Contents

- Azure Blockchain Workbench (public preview)
- Overview
  - About Azure Blockchain Workbench
- Tutorials
  - Create blockchain app
  - Use blockchain app
- Concepts
  - Architecture
  - Integration patterns
- How-to guides
  - Deploy Workbench
  - Manage Workbench users
  - Versioning apps
  - Use Workbench API
  - Troubleshoot apps
  - Use Workbench data
    - Configure DB firewall
    - Get DB details
    - Use data in Excel
    - Use data in Power BI
    - Use data in SQL Management Studio
- Reference
  - Configuration
  - Database views
  - Messaging API
  - REST API
- Resources
  - Blog
  - Code samples

Community
Forum
Product feedback
Solution templates
Ethereum Proof-of-Authority consortium
Hyperledger Fabric consortium

What is Azure Blockchain Workbench?

5/20/2019 • 2 minutes to read • Edit Online

Azure Blockchain Workbench is a collection of Azure services and capabilities designed to help you create and deploy blockchain applications to share business processes and data with other organizations. Azure Blockchain Workbench provides the infrastructure scaffolding for building blockchain applications enabling developers to focus on creating business logic and smart contracts. It also makes it easier to create blockchain applications by integrating several Azure services and capabilities to help automate common development tasks.

Create blockchain applications

With Blockchain Workbench, you can define blockchain applications using configuration and writing smart contract code. You can jumpstart blockchain application development and focus on defining your contract and writing business logic instead of building scaffolding and setting up supporting services.

Manage applications and users

Azure Blockchain Workbench provides a web application and REST APIs for managing blockchain applications and users. Blockchain Workbench administrators can manage application access and assign your users to application roles. Azure AD users are automatically mapped to members in the application.

Integrate blockchain with applications

You can use the Blockchain Workbench REST APIs and message-based APIs to integrate with existing systems. The APIs provide an interface to allow for replacing or using multiple distributed ledger technologies, storage, and database offerings.

Blockchain Workbench can transform messages sent to its message-based API to build transactions in a format expected by that blockchain's native API. Workbench can sign and route transactions to the appropriate blockchain.

Workbench automatically delivers events to Service Bus and Event Grid to send messages to downstream consumers. Developers can integrate with either of these messaging systems to drive transactions and to look at results.

Deploy a blockchain network

Azure Blockchain Workbench simplifies consortium blockchain network setup as a preconfigured solution with an Azure Resource Manager solution template. The template provides simplified deployment that deploys all components needed to run a consortium. Today, Blockchain Workbench currently supports Ethereum.

Use Active Directory

With existing blockchain protocols, blockchain identities are represented as an address on the network. Azure Blockchain Workbench abstracts away the blockchain identity by associating it with an Active Directory identity, making it simpler to build enterprise applications with Active Directory identities.

Synchronize on-chain data with off-chain storage

Azure Blockchain Workbench makes it easier to analyze blockchain events and data by automatically synchronizing data on the blockchain to off-chain storage. Instead of extracting data directly from the blockchain, you can query

off-chain database systems such as SQL Server. Blockchain expertise is not required for end users who are doing data analysis tasks.

Next steps

Azure Blockchain Workbench architecture

Tutorial: Create a blockchain application in Azure Blockchain Workbench

5/30/2019 • 8 minutes to read • Edit Online

You can use Azure Blockchain Workbench to create blockchain applications that represent multi-party workflows defined by configuration and smart contract code.

You'll learn how to:

- Configure a blockchain application
- Create a smart contract code file
- Add a blockchain application to Blockchain Workbench
- Add members to the blockchain application

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

Prerequisites

- A Blockchain Workbench deployment. For more information, see Azure Blockchain Workbench deployment for details on deployment.
- Azure Active Directory users in the tenant associated with Blockchain Workbench. For more information, see add Azure AD users in Azure Blockchain Workbench.
- A Blockchain Workbench administrator account. For more information, see add Blockchain Workbench administrators in Azure Blockchain Workbench.

Hello, Blockchain!

Let's build a basic application in which a requestor sends a request and a responder send a response to the request. For example, a request can be, "Hello, how are you?", and the response can be, "I'm great!". Both the request and the response are recorded on the underlying blockchain.

Follow the steps to create the application files or you can download the sample from GitHub.

Configuration file

Configuration metadata defines the high-level workflows and interaction model of the blockchain application. Configuration metadata represents the workflow stages and interaction model of the blockchain application.

1. In your favorite editor, create a file named `HelloBlockchain.json`.
2. Add the following JSON to define the configuration of the blockchain application.

```
{
  "ApplicationName": "HelloBlockchain",
  "DisplayName": "Hello, Blockchain!",
  "Description": "A simple application to send request and get response",
  "ApplicationRoles": [
    {
      "Name": "Requestor",
      "Description": "A person sending a request."
    },
    {
      "Name": "Responder",
```

```
      "Description": "A person responding to a request"
    }
  ],
  "Workflows": [
    {
      "Name": "HelloBlockchain",
      "DisplayName": "Request Response",
      "Description": "A simple workflow to send a request and receive a response.",
      "Initiators": [ "Requestor" ],
      "StartState": "Request",
      "Properties": [
        {
          "Name": "State",
          "DisplayName": "State",
          "Description": "Holds the state of the contract.",
          "Type": {
            "Name": "state"
          }
        },
        {
          "Name": "Requestor",
          "DisplayName": "Requestor",
          "Description": "A person sending a request.",
          "Type": {
            "Name": "Requestor"
          }
        },
        {
          "Name": "Responder",
          "DisplayName": "Responder",
          "Description": "A person sending a response.",
          "Type": {
            "Name": "Responder"
          }
        },
        {
          "Name": "RequestMessage",
          "DisplayName": "Request Message",
          "Description": "A request message.",
          "Type": {
            "Name": "string"
          }
        },
        {
          "Name": "ResponseMessage",
          "DisplayName": "Response Message",
          "Description": "A response message.",
          "Type": {
            "Name": "string"
          }
        }
      ],
      "Constructor": {
        "Parameters": [
          {
            "Name": "message",
            "Description": "...",
            "DisplayName": "Request Message",
            "Type": {
              "Name": "string"
            }
          }
        ]
      },
      "Functions": [
        {
          "Name": "SendRequest",
          "DisplayName": "Request",
          "Description": "..."
```

```
          "Description": "..." },
          "Parameters": [
            {
              "Name": "requestMessage",
              "Description": "...",
              "DisplayName": "Request Message",
              "Type": {
                "Name": "string"
              }
            }
          ]
        },
        {
          "Name": "SendResponse",
          "DisplayName": "Response",
          "Description": "...",
          "Parameters": [
            {
              "Name": "responseMessage",
              "Description": "...",
              "DisplayName": "Response Message",
              "Type": {
                "Name": "string"
              }
            }
          ]
        }
      ],
      "States": [
        {
          "Name": "Request",
          "DisplayName": "Request",
          "Description": "...",
          "PercentComplete": 50,
          "Value": 0,
          "Style": "Success",
          "Transitions": [
            {
              "AllowedRoles": ["Responder"],
              "AllowedInstanceRoles": [],
              "Description": "...",
              "Function": "SendResponse",
              "NextStates": [ "Respond" ],
              "DisplayName": "Send Response"
            }
          ]
        },
        {
          "Name": "Respond",
          "DisplayName": "Respond",
          "Description": "...",
          "PercentComplete": 90,
          "Value": 1,
          "Style": "Success",
          "Transitions": [
            {
              "AllowedRoles": [],
              "AllowedInstanceRoles": ["Requestor"],
              "Description": "...",
              "Function": "SendRequest",
              "NextStates": [ "Request" ],
              "DisplayName": "Send Request"
            }
          ]
        }
      ]
    }
  ]
}
```

3. Save the `HelloBlockchain.json` file.

The configuration file has several sections. Details about each section are as follows:

Application metadata

The beginning of the configuration file contains information about the application including application name and description.

Application roles

The application roles section defines the user roles who can act or participate within the blockchain application. You define a set of distinct roles based on functionality. In the request-response scenario, there is a distinction between the functionality of a requestor as an entity that produces requests and a responder as an entity that produces responses.

Workflows

Workflows define one or more stages and actions of the contract. In the request-response scenario, the first stage (state) of the workflow is a requestor (role) takes an action (transition) to send a request (function). The next stage (state) is a responder (role) takes an action (transition) to send a response (function). An application's workflow can involve properties, functions, and states required describe the flow of a contract.

For more information about the contents of configuration files, see Azure Blockchain Workflow configuration reference.

Smart contract code file

Smart contracts represent the business logic of the blockchain application. Currently, Blockchain Workbench supports Ethereum for the blockchain ledger. Ethereum uses Solidity as its programming language for writing self-enforcing business logic for smart contracts.

Smart contracts in Solidity are similar to classes in object-oriented languages. Each contract contains state and functions to implement stages and actions of the smart contract.

In your favorite editor, create a file called `HelloBlockchain.sol`.

Version pragma

As a best practice, indicate the version of Solidity you are targeting. Specifying the version helps avoid incompatibilities with future Solidity versions.

Add the following version pragma at the top of `HelloBlockchain.sol` smart contract code file.

```
pragma solidity >=0.4.25 <0.6.0;
```

Configuration and smart contract code relationship

Blockchain Workbench uses the configuration file and smart contract code file to create a blockchain application. There is a relationship between what is defined in the configuration and the code in the smart contract. Contract details, functions, parameters, and types are required to match to create the application. Blockchain Workbench verifies the files prior to application creation.

Contract

Add the **contract** header to your `HelloBlockchain.sol` smart contract code file.

```
contract HelloBlockchain {
```

State variables

State variables store values of the state for each contract instance. The state variables in your contract must match the workflow properties defined in the configuration file.

Add the state variables to your contract in your `HelloBlockchain.sol` smart contract code file.

```
    //Set of States
    enum StateType { Request, Respond}

    //List of properties
    StateType public  State;
    address public  Requestor;
    address public  Responder;

    string public RequestMessage;
    string public ResponseMessage;
```

Constructor

The constructor defines input parameters for a new smart contract instance of a workflow. Required parameters for the constructor are defined as constructor parameters in the configuration file. The number, order, and type of parameters must match in both files.

In the constructor function, write any business logic you want to perform prior to creating the contract. For example, initialize the state variables with starting values.

Add the constructor function to your contract in your `HelloBlockchain.sol` smart contract code file.

```
    // constructor function
    constructor(string memory message) public
    {
        Requestor = msg.sender;
        RequestMessage = message;
        State = StateType.Request;
    }
```

Functions

Functions are the executable units of business logic within a contract. Required parameters for the function are defined as function parameters in the configuration file. The number, order, and type of parameters must match in both files. Functions are associated to transitions in a Blockchain Workbench workflow in the configuration file. A transition is an action performed to move to the next stage of an application's workflow as determined by the contract.

Write any business logic you want to perform in the function. For example, modifying a state variable's value.

1. Add the following functions to your contract in your `HelloBlockchain.sol` smart contract code file.

```
    // call this function to send a request
    function SendRequest(string memory requestMessage) public
    {
        if (Requestor != msg.sender)
        {
            revert();
        }

        RequestMessage = requestMessage;
        State = StateType.Request;
    }

    // call this function to send a response
    function SendResponse(string memory responseMessage) public
    {
        Responder = msg.sender;

        ResponseMessage = responseMessage;
        State = StateType.Respond;
    }
}
```

2. Save your `HelloBlockchain.sol` smart contract code file.

Add blockchain application to Blockchain Workbench

To add a blockchain application to Blockchain Workbench, you upload the configuration and smart contract files to define the application.

1. In a web browser, navigate to the Blockchain Workbench web address. For example, `https://{workbench URL}.azurewebsites.net/` The web application is created when you deploy Blockchain Workbench. For information on how to find your Blockchain Workbench web address, see Blockchain Workbench Web URL
2. Sign in as a Blockchain Workbench administrator.
3. Select **Applications** > **New**. The **New application** pane is displayed.
4. Select **Upload the contract configuration** > **Browse** to locate the **HelloBlockchain.json** configuration file you created. The configuration file is automatically validated. Select the **Show** link to display validation errors. Fix validation errors before you deploy the application.
5. Select **Upload the contract code** > **Browse** to locate the **HelloBlockchain.sol** smart contract code file. The code file is automatically validated. Select the **Show** link to display validation errors. Fix validation errors before you deploy the application.
6. Select **Deploy** to create the blockchain application based on the configuration and smart contract files.

Deployment of the blockchain application takes a few minutes. When deployment is finished, the new application is displayed in **Applications**.

> **NOTE**
> You can also create blockchain applications by using the Azure Blockchain Workbench REST API.

Add blockchain application members

Add application members to your application to initiate and take actions on contracts. To add application members, you need to be a Blockchain Workbench administrator.

1. Select **Applications** > **Hello, Blockchain!**.

2. The number of members associated to the application is displayed in the upper right corner of the page. For a new application, the number of members will be zero.
3. Select the **members** link in the upper right corner of the page. A current list of members for the application is displayed.
4. In the membership list, select **Add members**.
5. Select or enter the member's name you want to add. Only Azure AD users that exist in the Blockchain Workbench tenant are listed. If the user is not found, you need to add Azure AD users.
6. Select the **Role** for the member. For the first member, select**Requestor** as the role.
7. Select **Add** to add the member with the associated role to the application.
8. Add another member to the application with the **Responder** role.

For more information about managing users in Blockchain Workbench, see managing users in Azure Blockchain Workbench

Next steps

In this how-to article, you've created a basic request and response application. To learn how to use the application, continue to the next how-to article.

Using a blockchain application

Tutorial: Using applications in Azure Blockchain Workbench

4/15/2019 • 2 minutes to read • Edit Online

You can use Blockchain Workbench to create and take actions on contracts. You can also view contract details such as status and transaction history.

You'll learn how to:

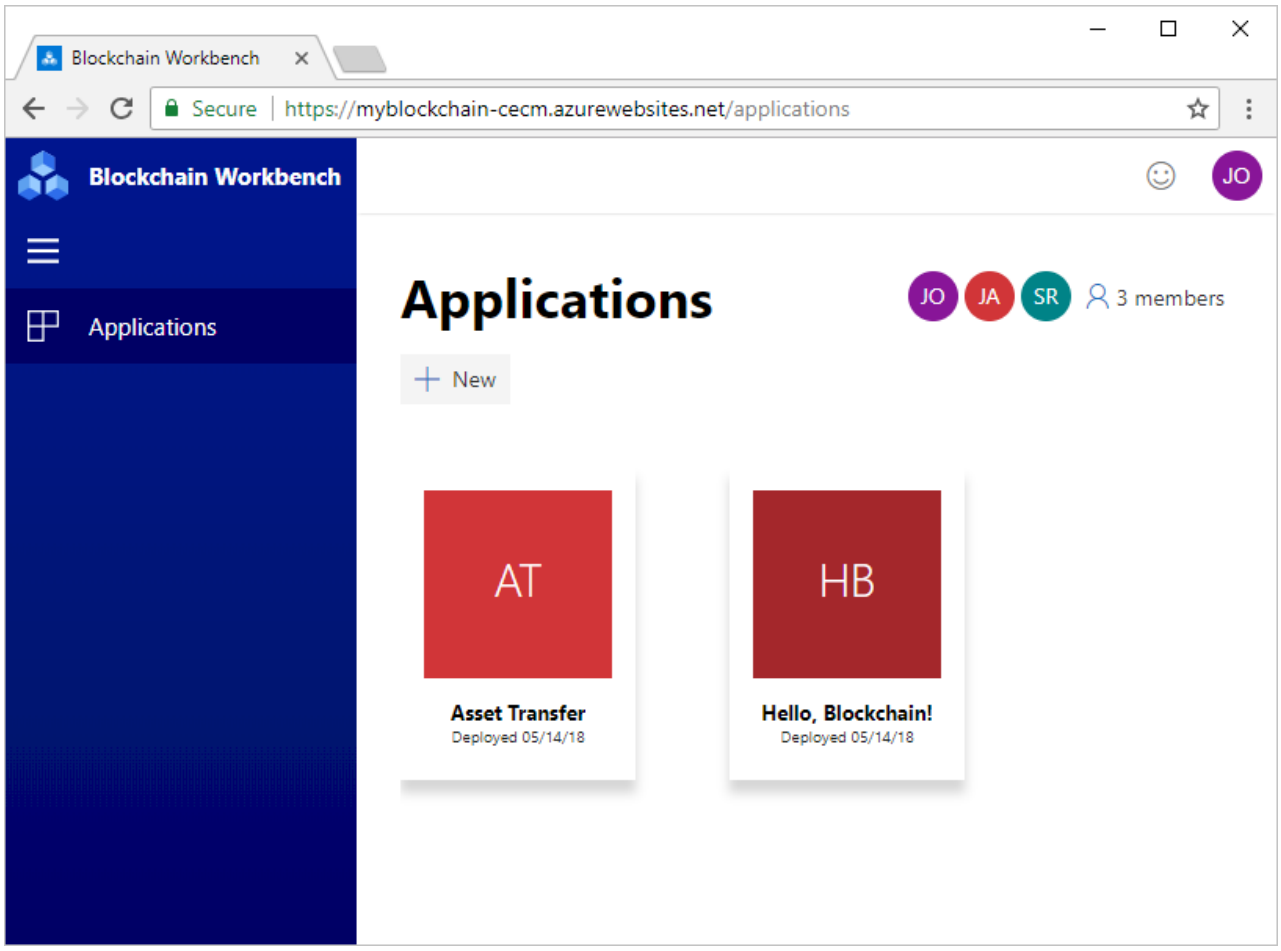
- Create a new contract
- Take an action on a contract

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

Prerequisites

- A Blockchain Workbench deployment. For more information, see Azure Blockchain Workbench deployment for details on deployment
- A deployed blockchain application in Blockchain Workbench. See Create a blockchain application in Azure Blockchain Workbench

Open the Blockchain Workbench in your browser.



