract, you can use the extension to copy the contract bytecode to the clipboard.

1. In the Visual Studio Code explorer pane, expand the **build/contracts** folder of your Solidity project.
2. Right-click the contract metadata JSON file. The file name is the smart contract name followed by the **.json** extension.
3. Select **Copy Contract Bytecode**.



The contract bytecode is copied to the clipboard.

To use the contract metadata file:

1. Open the contract metadata file contained in the **build/contracts** folder of your Solidity project. The file name is the smart contract name followed by the **.json** extension.
2. Find the **bytecode** element in the JSON file.
3. Copy the **bytecode** value.

To use the Solidity compiler:

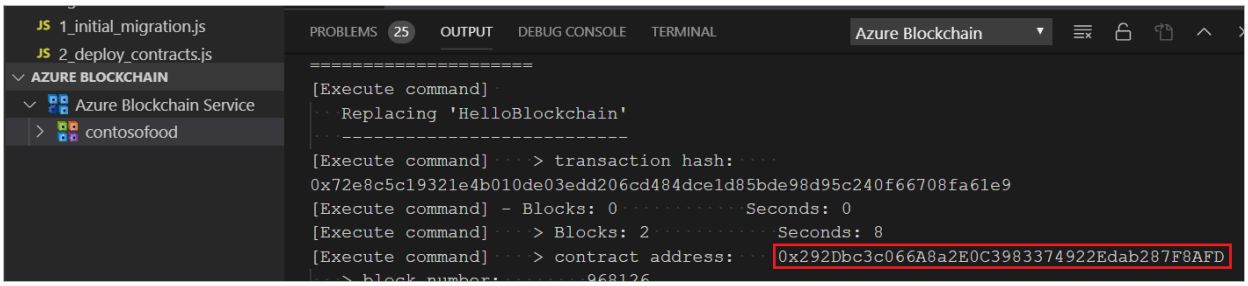
Use the command `solc --bin <smart contract>.sol` to generate the contract bytecode.

Get the contract address

The contract address is the smart contract destination address on the Ethereum blockchain. You use this address to send a transaction or query state of a smart contract. You can get the contract address from the Truffle migration output or the contract metadata file.

To use the Truffle migrate output:

Truffle displays the contract address after deployment of the smart contract. Copy the **contract address** from the output.



To use the contract metadata file:

1. Open the contract metadata file contained in the **build/contracts** folder of your Solidity project. The file name is the smart contract name followed by the **.json** extension.
2. Find the **networks** section in the JSON file.
3. Private networks are identified by an integer network ID. Find the address value within the network section.
4. Copy the **address** value.

Next steps

Watch common scenarios in the video Doing more with Logic Apps.

Monitor Azure Blockchain Service through Azure Monitor

3/5/2021 • 7 minutes to read • Edit Online

As customers run production grade blockchain scenarios on Azure Blockchain Service (ABS), it becomes critical to monitor the resources for availability, performance, and operations. This article describes the monitoring data generated by Azure Blockchain Service and how one can use the various features and integrations of Azure Monitor to analyze and alert on, to manage production grade environments.

What is Azure Monitor?

Azure Blockchain Service creates monitoring data using Azure Monitor, which is a full stack monitoring service in Azure that provides a complete set of features to monitor your Azure resources. For more information about Azure Monitor, see Monitoring Azure resources with Azure Monitor.

The following sections build on this article by describing the specific data gathered from Azure Blockchain Service and providing examples for configuring data collection and analyzing this data with Azure tools.

Monitor data collected from Azure Blockchain Service

Azure Blockchain Service collects the same kind of monitoring data as other Azure resources, which are described in Monitoring data from Azure resources. See Monitor Azure Blockchain Service data reference for a detailed reference of the logs and metrics created by Azure Blockchain Service.

The overview page in the Azure portal for each Azure Blockchain Service member resource includes a brief view of the transactions including the requests handled and processed blocks. Some of this data is collected automatically and available for analysis once you create the Azure Blockchain Service member resource, while you can enable additional data collection with additional configuration.