You need to sign in as a member of the Blockchain Workbench. If there are no applications listed, you are a member of Blockchain Workbench but not a member of any applications. The Blockchain Workbench

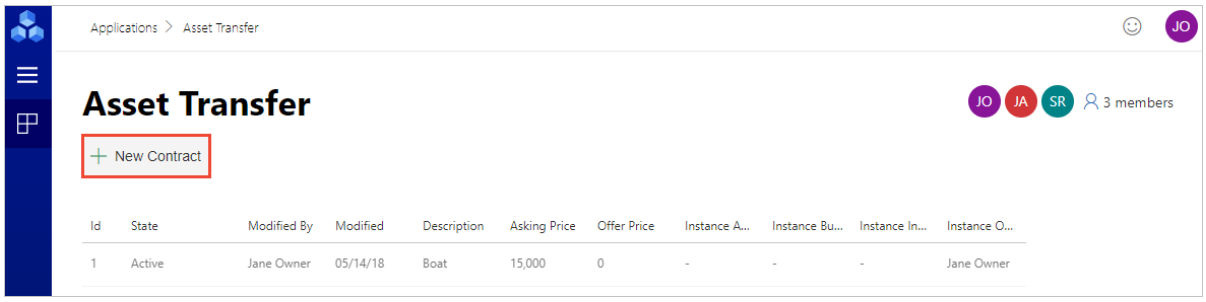
administrator can assign members to applications.

Create new contract

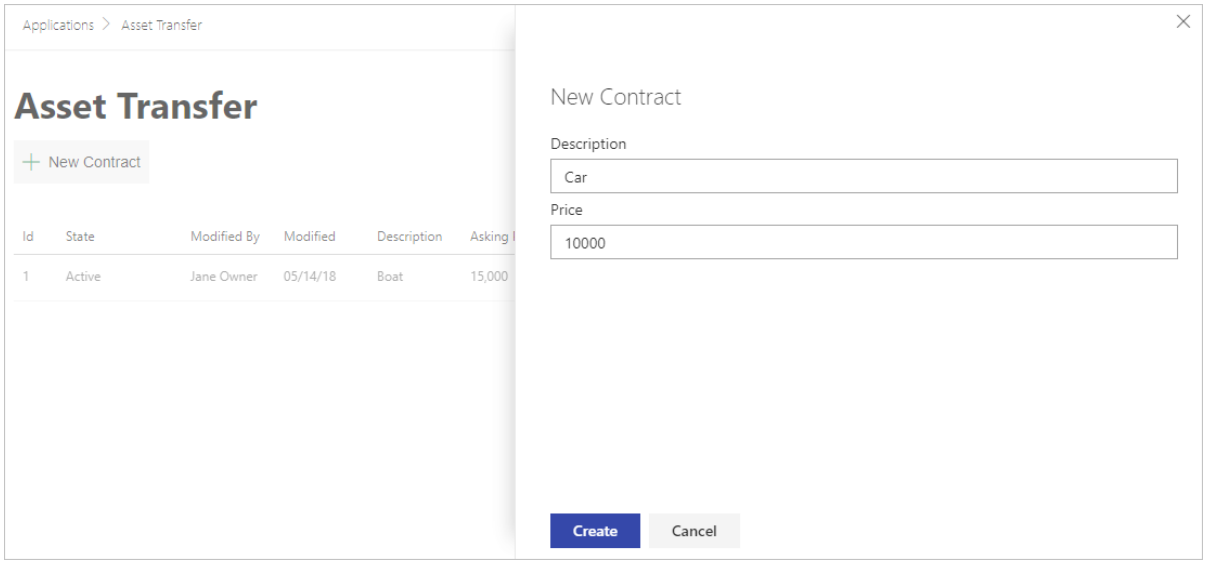
To create a new contract, you need to be a member specified as a contract **initiator**. For information defining application roles and initiators for the contract, see workflows in the configuration overview. For information on assigning members to application roles, see add a member to application.

1. In Blockchain Workbench application section, select the application tile that contains the contract you want to create. A list of active contracts is displayed.

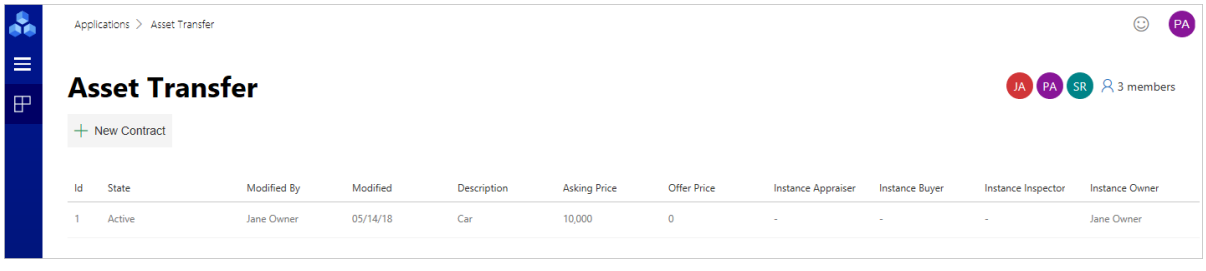
2. To create a new contract, select **New contract**.



3. The **New contract** pane is displayed. Specify the initial parameters values. Select **Create**.



The newly created contract is displayed in the list with the other active contracts.

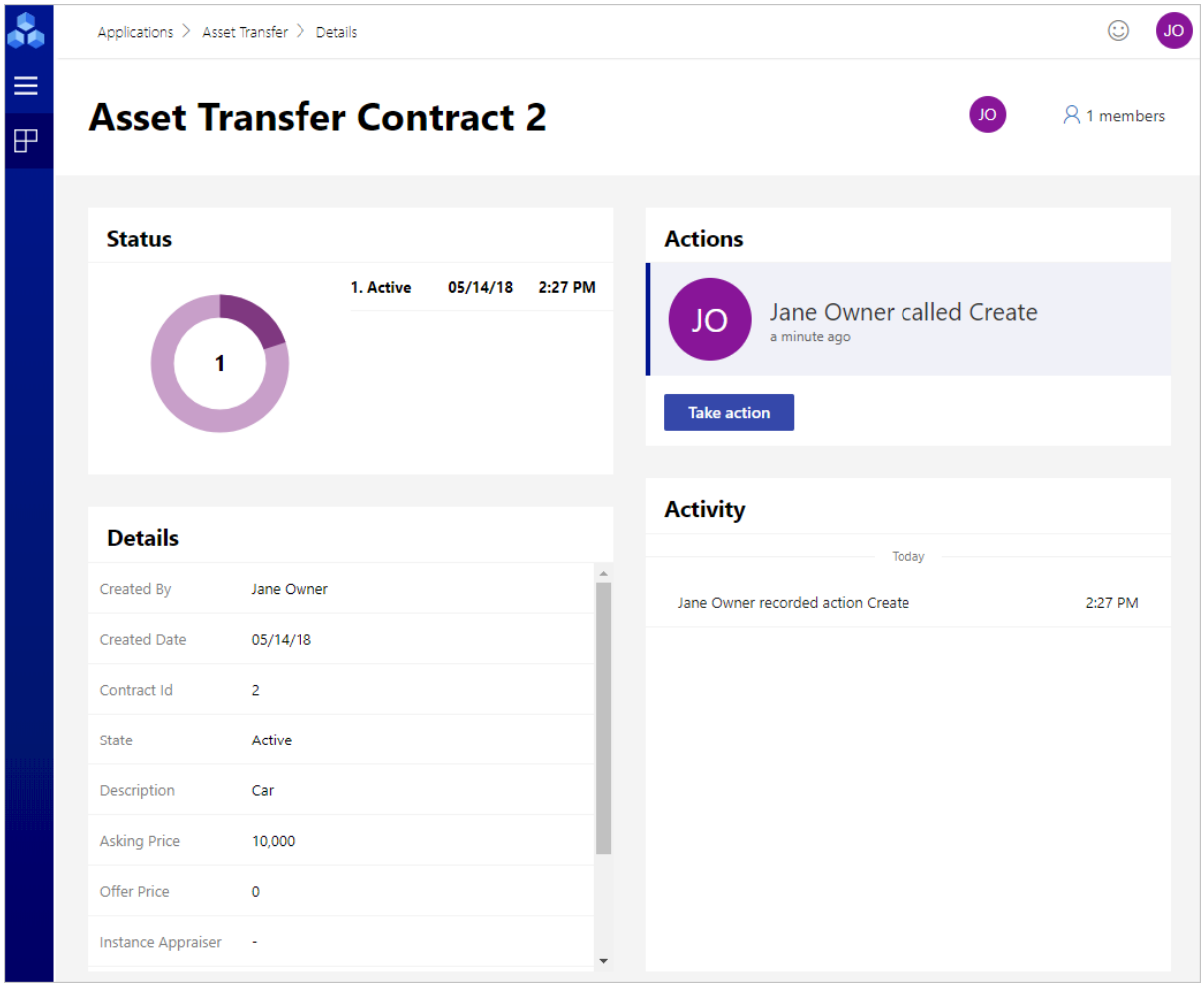


Take action on contract

Depending on the state the contract is in, members can take actions to transition to the next state of the contract. Actions are defined as transitions within a state. Members belonging to an allowed application or instance role for the transition can take the action.

1. In Blockchain Workbench application section, select the application tile that contains the contract to take the action.

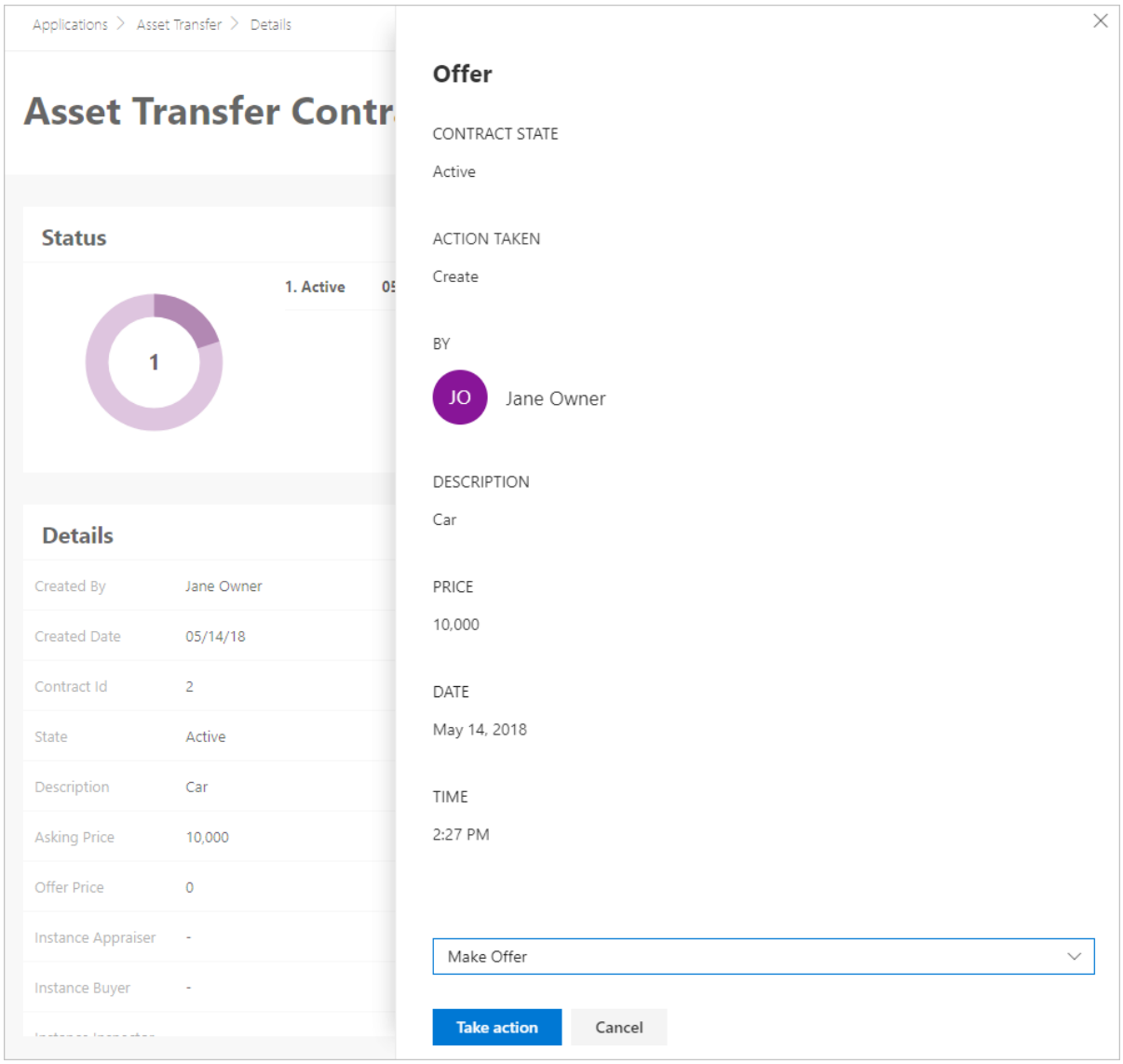
2. Select the contract in the list. Details about the contract are displayed in different sections.



SECTION	DESCRIPTION
Status	Lists the current progress within the contract stages
Details	The current values of the contract
Action	Details about the last action
Activity	Transaction history of the contract

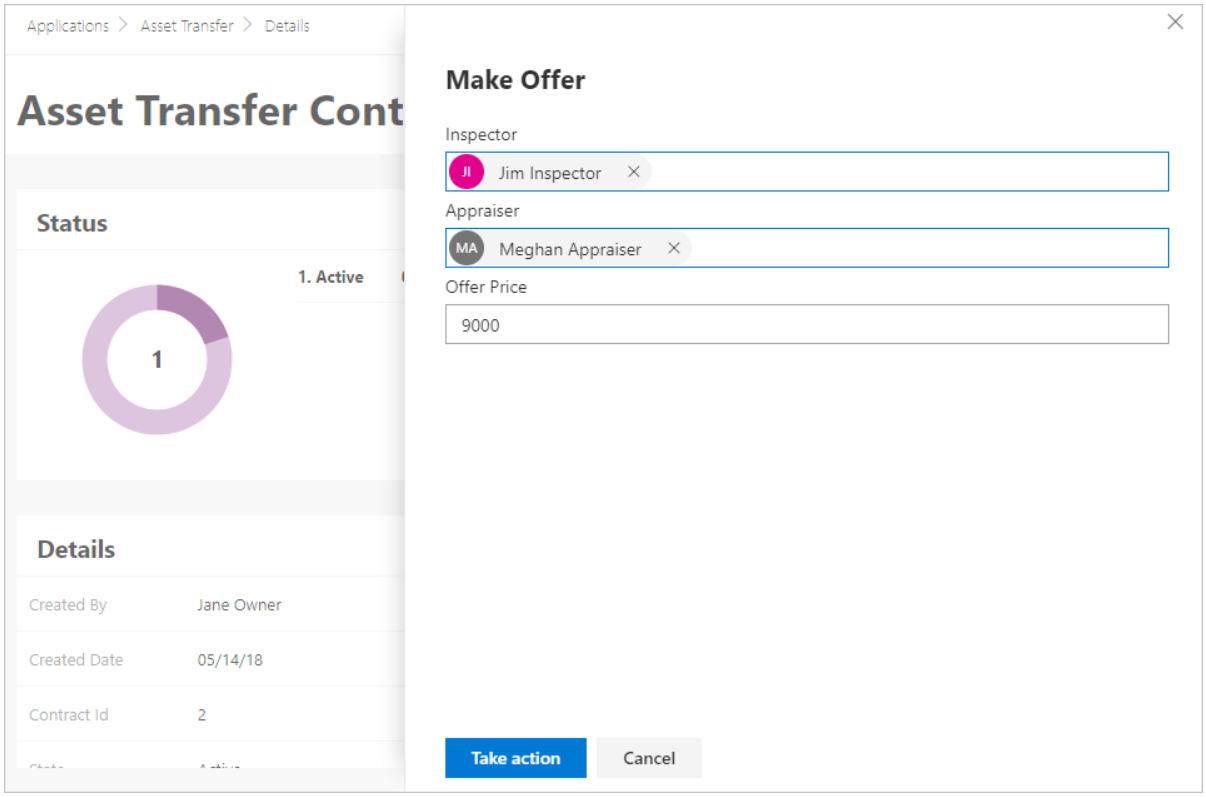
3. In the **Action** section, select **Take action**.

4. The details about the current state of the contract are displayed in a pane. Choose the action you want to take in the drop-down.



5. Select **Take action** to initiate the action.

6. If parameters are required for the action, specify the values for the action.



7. Select **Take action** to execute the action.

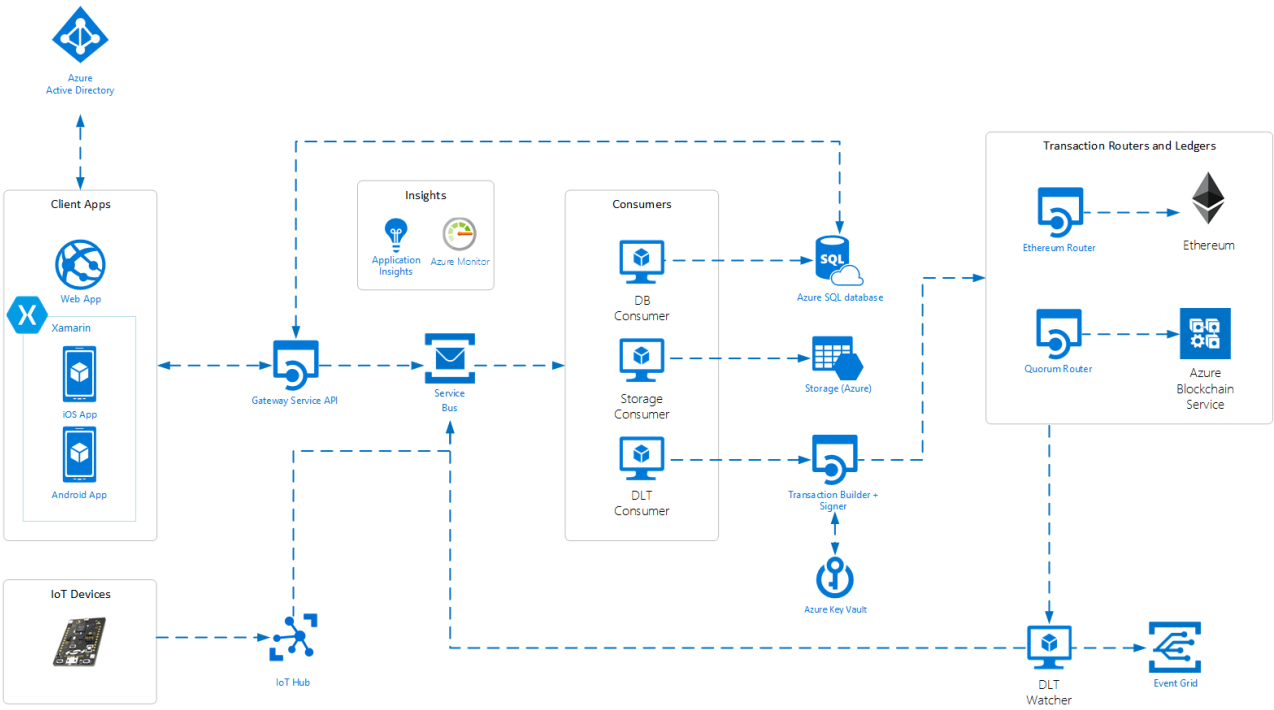
Next steps

Azure Blockchain Workbench application versioning

Azure Blockchain Workbench architecture

6/10/2019 • 7 minutes to read • Edit Online

Azure Blockchain Workbench simplifies blockchain application development by providing a solution using several Azure components. Blockchain Workbench can be deployed using a solution template in the Azure Marketplace. The template allows you to pick modules and components to deploy including blockchain stack, type of client application, and support for IoT integration. Once deployed, Blockchain Workbench provides access to a web app, iOS app, and Android app.



Identity and authentication

Using Blockchain Workbench, a consortium can federate their enterprise identities using Azure Active Directory (Azure AD). Workbench generates new user accounts for on-chain identities with the enterprise identities stored in Azure AD. The identity mapping facilitates authenticated login to client APIs and applications and uses the authentication policies of organizations. Workbench also provides the ability to associate enterprise identities to specific roles within a given smart contract. In addition, Workbench also provides a mechanism to identify the actions those roles can take and at what time.

After Blockchain Workbench is deployed, users interact with Blockchain Workbench either via the client applications, REST-based client API, or Messaging API. In all cases, interactions must be authenticated, either via Azure Active Directory (Azure AD) or device-specific credentials.

Users federate their identities to a consortium Azure AD by sending an email invitation to participants at their email address. When logging in, these users are authenticated using the name, password, and policies. For example, two-factor authentication of their organization.

Azure AD is used to manage all users who have access to Blockchain Workbench. Each device connecting to a smart contract is also associated with Azure AD.

Azure AD is also used to assign users to a special administrator group. Users associated with the administrator group are granted access to rights and actions within Blockchain Workbench including deploying contracts and giving permissions to a user to access a contract. Users outside this group do not have access to administrator

actions.

Client applications

Workbench provides automatically generated client applications for web and mobile (iOS, Android), which can be used to validate, test, and view blockchain applications. The application interface is dynamically generated based on smart contract metadata and can accommodate any use case. The client applications deliver a user-facing front end to the complete blockchain applications generated by Blockchain Workbench. Client applications authenticate users via Azure Active Directory (Azure AD) and then present a user experience tailored to the business context of the smart contract. The user experience enables the creation of new smart contract instances by authorized individuals and then presents the ability to execute certain types of transactions at appropriate points in the business process the smart contract represents.

In the web application, authorized users can access the Administrator Console. The console is available to users in the Administrator group in Azure AD and provides access to the following functionality:

- Deploy Microsoft provided smart contracts for popular scenarios. For example, an asset transfer scenario.
- Upload and deploy their own smart contracts.
- Assign a user access to the smart contract in the context of a specific role.

For more information, see the Azure Blockchain Workbench sample client applications on GitHub.

Gateway service API

Blockchain Workbench includes a REST-based gateway service API. When writing to a blockchain, the API generates and delivers messages to an event broker. When data is requested by the API, queries are sent to the off-chain SQL database. The SQL database contains a replica of on-chain data and metadata that provides context and configuration information for supported smart contracts. Queries return the required data from the off-chain replica in a format informed by the metadata for the contract.

Developers can access the gateway service API to build or integrate blockchain solutions without relying on Blockchain Workbench client apps.

> **NOTE**
> To enable authenticated access to the API, two client applications are registered in Azure Active Directory. Azure Active Directory requires distinct application registrations each application type (native and web).

Message broker for incoming messages

Developers who want to send messages directly to Blockchain Workbench can send messages directly to Service Bus. For example, messages API could be used for system-to-system integration or IoT devices.

Message broker for downstream consumers

During the lifecycle of the application, events occur. Events can be triggered by the Gateway API or on the ledger. Event notifications can initiate downstream code based on the event.

Blockchain Workbench automatically deploys two types of event consumers. One consumer is triggered by blockchain events to populate the off-chain SQL store. The other consumer is to capture metadata for events generated by the API related to the upload and storage of documents.

Message consumers

Message consumers take messages from Service Bus. The underlying eventing model for message consumers allows for extensions of additional services and systems. For example, you could add support to populate CosmosDB or evaluate messages using Azure Streaming Analytics. The following sections describe the message consumers included in Blockchain Workbench.

Distributed ledger consumer

Distributed ledger technology (DLT) messages contain the metadata for transactions to be written to the blockchain. The consumer retrieves the messages and pushes the data to a transaction builder, signer, and router.

Database consumer

The database consumer takes messages from Service Bus and pushes the data to an attached database, such as SQL database.

Storage consumer

The storage consumer takes messages from Service Bus and pushes data to an attached storage. For example, storing hashed documents in Azure Storage.

Transaction builder and signer

If a message on the inbound message broker needs to be written to the blockchain, it will be processed by the DLT consumer. The DLT consumer is a service, which retrieves the message containing metadata for a desired transaction to execute and then sends the information to the *transaction builder and signer*. The *transaction builder and signer* assembles a blockchain transaction based on the data and the desired blockchain destination. Once assembled, the transaction is signed. Private keys are stored in Azure Key Vault.

Blockchain Workbench retrieves the appropriate private key from Key Vault and signs the transaction outside of Key Vault. Once signed, the transaction is sent to transaction routers and ledgers.

Transaction routers and ledgers

Transaction routers and ledgers take signed transactions and route them to the appropriate blockchain. Currently, Blockchain Workbench supports Ethereum as its target blockchain.

DLT watcher

A distributed ledger technology (DLT) watcher monitors events occurring on block chains attached to Blockchain Workbench. Events reflect information relevant to individuals and systems. For example, the creation of new contract instances, execution of transactions, and changes of state. The events are captured and sent to the outbound message broker, so they can be consumed by downstream consumers.

For example, the SQL consumer monitors events, consumes them, and populates the SQL database with the included values. The copy enables recreation of a replica of on-chain data in an off-chain store.

Azure SQL database

The Azure SQL database attached to Blockchain Workbench stores contract definitions, configuration metadata, and a SQL-accessible replica of data stored in the blockchain. This data can easily be queried, visualized, or analyzed by directly accessing the database. Developers and other users can use the database for reporting, analytics, or other data-centric integrations. For example, users can visualize transaction data using Power BI.

This off-chain storage provides the ability for enterprise organizations to query data in SQL rather than in a blockchain ledger. Also, by standardizing on a standard schema that's agnostic of blockchain technology stacks, the off-chain storage enables the reuse of reports and other artifacts across projects, scenarios, and organizations.

Azure Storage

Azure Storage is used to store contracts and metadata associated with contracts.

From purchase orders and bills of lading, to images used in the news and medical imagery, to video originating from a continuum including police body cameras and major motion pictures, documents play a role in many blockchain-centric scenarios. Documents are not appropriate to place directly on the blockchain.

Blockchain Workbench supports the ability to add documents or other media content with blockchain business logic. A hash of the document or media content is stored in the blockchain and the actual document or media content is stored in Azure Storage. The associated transaction information is delivered to the inbound message broker, packaged up, signed, and routed to the blockchain. This process triggers events, which are shared via the outbound message broker. The SQL DB consumes this information and sends it to the DB for later querying. Downstream systems could also consume these events to act as appropriate.

Monitoring

Workbench provides application logging using Application Insights and Azure Monitor. Application Insights is used to store all logged information from Blockchain Workbench and includes errors, warnings, and successful operations. Application Insights can be used by developers to debug issues with Blockchain Workbench.

Azure Monitor provides information on the health of the blockchain network.

Next steps

Deploy Azure Blockchain Workbench

Smart contract integration patterns

5/20/2019 • 14 minutes to read • Edit Online

Smart contracts often represent a business workflow that needs to integrate with external systems and devices.

The requirements of these workflows include a need to initiate transactions on a distributed ledger that include data from an external system, service, or device. They also need to have external systems react to events originating from smart contracts on a distributed ledger.

The REST API and messaging integration sends transactions from external systems to smart contracts included in an Azure Blockchain Workbench application. It also sends event notifications to external systems based on changes that take place within an application.

For data integration scenarios, Azure Blockchain Workbench includes a set of database views that merge a combination of transactional data from the blockchain and meta-data about applications and smart contracts.

In addition, some scenarios, such as those related to supply chain or media, may also require the integration of documents. While Azure Blockchain Workbench does not provide API calls for handling documents directly, documents can be incorporated into a blockchain application. This section also includes that pattern.

This section includes the patterns identified for implementing each of these types of integrations in your end to end solutions.

REST API-based integration

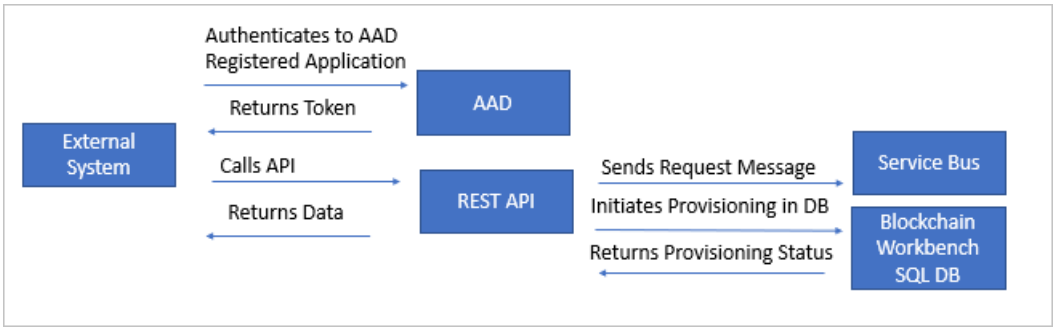
Capabilities within the Azure Blockchain Workbench generated web application are exposed via the REST API. Capabilities include Azure Blockchain Workbench uploading, configuration and administration of applications, sending transactions to a distributed ledger, and the querying of application metadata and ledger data.

The REST API is primarily used for interactive clients such as web, mobile, and bot applications.

This section looks at patterns focused on the aspects of the REST API that send transactions to a distributed ledger and patterns that query data about transactions from Azure Blockchain Workbench's *off chain* SQL database.

Sending transactions to a distributed ledger from an external system

The Azure Blockchain Workbench REST API sends authenticated requests to execute transactions on a distributed ledger.



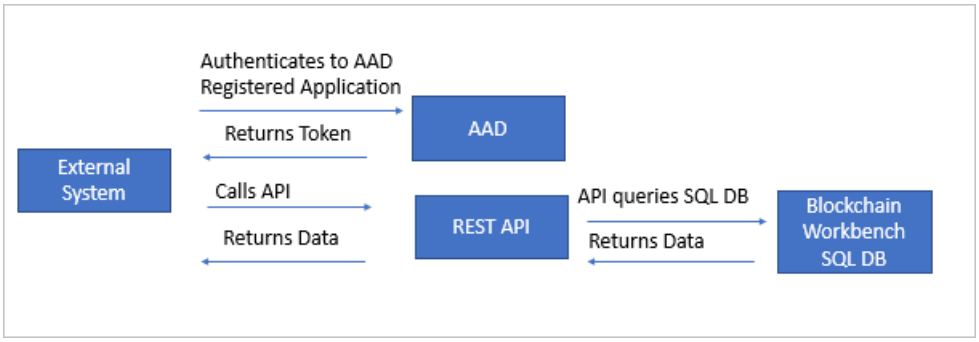
Executing transactions occurs using the process depicted previously, where:

- The external application authenticates to the Azure Active Directory provisioned as part of the Azure Blockchain Workbench deployment.
- Authorized users receive a bearer token that can be sent with requests to the API.
- External applications make calls to the REST API using the bearer token.

- The REST API packages the request as a message and sends it to the Service Bus. From here it is retrieved, signed, and sent to the appropriate distributed ledger.
- The REST API makes a request to the Azure Blockchain Workbench SQL DB to record the request and establish the current provisioning status.
- The SQL DB returns the provisioning status and the API call returns the ID to the external application that called it.

Querying Blockchain Workbench metadata and distributed ledger transactions

The Azure Blockchain Workbench REST API sends authenticated requests to query details related to smart contract execution on a distributed ledger.



Querying occurs using the process depicted previously, where:

1. The external application authenticates to the Azure Active Directory provisioned as part of the Azure Blockchain Workbench deployment.
2. Authorized users receive a bearer token that can be sent with requests to the API.
3. External applications make calls to the REST API using the bearer token.
4. The REST API queries the data for the request from the SQL DB and returns it to the client.

Messaging integration

Messaging integration facilitates interaction with systems, services, and devices where an interactive sign-in is not possible or desirable. Messaging integration focuses on two types of messages: messages requesting transactions be executed on a distributed ledger, and events exposed by that ledger when transactions have taken place.

Messaging integration focuses on the execution and monitoring of transactions related to user creation, contract creation, and execution of transactions on contracts and is primarily used by *headless* back-end systems.

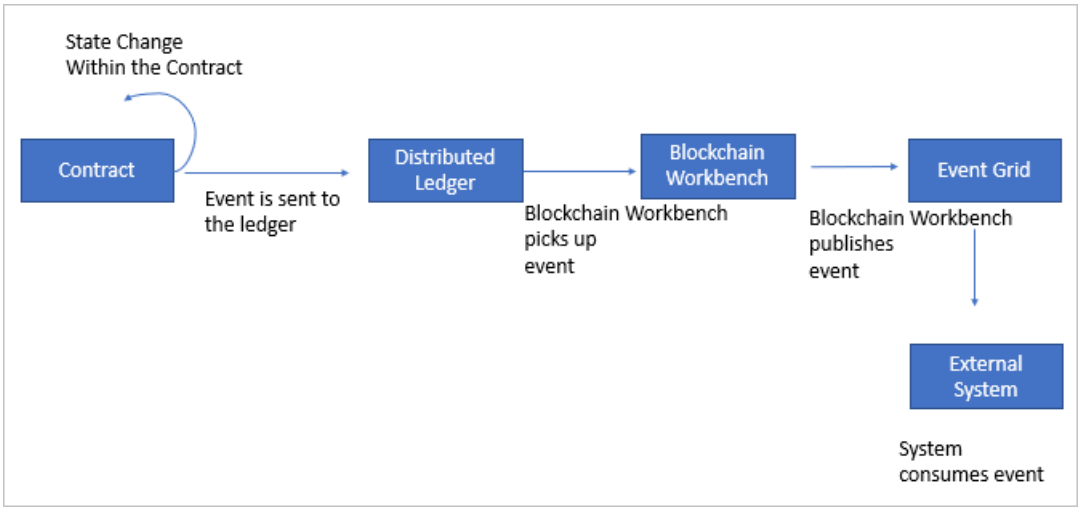
This section looks at patterns focused on the aspects of the message-based API that send transactions to a distributed ledger and patterns that represent event messages exposed by the underlying distributed ledger.

One-way event delivery from a smart contract to an event consumer

In this scenario, an event occurs within a smart contract, for example, a state change or the execution of a specific type of transaction. This event is broadcast via an Event Grid to downstream consumers, and those consumers then take appropriate actions.

An example of this scenario is that when a transaction occurs, a consumer would be alerted and could take action, such as recording the information in a SQL DB or the Common Data Service. This scenario is the same pattern that Workbench follows to populate its *off chain* SQL DB.

Another would be if a smart contract transitions to a particular state, for example when a contract goes into an *OutOfCompliance*. When this state change happens, it could trigger an alert to be sent to an administrator's mobile phone.



This scenario occurs using the process depicted previously, where:

- The smart contract transitions to a new state and sends an event to the ledger.
- The ledger receives and delivers the event to Azure Blockchain Workbench.
- Azure Blockchain Workbench is subscribed to events from the ledger and receives the event.
- Azure Blockchain Workbench publishes the event to subscribers on the Event Grid.
- External systems are subscribed to the Event Grid, consume the message, and take the appropriate actions.