Diagnostic settings

Platform metrics and the Activity log are collected automatically, but you must create a diagnostic setting to collect resource logs or forward them outside of Azure Monitor. See Create diagnostic setting to collect platform logs and metrics in Azure for the detailed process for creating a diagnostic setting using the Azure portal, CLI, or PowerShell.

When you create a diagnostic setting, you specify which categories of logs to collect. The categories for Azure Blockchain Service are listed below.

Blockchain proxy logs – Select the category if you want to monitor the NGNIX proxy logs. All the customer transaction details are available for audit and debug purpose.

Blockchain application logs – Select the category to get logs of the blockchain application hosted by the managed service. For example, for an ABS-Quorum member, these logs would be the logs from Quorum itself.

Metric requests: Select the option to collect metric data from Azure Cosmos DB to the destinations in the diagnostic setting, which is collected automatically in Azure Metrics. Collect metric data with resource logs to analyze both kinds of data together and to send metric data outside of Azure Monitor.

Analyze metric data

You can analyze metrics for Azure Blockchain Service with Metrics explorer, navigate to Metrics tab under Monitoring section in ABS resource blade. See Getting started with Azure Metrics Explorer for details on using the tool. The complete metrics for Azure Blockchain Service are in the namespace Azure Blockchain Service standard metrics.

You can use **node** dimension when adding a filter or splitting the metrics, which basically provides metric values per transaction nodes and validator nodes of the ABS member.

Analyze log data

Here are some queries that you can enter in the Log search bar to help you monitor your Azure Blockchain Service members. These queries work with the new language.

To query the error conditions in the Blockchain application logs, use the below query:

```
BlockchainApplicationLog | where BlockchainMessage contains "ERROR" or BlockchainMessage contains "fatal"
```

To query the error conditions in the Blockchain proxy logs, use the below query

```
BlockchainProxyLog
|filterCode!=200
|limit500
```

You can use the time filters available in Azure logs to filter the query for a specific time range.

Monitor Azure Blockchain Service data reference

This article provides a reference of log and metric data collected to analyze the performance and availability of Azure Blockchain Service.

Resource logs

All resource logs share a top-level common schema with few unique properties specific to the blockchain service. You can refer to the article Top-level resource logs schema, details of the Azure Blockchain Service specific properties are covered below

The following table lists the properties for Azure Blockchain proxy logs when they're collected in Azure Monitor Logs or Azure Storage.

PROPERTY NAME	DESCRIPTION
time	The date and time (UTC) when the operation occurred.
resourceID	The Azure Blockchain Service resource for which logs are enabled.
category	For Azure Blockchain Service, the values possible are **Proxylogs** and **Applicationlogs**.
operationName	The name of the operation represented by this event.
Log level	By default, Azure Blockchain Service enables **Informational** log level.

PROPERTY NAME	DESCRIPTION
NodeLocation	Azure region where the blockchain member is deployed.
BlockchainNodeName	The name of the node of the Azure Blockchain Service member on which operation is performed.
EthMethod	The method, which is called by the underlying blockchain protocol, in Quorum, it could be eth_sendTransactions, eth_getBlockByNumber etc.
Agent	The user agent that is acting on behalf of a user, such as web browser Mozilla, Edge etc. Examples of the values are: "Mozilla/5.0 (Linux x64) node.js/8.16.0 v8/6.2.414.77"
Code	HTTP error codes. Usually 4XX and 5XX are error conditions.
NodeHost	The DNS name of the node.
RequestMethodName	HTTP method called, the possible values here are PUT for create member, GET for getting details of existing member, DELETE for delete member, PATCH for updating member.
BlockchainMemberName	Azure Blockchain Service member name provided by the user.
Consortium	Name of the consortium as provided by the user.
Remote	The IP of the client where the request is coming.
RequestSize	Size of the request made in bytes.
RequestTime	The duration of the request in milliseconds.