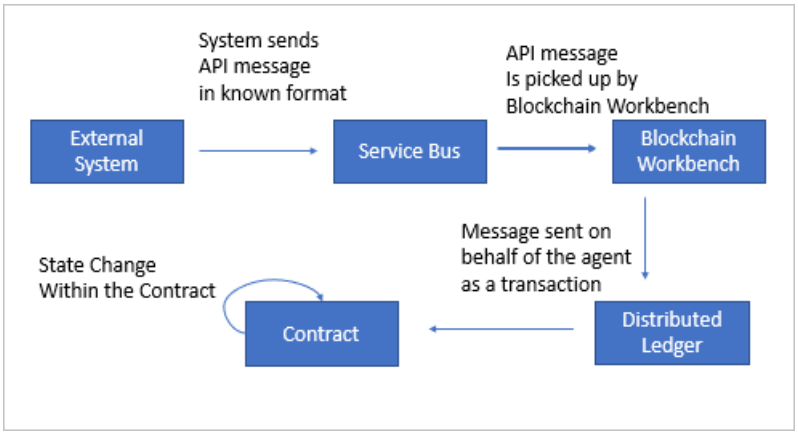
One-way event delivery of a message from an external system to a smart contract

There is also a scenario that flows from the opposite direction. In this case, an event is generated by a sensor or an external system and the data from that event should be sent to a smart contract.

A common example is the delivery of data from financial markets, for example, prices of commodities, stock, or bonds, to a smart contract.

Direct delivery of an Azure Blockchain Workbench in the expected format

Some applications are built to integrate with Azure Blockchain Workbench and directly generates and send messages in the expected formats.



This delivery occurs using the process depicted previously, where:

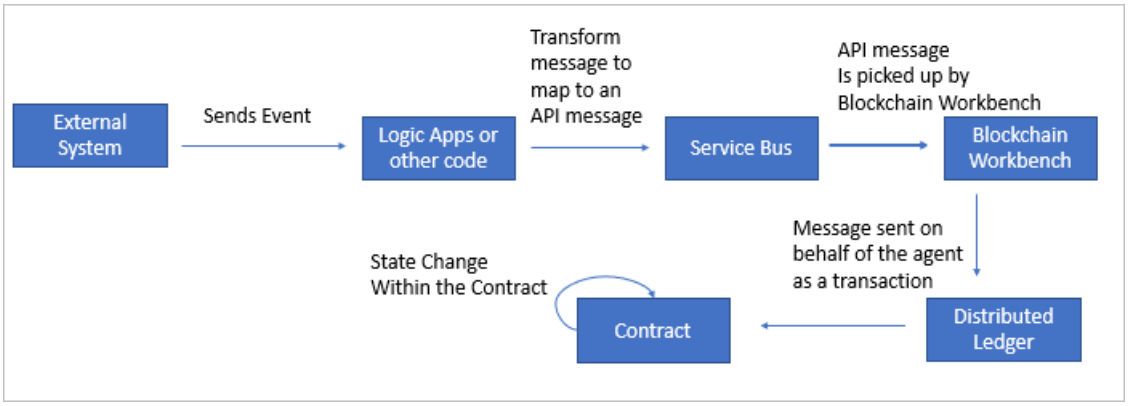
- An event occurs in an external system that triggers the creation of a message for Azure Blockchain Workbench.
- The external system has code written to create this message in a known format and sends it directly to the Service Bus.
- Azure Blockchain Workbench is subscribed to events from the Service Bus and retrieves the message.
- Azure Blockchain Workbench initiates a call to the ledger, sending data from the external system to a specific

contract.

- Upon receipt of the message, the contract transitions to a new state.

Delivery of a message in a format unknown to Azure Blockchain Workbench

Some systems cannot be modified to deliver messages in the standard formats used by Azure Blockchain Workbench. In these cases, existing mechanisms and message formats from these systems can often be used. Specifically, the native message types of these systems can be transformed using Logic Apps, Azure Functions, or other custom code to map to one of the standard messaging formats expected.



This occurs using the process depicted previously, where:

- An event occurs in an external system that triggers the creation of a message.
- A Logic App or custom code is used to receive that message and transform it to a standard Azure Blockchain Workbench formatted message.
- The Logic App sends the transformed message directly to the Service Bus.
- Azure Blockchain Workbench is subscribed to events from the Service Bus and retrieves the message.
- Azure Blockchain Workbench initiates a call to the ledger, sending data from the external system to a specific function on the contract.
- The function executes and typically modifies the state. The change of state moves forward the business workflow reflected in the smart contract, enabling other functions to now be executed as appropriate.

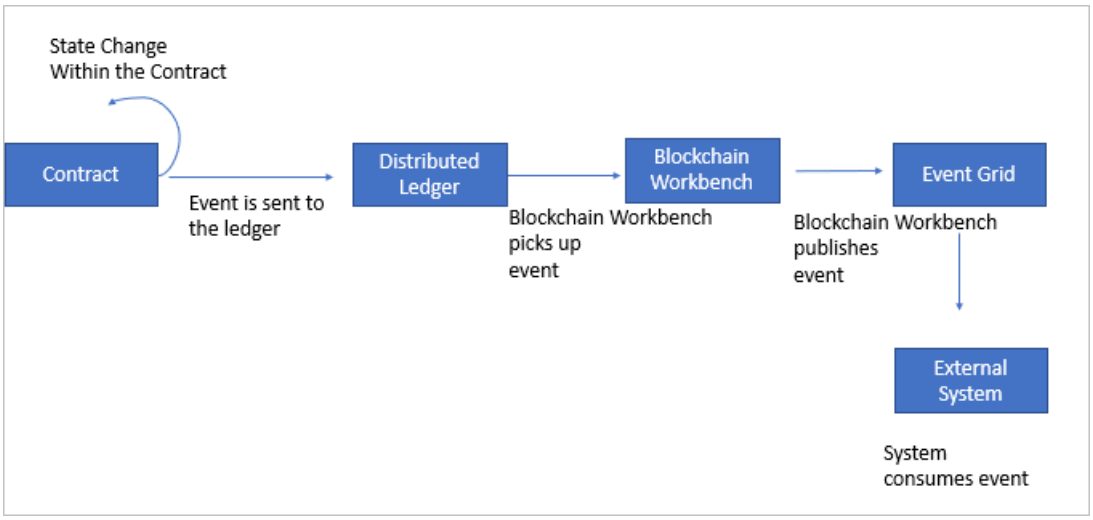
Transitioning control to an external process and await completion

There are scenarios where a smart contract must stop internal execution and hand off to an external process. That external process would then complete, send a message to the smart contract, and execution would then continue within the smart contract.

Transition to the external process

This pattern is typically implemented using the following approach:

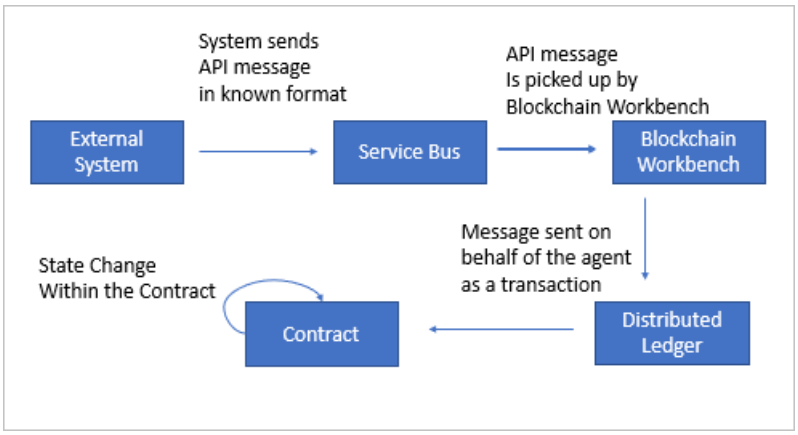
- The smart contract transitions to a specific state. In this state, either no or a limited number of functions can be executed until an external system takes a desired action.
- The change of state is surfaced as an event to a downstream consumer.
- The downstream consumer receives the event and triggers external code execution.



Return of control from the smart contract

Depending on the ability to customize the external system, it may or may not be able to deliver messages in one of the standard formats that Azure Blockchain Workbench expects. Based on the external systems ability to generate one of these messages determine which of the following two return paths is taken.

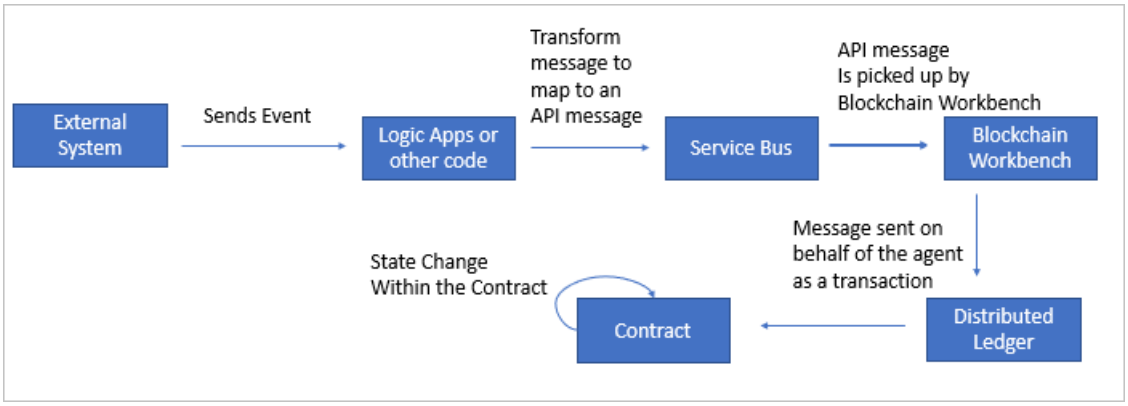
Direct delivery of an Azure Blockchain Workbench in the expected format



In this model, the communication to the contract and subsequent state change occurs following the previous process where -

- Upon reaching the completion or a specific milestone in the external code execution, an event is sent to the Service Bus connected to Azure Blockchain Workbench.
- For systems that can't be directly adapted to write a message that conforms to the expectations of the API, it is transformed.
- The content of the message is packaged up and sent to a specific function on the smart contract. This delivery is done on behalf of the user associated with the external system.
- The function executes and typically modifies the state. The change of state moves forward the business workflow reflected in the smart contract, enabling other functions to now be executed as appropriate.

Delivery of a message in a format unknown to Azure Blockchain Workbench



In this model where a message in a standard format cannot be sent directly, the communication to the contract and subsequent state change occurs following the previous process where:

1. Upon reaching the completion or a specific milestone in the external code execution, an event is sent to the Service Bus connected to Azure Blockchain Workbench.
2. A Logic App or custom code is used to receive that message and transform it to a standard Azure Blockchain Workbench formatted message.
3. The Logic App sends the transformed message directly to the Service Bus.
4. Azure Blockchain Workbench is subscribed to events from the Service Bus and retrieves the message.
5. Azure Blockchain Workbench initiates a call to the ledger, sending data from the external system to a specific contract.
6. The content of the message is packaged up and sent to a specific function on the smart contract. This delivery is done on behalf of the user associated with the external system.
7. The function executes and typically modifies the state. The change of state moves forward the business workflow reflected in the smart contract, enabling other functions to now be executed as appropriate.

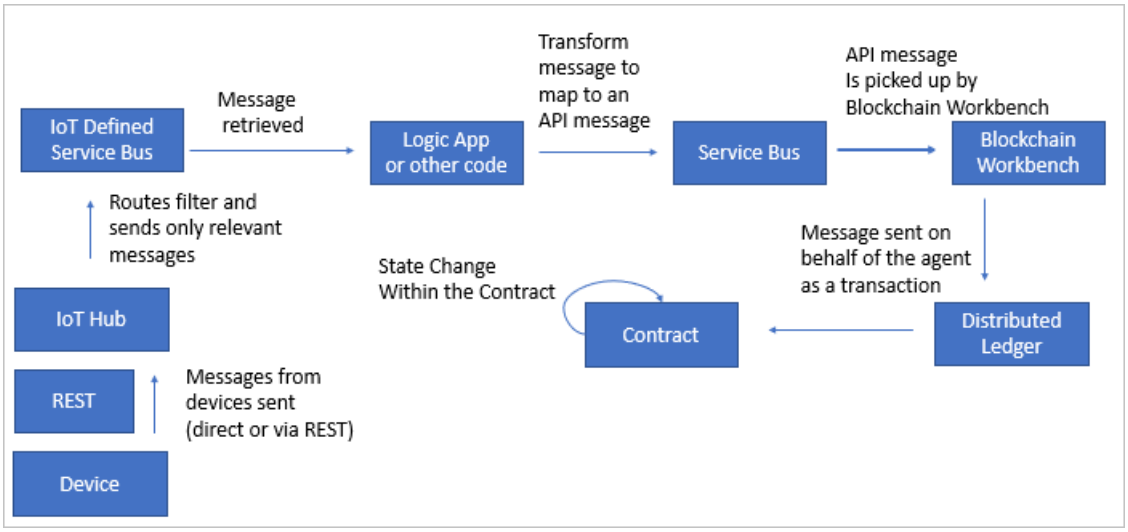
IoT integration

A common integration scenario is the inclusion of telemetry data retrieved from sensors in a smart contract. Based on data delivered by sensors, smart contracts could take informed actions and alter the state of the contract.

For example, if a truck delivering medicine had its temperature soar to 110 degrees, it may impact the effectiveness of the medicine and may cause a public safety issue if not detected and removed from the supply chain. If a driver accelerated their car to 100 miles per hour, the resulting sensor information could trigger a cancellation of insurance by their insurance provider. If the car was a rental car, GPS data could indicate when the driver went outside a geography covered by their rental agreement and charge a penalty.

The challenge is that these sensors can be delivering data on a constant basis and it is not appropriate to send all of this data to a smart contract. A typical approach is to limit the number of messages sent to the blockchain while delivering all messages to a secondary store. For example, deliver messages received at only fixed interval, for example, once per hour, and when a contained value falls outside of an agreed upon range for a smart contract. Checking values that fall outside of tolerances, ensures that the data relevant to the contracts business logic is received and executed. Checking the value at the interval confirms that the sensor is still reporting. All data is sent to a secondary reporting store to enable broader reporting, analytics, and machine learning. For example, while getting sensor readings for GPS may not be required every minute for a smart contract, they could provide interesting data to be used in reports or mapping routes.

On the Azure platform, integration with devices is typically done with IoT Hub. IoT Hub provides routing of messages based on content, and enables the type of functionality described previously.

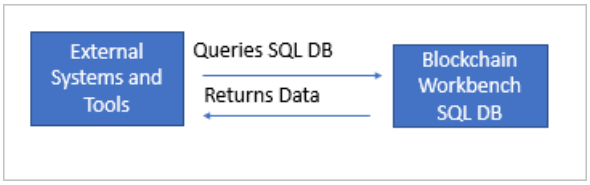


The process depicts a pattern:

- A device communicates directly or via a field gateway to IoT Hub.
- IoT Hub receives the messages and evaluates the messages against routes established that check the content of the message, for example. *Does the sensor report a temperature greater than 50 degrees?*
- The IoT Hub sends messages that meet the criteria to a defined Service Bus for the route.
- A Logic App or other code listens to the Service Bus that IoT Hub has established for the route.
- The Logic App or other code retrieves and transform the message to a known format.
- The transformed message, now in a standard format, is sent to the Service Bus for Azure Blockchain Workbench.
- Azure Blockchain Workbench is subscribed to events from the Service Bus and retrieves the message.
- Azure Blockchain Workbench initiates a call to the ledger, sending data from the external system to a specific contract.
- Upon receipt of the message, the contract evaluates the data and may change the state based on the outcome of that evaluation, for example, for a high temperature, change the state to *Out of Compliance*.

Data integration

In addition to REST and message-based API, Azure Blockchain Workbench also provides access to a SQL DB populated with application and contract meta-data as well as transactional data from distributed ledgers.

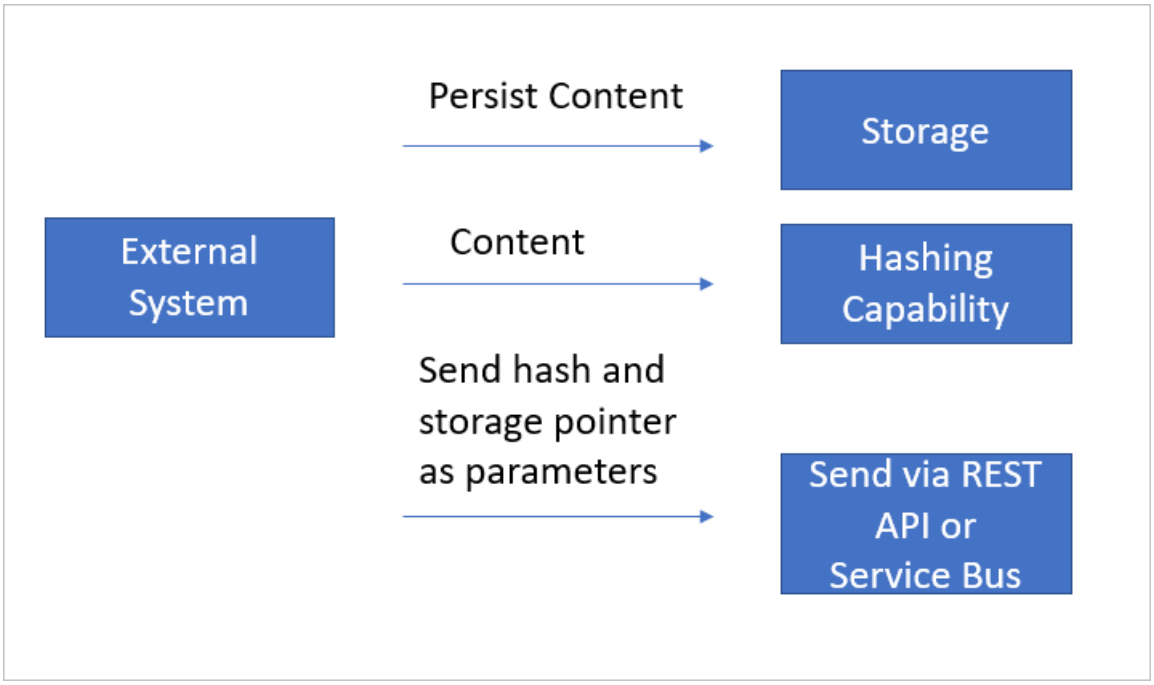


The data integration is well known:

- Azure Blockchain Workbench stores metadata about applications, workflows, contracts, and transactions as part of its normal operating behavior.
- External systems or tools provide one or more dialogs to facilitate the collection of information about the database, such as database server name, database name, type of authentication, login credentials, and which database views to utilize.
- Queries are written against SQL database views to facilitate downstream consumption by external systems, services, reporting, developer tools, and enterprise productivity tools.

Storage integration

Many scenarios may require the need to incorporate attestable files. For multiple reasons, it is inappropriate to put files on a blockchain. Instead, a common approach is to perform a cryptographic hash (for example, SHA-256) against a file and share that hash on a distributed ledger. Performing the hash again at any future time should return the same result. If the file is modified, even if just one pixel is modified in an image, the hash returns a different value.



The pattern can be implemented where:

- An external system persists a file in a storage mechanism, such as Azure Storage.
- A hash is generated with the file or the file and associated metadata such as an identifier for the owner, the URL where the file is located, etc.
- The hash and any metadata is sent to a function on a smart contract, such as *FileAdded*
- In future, the file and meta-data can be hashed again and compared against the values stored on the ledger.

Prerequisites for implementing integration patterns using the REST and message APIs

To facilitate the ability for an external system or device to interact with the smart contract using either the REST or message API, the following must occur -

1. In the Azure Active Directory for the consortium, an account is created that represents the external system or device.
2. One or more appropriate smart contracts for your Azure Blockchain Workbench application have functions defined to accept the events from your external system or device.
3. The application configuration file for your smart contract contains the role, which the system or device is assigned.
4. The application configuration file for your smart contract identifies in which states this function is called by the defined role.
5. The Application configuration file and its smart contracts are uploaded to Azure Blockchain Workbench.

Once the application is uploaded, the Azure Active Directory account for the external system is assigned to the contract and the associated role.

Testing External System Integration Flows Prior to Writing Integration

Code

Integrating with external systems is a key requirement of many scenarios. It is desirable to be able to validate smart contract design prior or in parallel to the development of code to integrate with external systems.

The use of Azure Active Directory (Azure AD) can greatly accelerate developer productivity and time to value. Specifically, the code integration with an external system may take a non-trivial amount of time. By using Azure AD and the auto-generation of UX by Azure Blockchain Workbench, you can allow developers to sign in to Blockchain Workbench as the external system and populate values from the external system via the UX. You can rapidly develop and validate ideas in a proof of concept environment before integration code is written for the external systems.

Deploy Azure Blockchain Workbench

5/6/2019 • 11 minutes to read • Edit Online

Azure Blockchain Workbench is deployed using a solution template in the Azure Marketplace. The template simplifies the deployment of components needed to create blockchain applications. Once deployed, Blockchain Workbench provides access to client apps to create and manage users and blockchain applications.

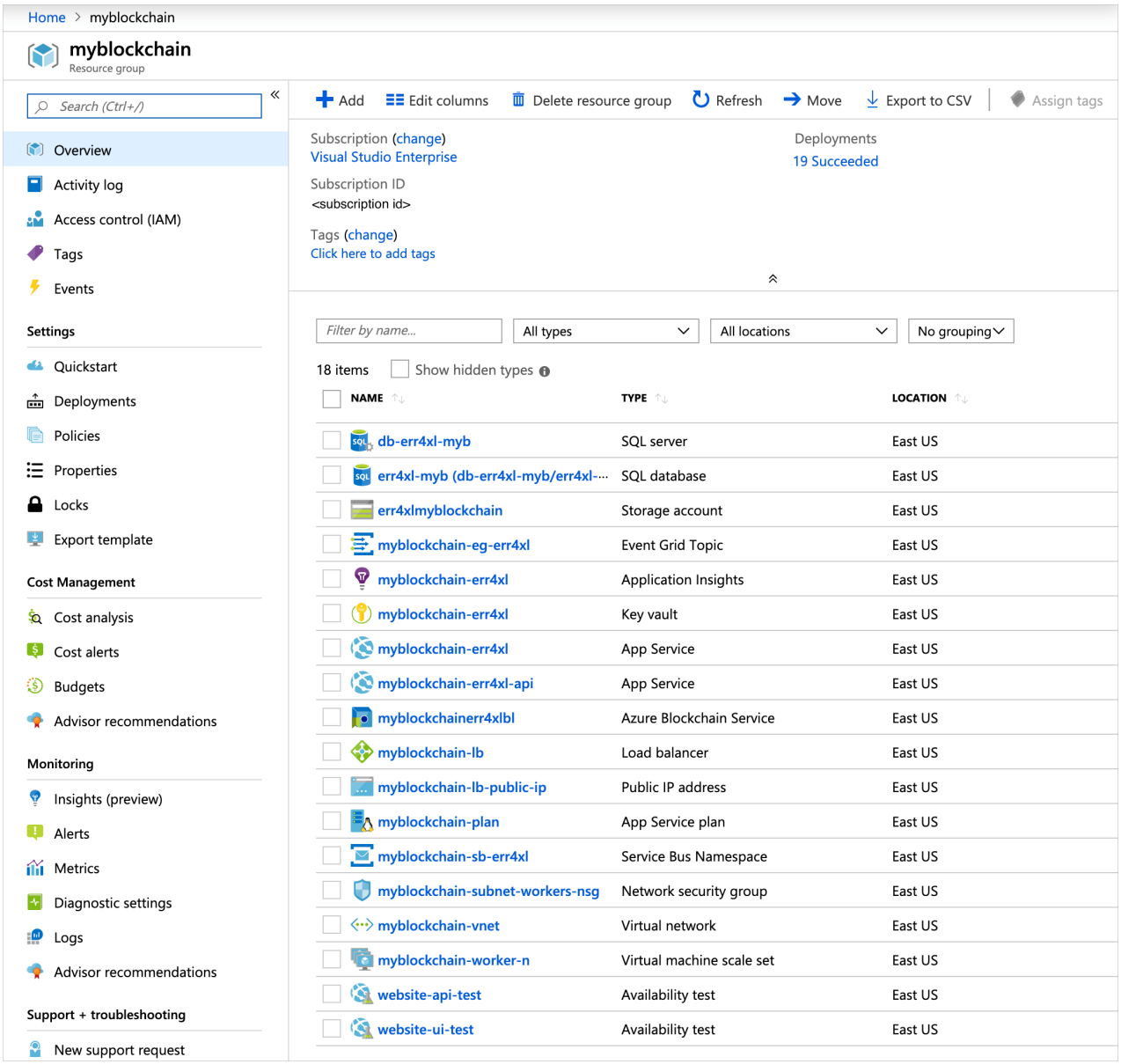
For more information about the components of Blockchain Workbench, see Azure Blockchain Workbench architecture.

Prepare for deployment

Blockchain Workbench allows you to deploy a blockchain ledger along with a set of relevant Azure services most often used to build a blockchain-based application. Deploying Blockchain Workbench results in the following Azure services being provisioned within a resource group in your Azure subscription.

- App Service Plan (Standard)
- Application Insights
- Event Grid
- Azure Key Vault
- Service Bus
- SQL Database (Standard S0) + SQL Logical Server
- Azure Storage account (Standard LRS)
- Virtual machine scale set with capacity of 1
- Virtual Network resource group (with Load Balancer, Network Security Group, Public IP Address, Virtual Network)
- Optional: Azure Blockchain Service (Basic B0 default)

The following is an example deployment created in **myblockchain** resource group.



The cost of Blockchain Workbench is an aggregate of the cost of the underlying Azure services. Pricing information for Azure services can be calculated using the pricing calculator.

Prerequisites

Azure Blockchain Workbench requires Azure AD configuration and application registrations. You can choose to do the Azure AD configurations manually before deployment or run a script post deployment. If you are redeploying Blockchain Workbench, see Azure AD configuration to verify your Azure AD configuration.

> **IMPORTANT**
>
> Workbench does not have to be deployed in the same tenant as the one you are using to register an Azure AD application. Workbench must be deployed in a tenant where you have sufficient permissions to deploy resources. For more information on Azure AD tenants, see How to get an Active Directory tenant and Integrating applications with Azure Active Directory.

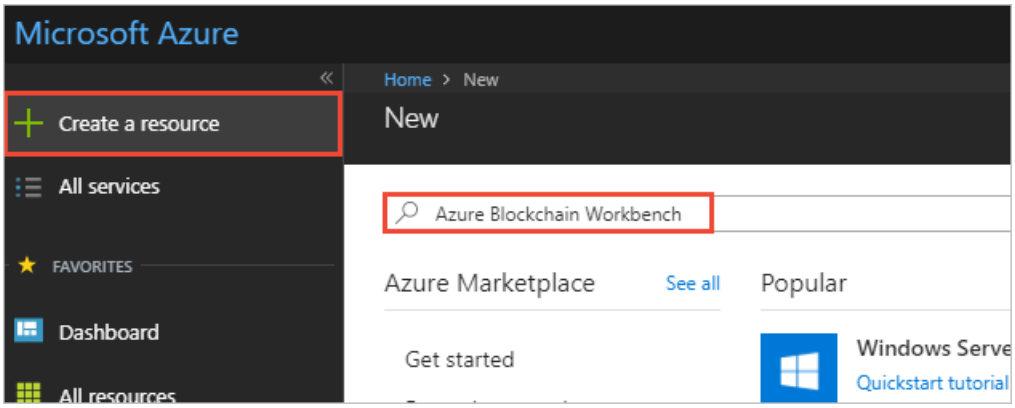
Deploy Blockchain Workbench

Once the prerequisite steps have been completed, you are ready to deploy the Blockchain Workbench. The following sections outline how to deploy the framework.

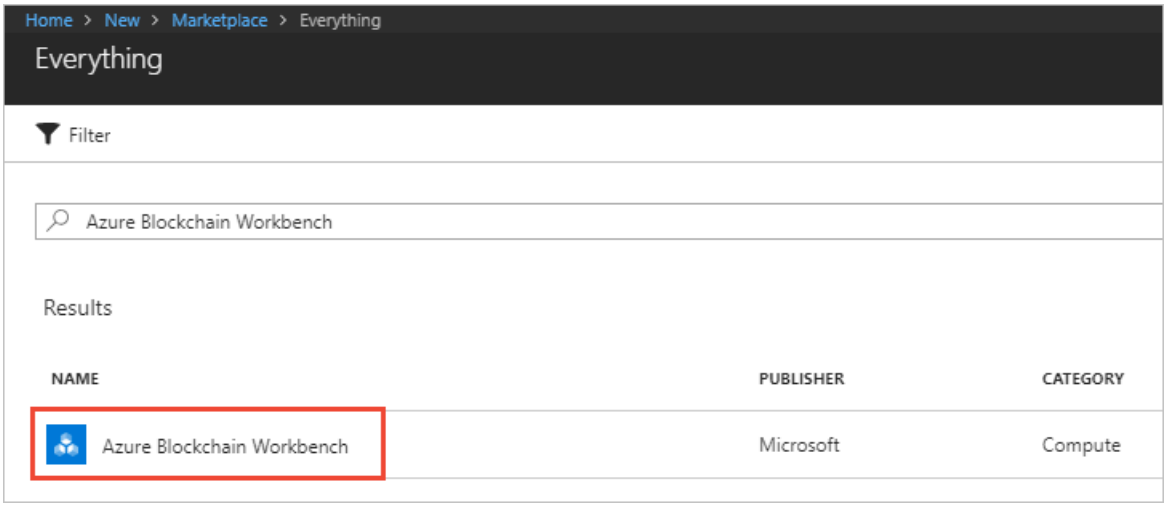
1. Sign in to the Azure portal.

2. Select your account in the top-right corner, and switch to the desired Azure AD tenant where you want to deploy Azure Blockchain Workbench.

3. In the left pane, select **Create a resource**. Search for `Azure Blockchain Workbench` in the **Search the Marketplace** search bar.

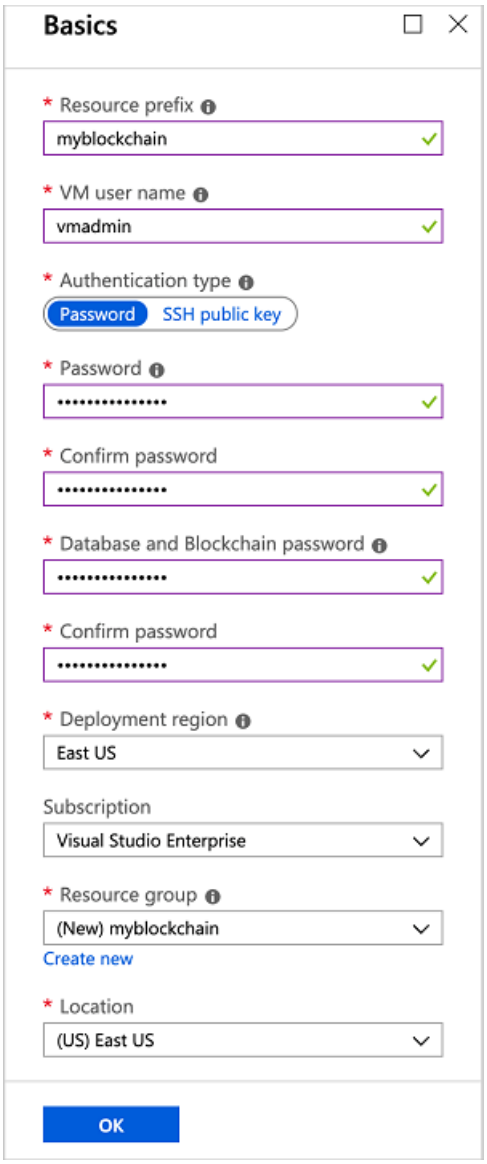


4. Select **Azure Blockchain Workbench**.



5. Select **Create**.

6. Complete the basic settings.



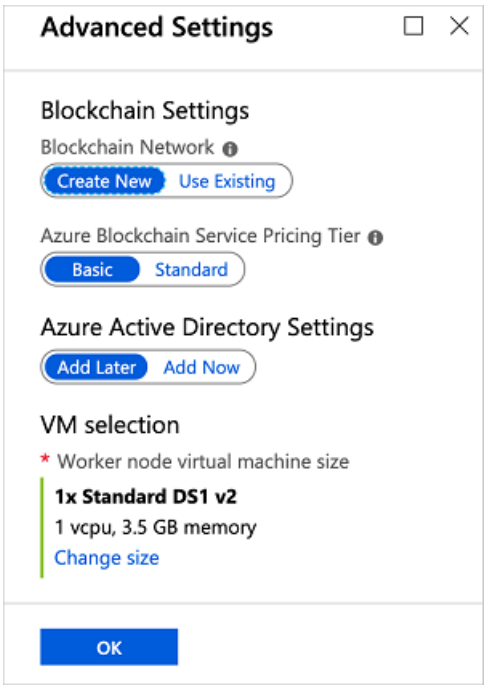
SETTING	DESCRIPTION
Resource prefix	Short unique identifier for your deployment. This value is used as a base for naming resources.
VM user name	The user name is used as administrator for all virtual machines (VM).
Authentication type	Select if you want to use a password or key for connecting to VMs.
Password	The password is used for connecting to VMs.
SSH	Use an RSA public key in the single-line format beginning with **ssh-rsa** or use the multi-line PEM format. You can generate SSH keys using `ssh-keygen` on Linux and OS X, or by using PuTTYGen on Windows. More information on SSH keys, see How to use SSH keys with Windows on Azure.

SETTING	DESCRIPTION
Database and Blockchain password	Specify the password to use for access to the database created as part of the deployment. 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(;)
Deployment region	Specify where to deploy Blockchain Workbench resources. For best availability, this should match the **Location** setting.
Subscription	Specify the Azure Subscription you wish to use for your deployment.
Resource groups	Create a new Resource group by selecting **Create new** and specify a unique resource group name.
Location	Specify the region you wish to deploy the framework.

7. Select **OK** to finish the basic setting configuration section.
8. In **Advanced Settings**, choose if you want to create a new blockchain network or use an existing proof-of-authority blockchain network.

   For **Create new**:

   The *create new* option deploys an Azure Blockchain Service Quorum ledger with the default basic sku.



SETTING	DESCRIPTION
Azure Blockchain Service pricing tier	Choose **Basic** or **Standard** Azure Blockchain Service tier that is used for Blockchain Workbench

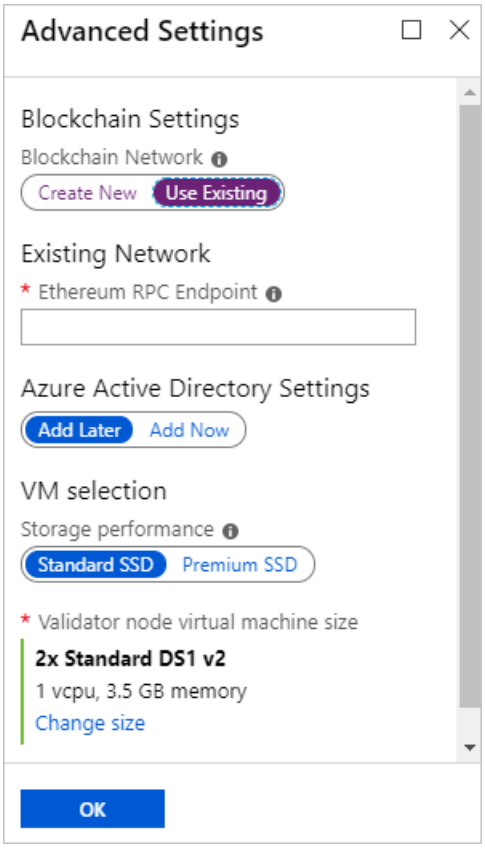
SETTING	DESCRIPTION
Azure Active Directory settings	Choose **Add Later**. Note: If you chose to pre-configure Azure AD or are redeploying, choose to *Add Now*.
VM selection	Select preferred storage performance and VM size for your blockchain network. Choose a smaller VM size such as *Standard DS1 v2* if you are on a subscription with low service limits like Azure free tier.