The following table lists the properties for Azure Blockchain application logs.

PROPERTY NAME	DESCRIPTION
time	The date and time (UTC) when the operation occurred.
resourceID	The Azure Blockchain Service resource for which logs are enabled.
category	For Azure Blockchain Service, the value possible are **Proxylogs** and **Applicationlogs**.
operationName	The name of the operation represented by this event.
Log level	By default, Azure Blockchain Service enables **Informational** log level.
NodeLocation	Azure region where the blockchain member is deployed.

PROPERTY NAME	DESCRIPTION
BlockchainNodeName	The name of the node of the Azure Blockchain Service member on which operation is performed.
BlockchainMessage	This field will contain the Blockchain application log that is the data plain logs. For ABS-Quorum, this would have Quorum logs. It has information about what type of log entry is it that is informational, error, warning and a string that gives more information on the action executed.
TenantID	The region-specific tenant of the Azure Blockchain Service. The format of this field is https://westlake-rp-prod..cloudapp.azure.com where region specifies the Azure region of the member deployed.
SourceSystem	The system populates the logs, in this case it is **Azure**.

Metrics

The following tables lists the platform metrics collected for Azure Blockchain Service. All metrics are stored in the namespace **Azure Blockchain Service** standard metrics.

For a list of all Azure Monitor supported metrics (including Azure Blockchain Service), see Azure Monitor supported metrics.

Blockchain metrics

The following table specifies the list of Blockchain metrics that are collected for the Azure Blockchain Service member resource.

METRIC NAME	UNIT	AGGREGATION TYPE	DESCRIPTION
Pending Transactions	Count	Average	The number of transactions that are waiting to be mined.
Processed Blocks	Count	Sum	The number of blocks processed in each time interval. Currently the block size is 5 seconds, hence in a minute each node will process 12 blocks and 60 blocks in 5 minutes.
Processed Transactions	Count	Sum	The number of transactions processed in a block.

METRIC NAME	UNIT	AGGREGATION TYPE	DESCRIPTION
Queued Transactions	Count	Average	The number of transactions that cannot be immediately mined. It can be because they arrived out of order and the future one is waiting for previous transaction to arrive. Or, it can be two transactions have the same number only used once (nonce) and the same gas value, hence the second one cannot be mined.

Connection metrics

The following table lists the different connection metrics that are collected for the Azure Blockchain Service member resource. These are NGINX proxy metrics.

METRIC NAME	UNIT	AGGREGATION TYPE	DESCRIPTION
Accepted Connections	Count	Sum	The total number of accepted client connections.
Active Connections	Count	Average	The current number of active client connections including Waiting connections.
Handled Connections	Count	Sum	The total number of handled connections. Generally, the parameter value is the same as accepted connections unless some resource limits have been reached.
Handled Requests	Count	Sum	The total number of client requests.

Performance Metrics

The following table lists the performance metrics that are collected for each of the nodes of the Azure Blockchain member resource.

METRIC NAME	UNIT	AGGREGATION TYPE	DESCRIPTION
CPU Usage percentage	Percentage	Max	The percentage of the CPU usage.
IO Read Bytes	Kilobytes	Sum	The sum of IO read bytes across all nodes of the blockchain member resource.

METRIC NAME	UNIT	AGGREGATION TYPE	DESCRIPTION
IO Write Bytes	Kilobytes	Sum	The sum of IO writes bytes across all nodes of the blockchain member resource.
Memory Limit	Gigabytes	Average	Maximum memory available for the blockchain process per node.
Memory Usage	Gigabytes	Average	The amount of memory used averaged across all nodes.
Memory Usage Percentage	Percentage	Average	The percentage of the memory used averaged across all nodes.
Storage Usage	Gigabytes	Average	The GB of storage used averaged across all nodes.

Next Steps

Learn more about Blockchain Data Manager to capture and transform blockchain data to Azure Event Grid.