For **Use existing**:

The *use existing* option allows you to specify an Ethereum Proof-of-Authority (PoA) blockchain network. Endpoints have the following requirements.

- The endpoint must be an Ethereum Proof-of-Authority (PoA) blockchain network.
- The endpoint must be publicly accessible over the network.
- The PoA blockchain network should be configured to have gas price set to zero.

> **NOTE**
> Blockchain Workbench accounts are not funded. If funds are required, the transactions fail.

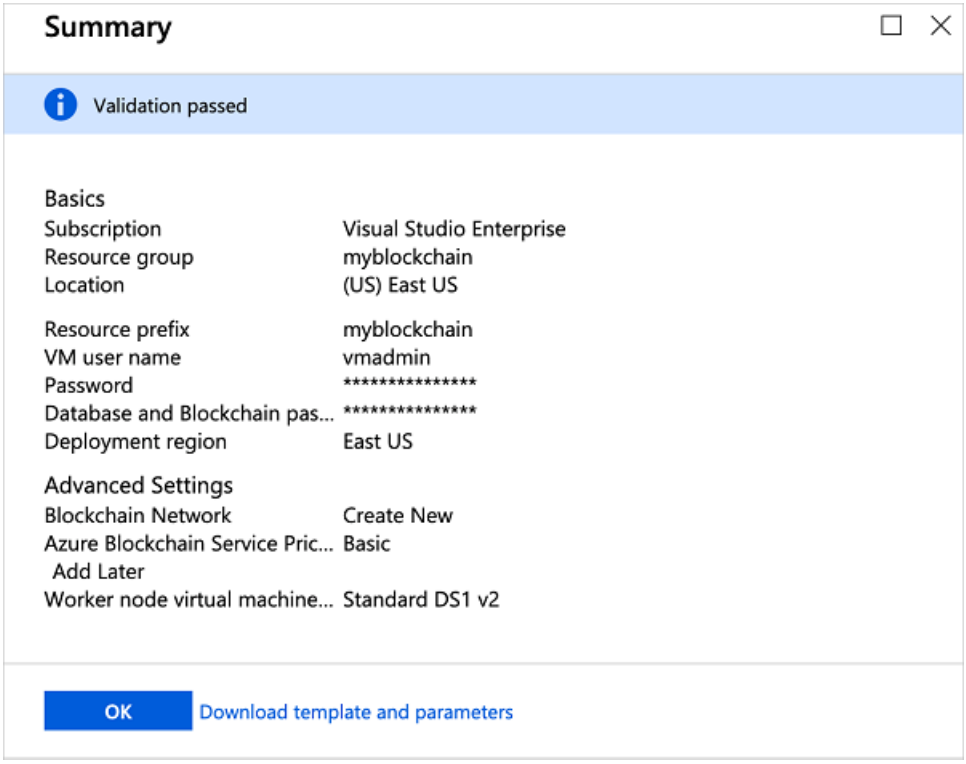


SETTING	DESCRIPTION
Ethereum RPC Endpoint	Provide the RPC endpoint of an existing PoA blockchain network. The endpoint starts with https:// or http:// and ends with a port number. For example, `http<s>://<network-url>:<port>`

SETTING	DESCRIPTION
Azure Active Directory settings	Choose **Add Later**. Note: If you chose to pre-configure Azure AD or are redeploying, choose to *Add Now*.
VM selection	Select preferred storage performance and VM size for your blockchain network. Choose a smaller VM size such as *Standard DS1 v2* if you are on a subscription with low service limits like Azure free tier.

9. Select **OK** to finish Advanced Settings.

10. Review the summary to verify your parameters are accurate.



11. Select **Create** to agree to the terms and deploy your Azure Blockchain Workbench.

The deployment can take up to 90 minutes. You can use the Azure portal to monitor progress. In the newly created resource group, select **Deployments > Overview** to see the status of the deployed artifacts.

> **IMPORTANT**
> Post deployment, you need to complete Active Directory settings. If you chose **Add Later**, you need to run the Azure AD configuration script. If you chose **Add now**, you need to configure the Reply URL.

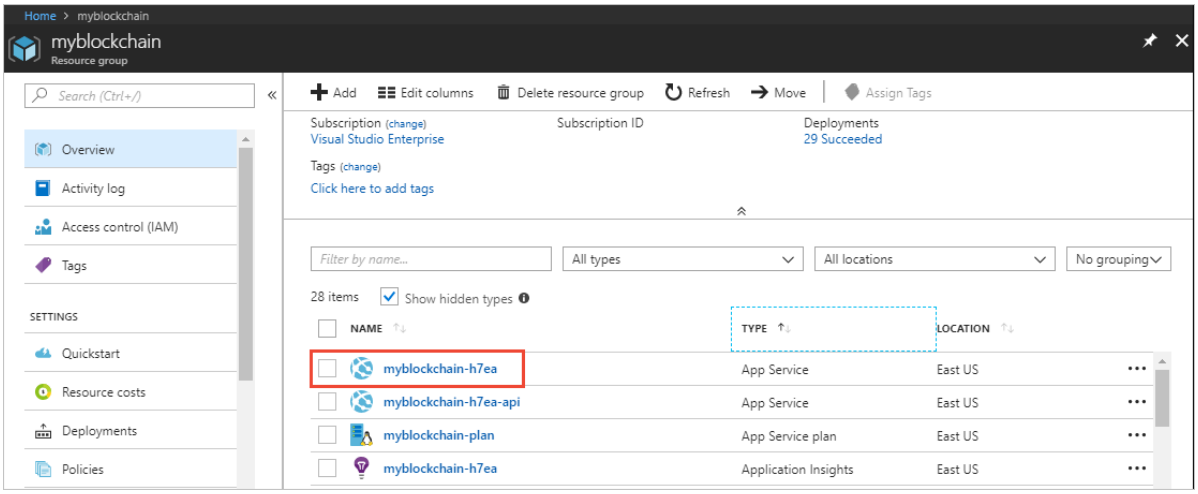
Blockchain Workbench Web URL

Once the deployment of the Blockchain Workbench has completed, a new resource group contains your Blockchain Workbench resources. Blockchain Workbench services are accessed through a web URL. The following steps show you how to retrieve the web URL of the deployed framework.

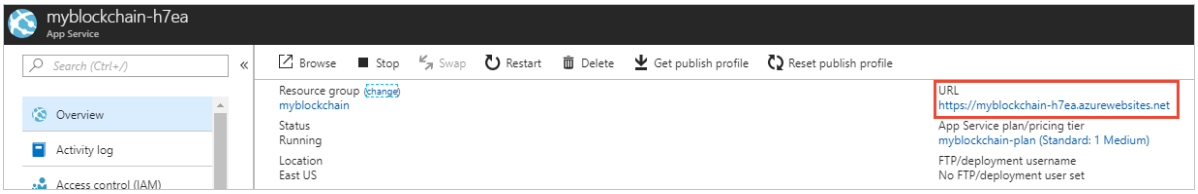
1. Sign in to the Azure portal.

2. In the left-hand navigation pane, select **Resource groups**

3. Choose the resource group name you specified when deploying Blockchain Workbench.
4. Select the **TYPE** column heading to sort the list alphabetically by type.
5. There are two resources with type **App Service**. Select the resource of type **App Service** *without* the "-api" suffix.



6. In the App Service **Essentials** section, copy the **URL** value, which represents the web URL to your deployed Blockchain Workbench.

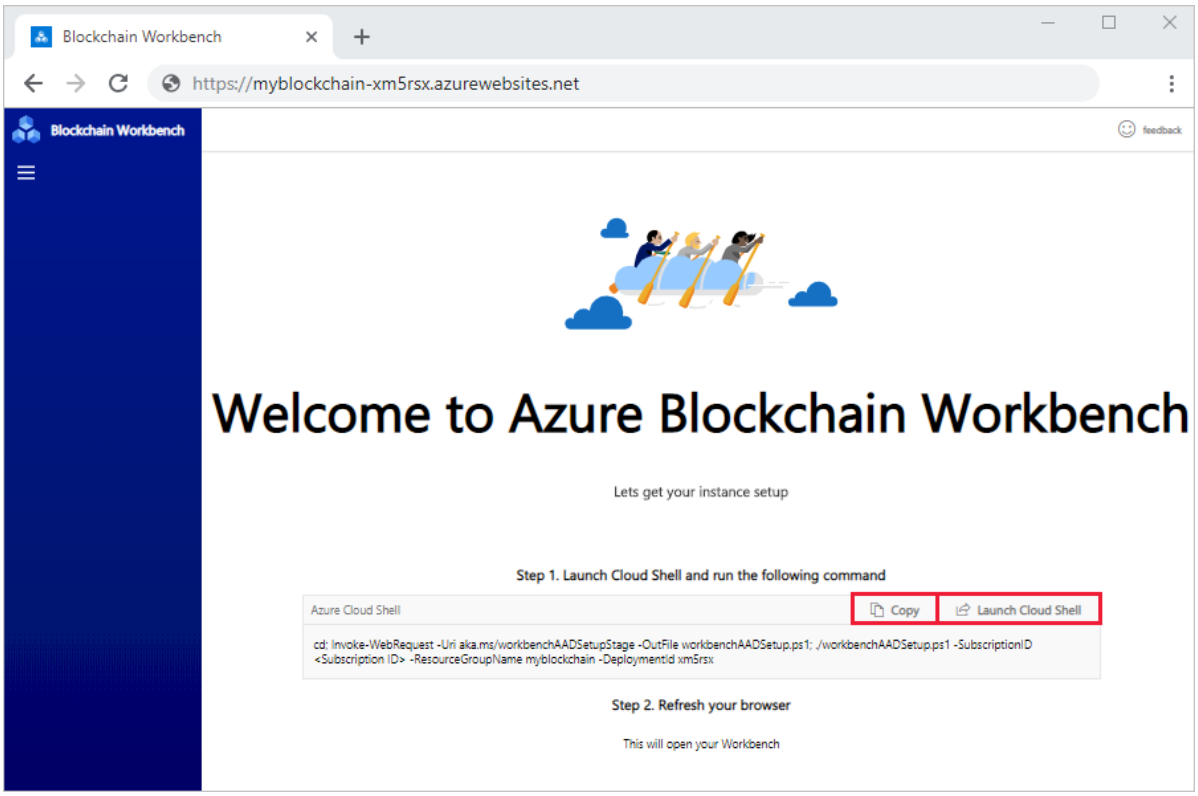


To associate a custom domain name with Blockchain Workbench, see configuring a custom domain name for a web app in Azure App Service using Traffic Manager.

Azure AD configuration script

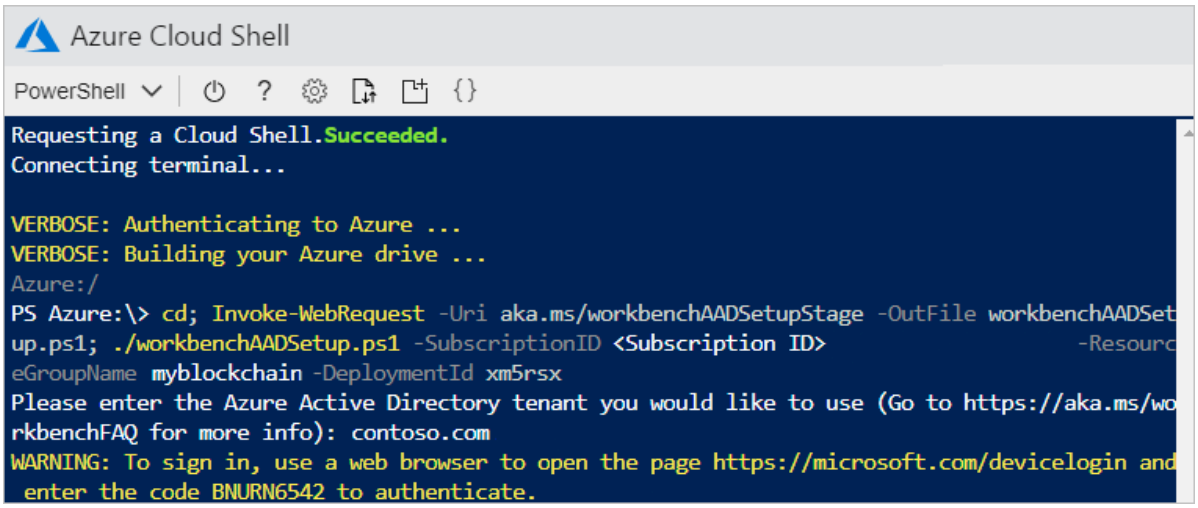
Azure AD must be configured to complete your Blockchain Workbench deployment. You'll use a PowerShell script to do the configuration.

1. In a browser, navigate to the Blockchain Workbench Web URL.
2. You'll see instructions to set up Azure AD using Cloud Shell. Copy the command and launch Cloud Shell.

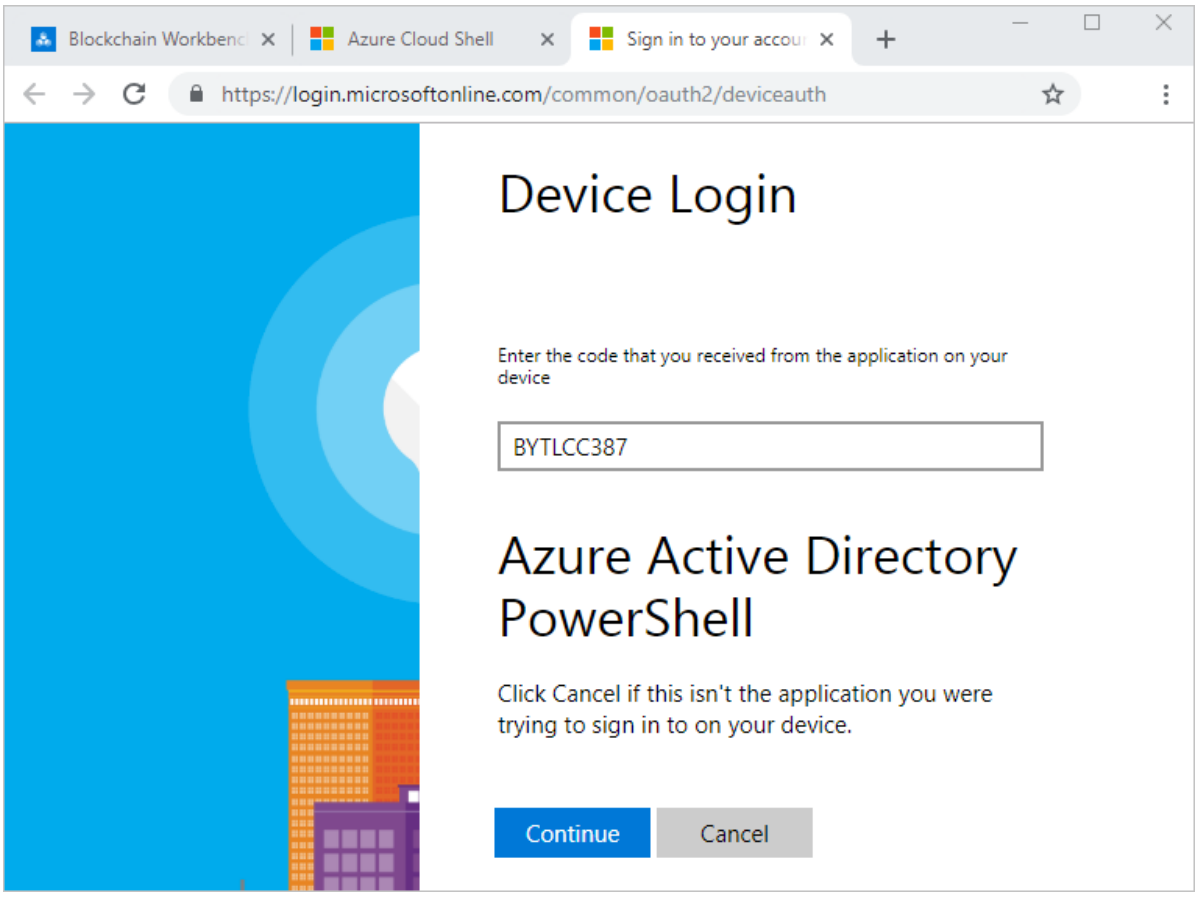


3. Choose the Azure AD tenant where you deployed Blockchain Workbench.
4. In Cloud Shell, paste and run the command.
5. When prompted, enter the Azure AD tenant you want to use for Blockchain Workbench. This will be the tenant containing the users for Blockchain Workbench.

> **IMPORTANT**
> The authenticated user requires permissions to create Azure AD application registrations and grant delegated application permissions in the tenant. You may need to ask an administrator of the tenant to run the Azure AD configuration script or create a new tenant.



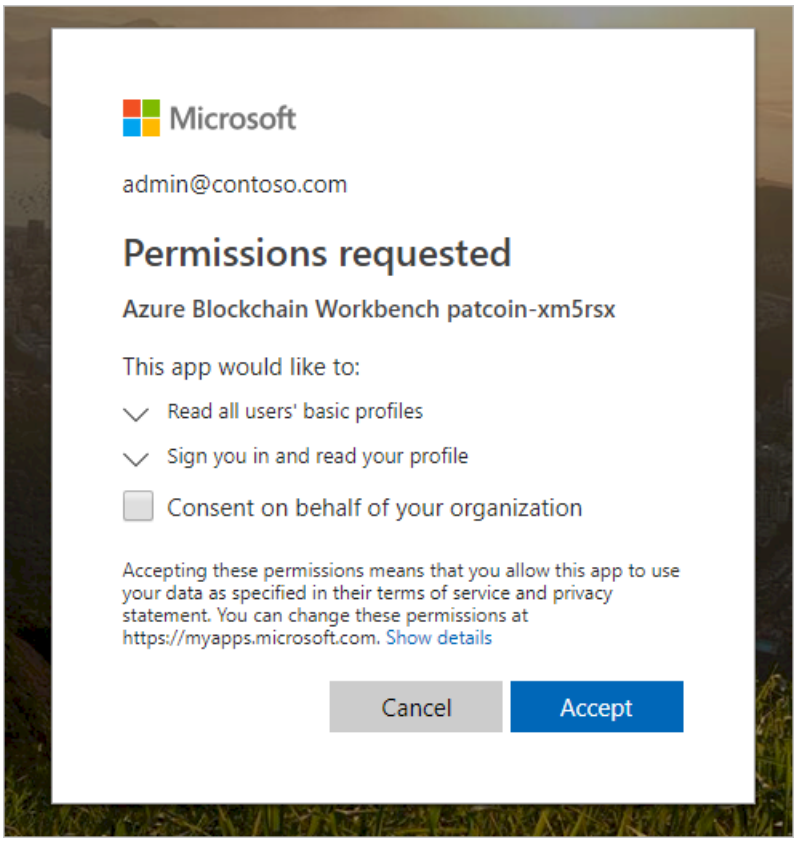
6. You'll be prompted to authenticate to the Azure AD tenant using a browser. Open the web URL in a browser, enter the code, and authenticate.



7. The script outputs several status messages. You get a **SUCCESS** status message if the tenant was successfully provisioned.

8. Navigate to the Blockchain Workbench URL. You are asked to consent to grant read permissions to the directory. This allows the Blockchain Workbench web app access to the users in the tenant. If you are the tenant administrator, you can choose to consent for the entire organization. This option accepts consent for all users in the tenant. Otherwise, each user is prompted for consent on first use of the Blockchain Workbench web application.

9. Select **Accept** to consent.



10. After consent, the Blockchain Workbench web app can be used.

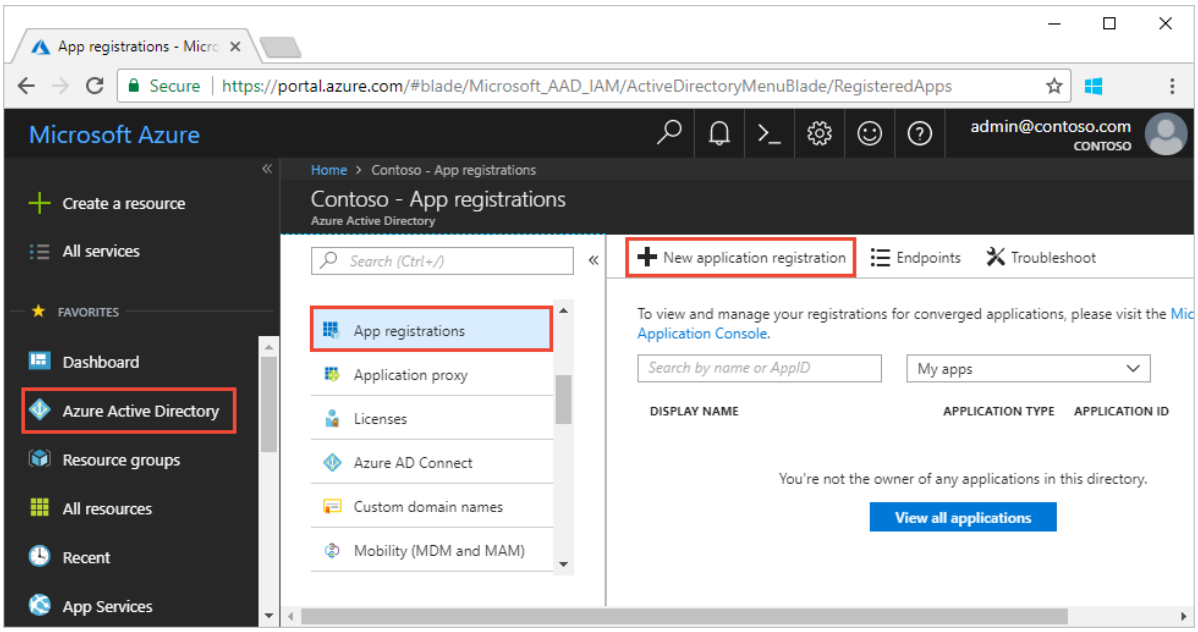
Azure AD configuration

If you choose to manually configure or verify Azure AD settings prior to deployment, complete all steps in this section. If you prefer to automatically configure Azure AD settings, use Azure AD configuration script after you deploy Blockchain Workbench.

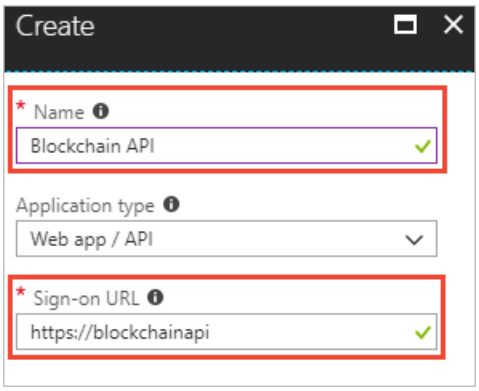
Blockchain Workbench API app registration

Blockchain Workbench deployment requires registration of an Azure AD application. You need an Azure Active Directory (Azure AD) tenant to register the app. You can use an existing tenant or create a new tenant. If you are using an existing Azure AD tenant, you need sufficient permissions to register applications, grant Graph API permissions, and allow guest access within an Azure AD tenant. If you do not have sufficient permissions in an existing Azure AD tenant create a new tenant.

1. Sign in to the Azure portal.
2. Select your account in the top-right corner, and switch to the desired Azure AD tenant. The tenant should be the subscription admin's tenant of the subscription where Workbench is deployed and you have sufficient permissions to register applications.
3. In the left-hand navigation pane, select the **Azure Active Directory** service. Select **App registrations** > **New application registration**.



4. Provide a **Name** and **Sign-on URL** for the application. You can use placeholder values since the values are changed during the deployment.



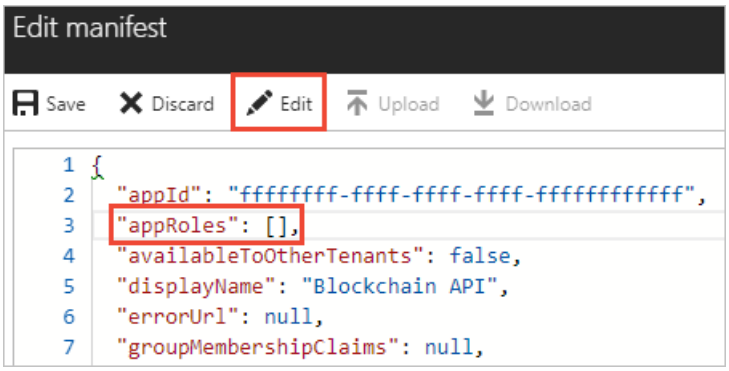
SETTING	VALUE
Name	`Blockchain API`
Application type	Web app / API
Sign-on URL	`https://blockchainapi`

5. Select **Create** to register the Azure AD application.

Modify manifest

Next, you need to modify the manifest to use application roles within Azure AD to specify Blockchain Workbench administrators. For more information about application manifests, see Azure Active Directory application manifest.

1. For the application you registered, select **Manifest** in the registered application details pane.
2. Generate a GUID. You can generate a GUID using the PowerShell command [guid] :: NewGuid () or New-GUID cmdlet. Another option is to use a GUID generator website.
3. You are going to update the **appRoles** section of the manifest. In the Edit manifest pane, select **Edit** and replace `"appRoles": []` with the provided JSON. Be sure to replace the value for the **id** field with the GUID you generated.




```
"appRoles": [
     {
       "allowedMemberTypes": [
         "User",
         "Application"
       ],
       "displayName": "Administrator",
       "id": "<A unique GUID>",
       "isEnabled": true,
       "description": "Blockchain Workbench administrator role allows creation of applications, user to
role assignments, etc.",
       "value": "Administrator"
     }
   ],
```

IMPORTANT

The value **Administrator** is needed to identify Blockchain Workbench administrators.

4. In the manifest, also change the **Oauth2AllowImplicitFlow** value to **true**.

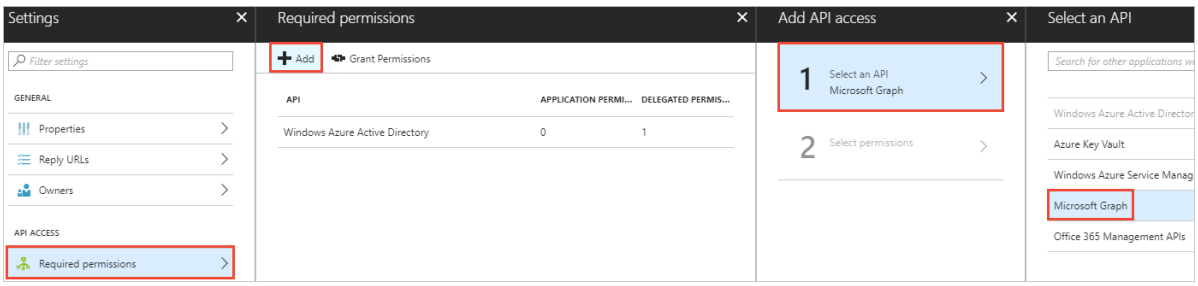
```
"oauth2AllowImplicitFlow": true,
```

5. Select **Save** to save the manifest changes.

Add Graph API required permissions

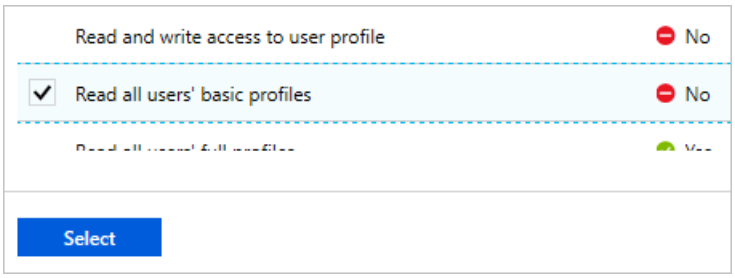
The API application needs to request permission from the user to access the directory. Set the following required permission for the API application:

1. In the Blockchain API app registration, select **Settings** > **Required permissions** > **Select an API** > **Microsoft Graph**.



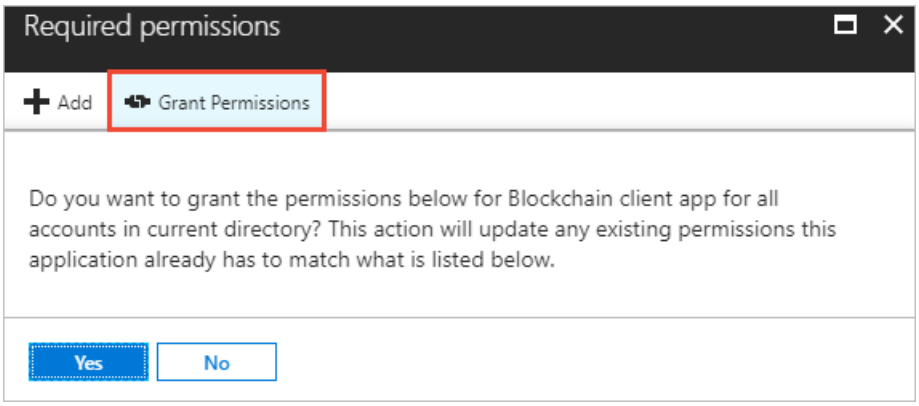
Click **Select**.

2. In **Enable Access** under **Delegated permissions**, choose **Read all users' basic profiles**.



Select **Save** then select **Done**.

3. In **Required permissions**, select **Grant Permissions** then select **Yes** for the verification prompt.

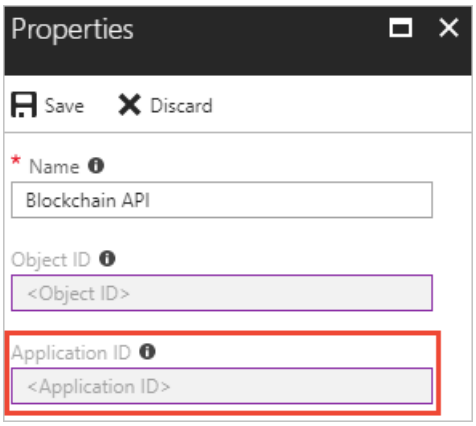


Granting permission allows Blockchain Workbench to access users in the directory. The read permission is required to search and add members to Blockchain Workbench.

Get application ID

The application ID and tenant information are required for deployment. Collect and store the information for use during deployment.

1. For the application you registered, select **Settings** > **Properties**.
2. In the **Properties** pane, copy and store the following values for later use during deployment.

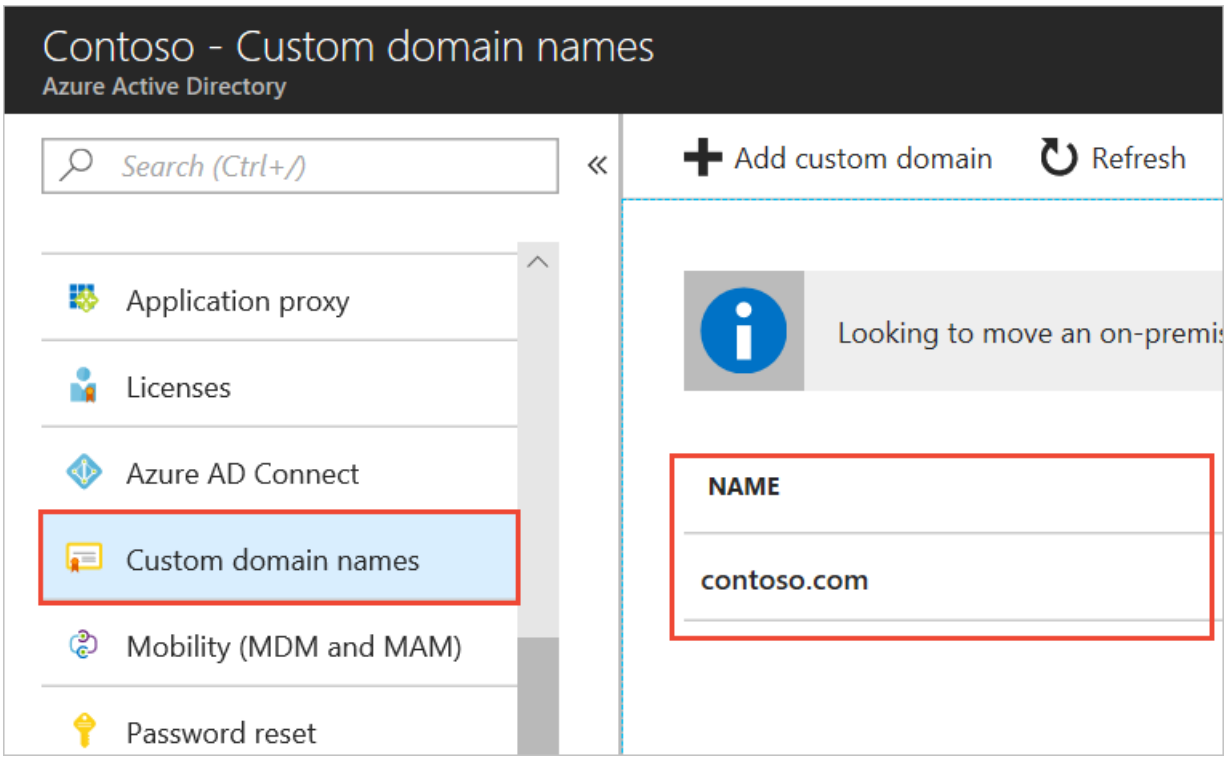


SETTING TO STORE	USE IN DEPLOYMENT
Application ID	Azure Active Directory setup > Application ID

Get tenant domain name

Collect and store the Active Directory tenant domain name where the applications are registered.

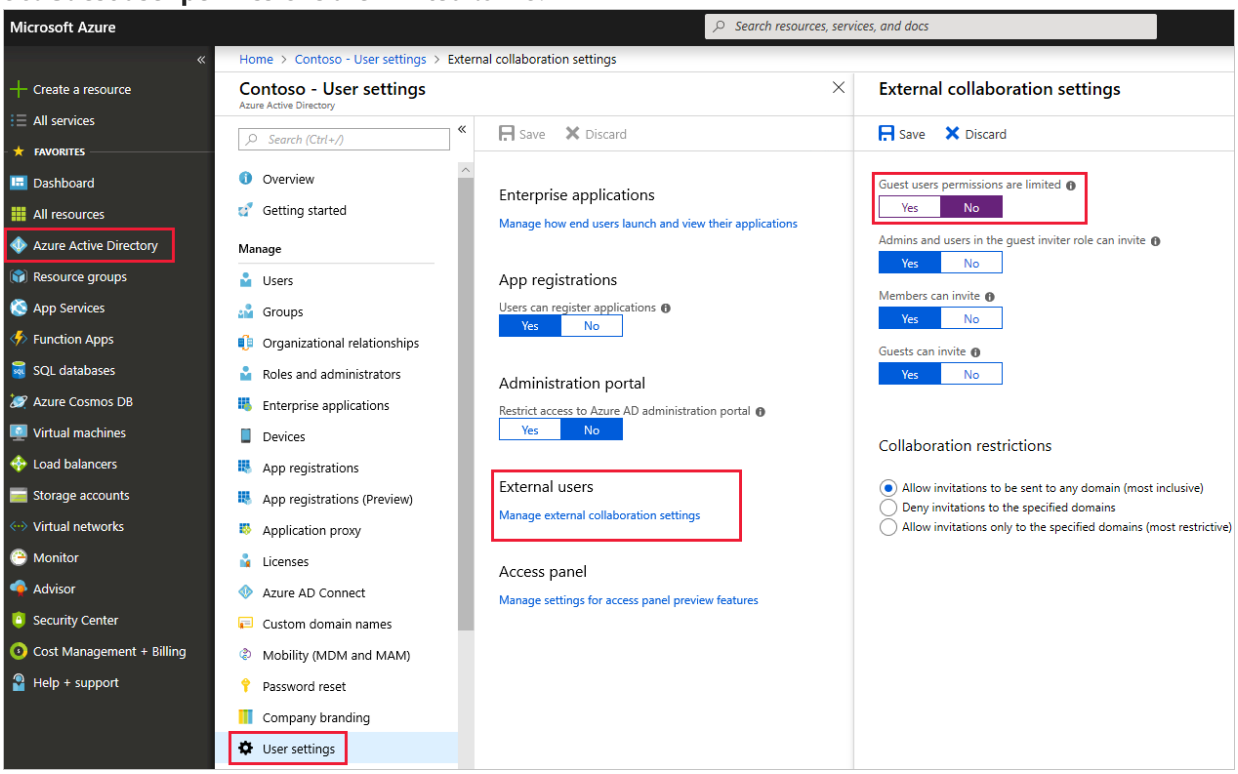
In the left-hand navigation pane, select the **Azure Active Directory** service. Select **Custom domain names**. Copy and store the domain name.



Guest user settings

If you have guest users in your Azure AD tenant, follow the additional steps to ensure Blockchain Workbench user assignment and management works properly.

1. Switch you your Azure AD tenant and select **Azure Active Directory > User settings > Manage external collaboration settings**.
2. Set **Guest user permissions are limited** to **No**.

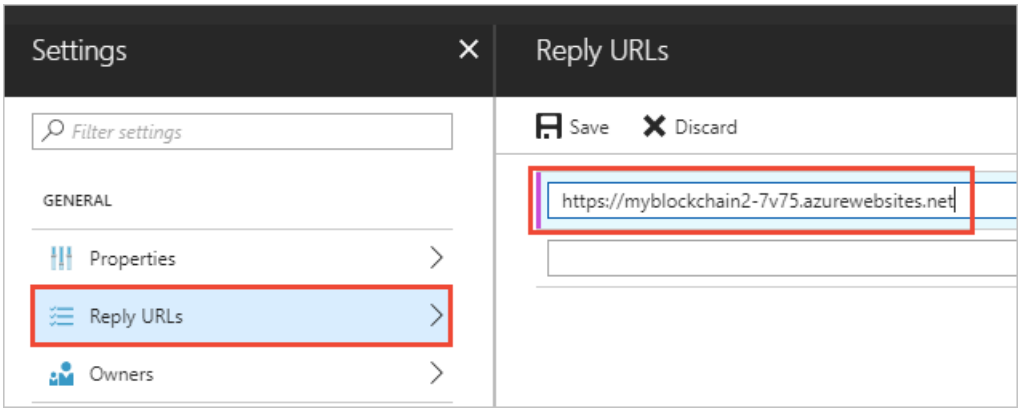


Configuring the Reply URL

Once the Azure Blockchain Workbench has been deployed, you have to configure the Azure Active Directory (Azure AD) client application **Reply URL** of the deployed Blockchain Workbench web URL.

1. Sign in to the Azure portal.

2. Verify you are in the tenant where you registered the Azure AD client application.
3. In the left-hand navigation pane, select the **Azure Active Directory** service. Select **App registrations**.
4. Select the Azure AD client application you registered in the prerequisite section.
5. Select **Settings > Reply URLs**.
6. Specify the main web URL of the Azure Blockchain Workbench deployment you retrieved in the **Get the Azure Blockchain Workbench Web URL** section. The Reply URL is prefixed with `https://`. For example, `https://myblockchain2-7v75.azurewebsites.net`

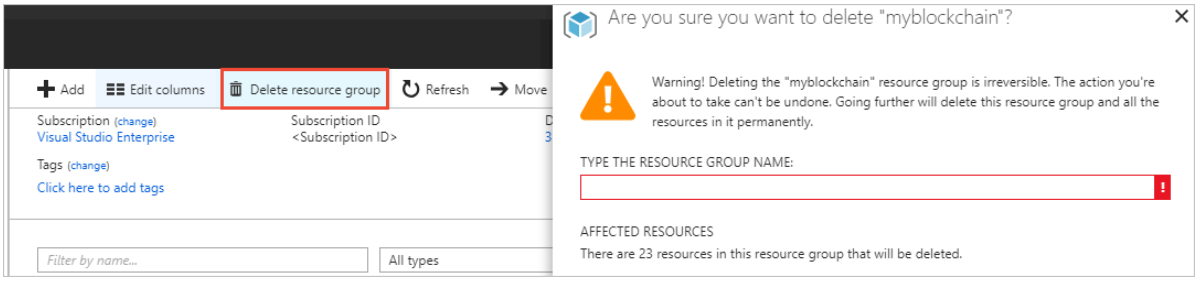


7. Select **Save** to update the client registration.

Remove a deployment

When a deployment is no longer needed, you can remove a deployment by deleting the Blockchain Workbench 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 how-to article, you deployed Azure Blockchain Workbench. To learn how to create a blockchain application, continue to the next how-to article.

Create a blockchain application in Azure Blockchain Workbench

Manage Users in Azure Blockchain Workbench

5/9/2019 • 3 minutes to read • Edit Online

Azure Blockchain Workbench includes user management for people and organizations that are part of your consortium.

Prerequisites

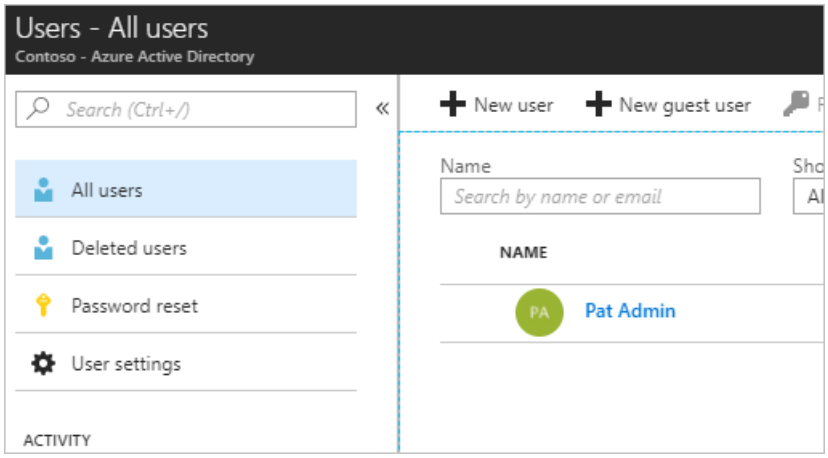
A Blockchain Workbench deployment is required. See Azure Blockchain Workbench deployment for details on deployment.

Add Azure AD users

The Azure Blockchain Workbench uses Azure Active Directory (Azure AD) for authentication, access control, and roles. Users in the Blockchain Workbench Azure AD tenant can authenticate and use Blockchain Workbench. Add users to the Administrator application role to interact and perform actions.

Blockchain Workbench users need to exist in the Azure AD tenant before you can assign them to applications and roles. To add users to Azure AD, use the following steps:

1. Sign in to the Azure portal.
2. Select your account in the top right corner, and switch to the Azure AD tenant associated to Blockchain Workbench.
3. Select **Azure Active Directory > Users**. You see a list of users in your directory.
4. To add users to the directory, select **New user**. For external users, select **New guest user**.



5. Complete the required fields for the new user. Select **Create**.

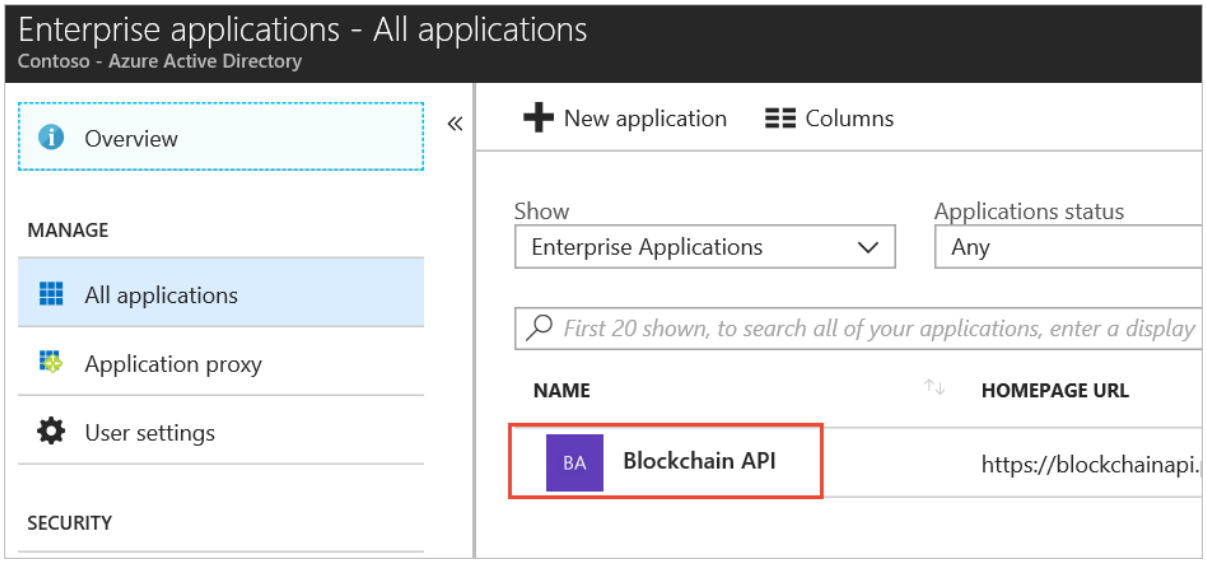
Visit Azure AD documentation for more details on how to manage users within Azure AD.

Manage Blockchain Workbench administrators

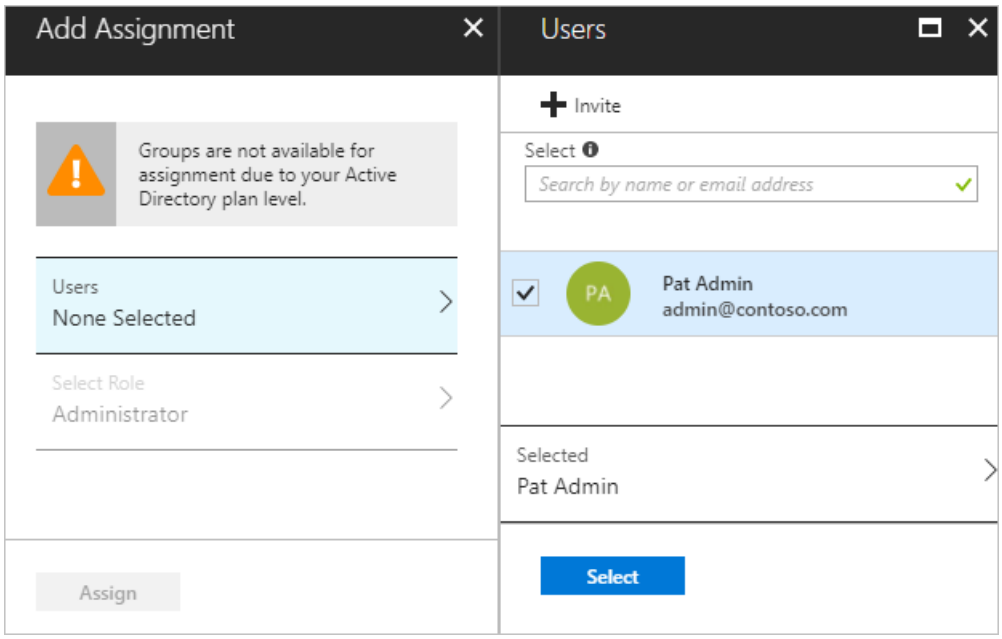
Once users have been added to the directory, the next step is to choose which users are Blockchain Workbench administrators. Users in the **Administrator** group are associated with the **Administrator application role** in Blockchain Workbench. Administrators can add or remove users, assign users to specific scenarios, and create new applications.

To add users to the **Administrator** group in the Azure AD directory:

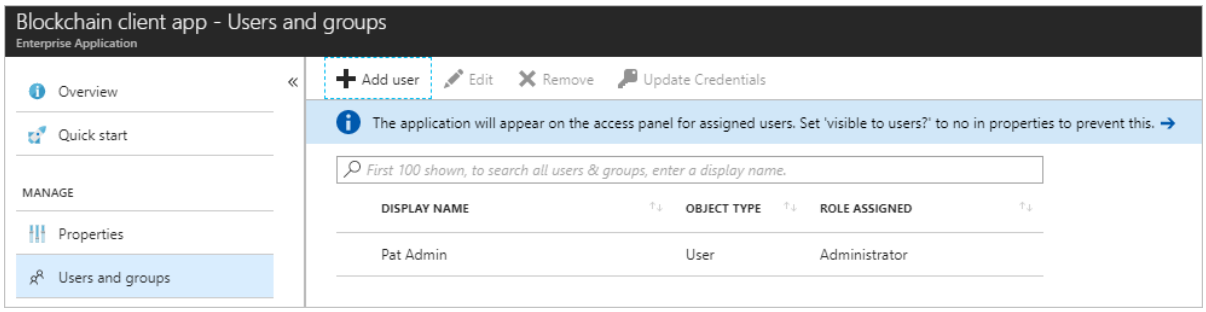
1. Sign in to the Azure portal.
2. Verify you are in the Azure AD tenant associated to Blockchain Workbench by selecting your account in the top-right corner.
3. Select **Azure Active Directory > Enterprise applications**.
4. Select the Azure AD client application for Blockchain Workbench



5. Select **Users and groups > Add user**.
6. In **Add Assignment**, select **Users**. Choose or search for the user you want to add as an administrator. Click **Select** when finished choosing.



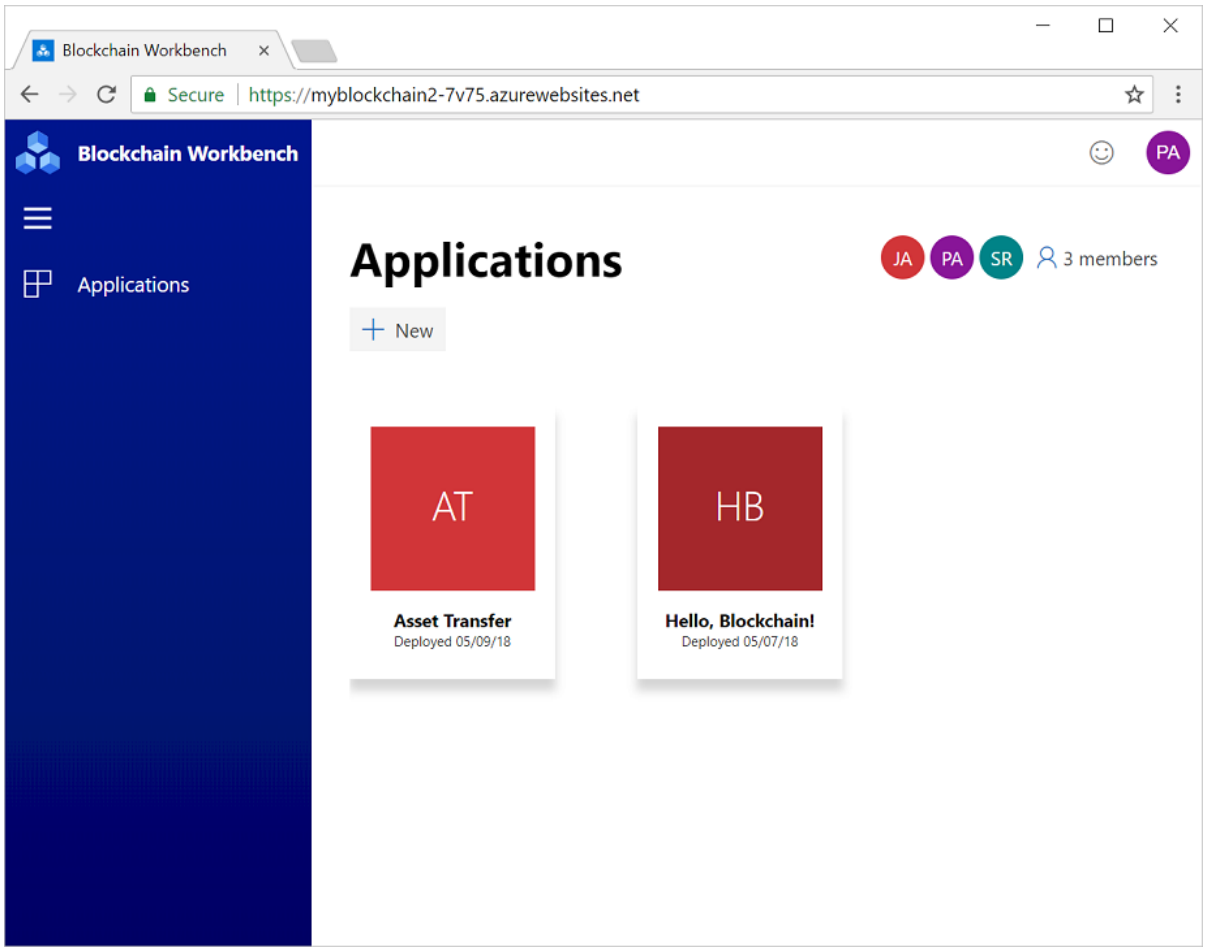
7. Verify **Role** is set to **Administrator**
8. Select **Assign**. The added users are displayed in the list with the administrator role assigned.



Managing Blockchain Workbench members

Use the Blockchain Workbench application to manage users and organizations that are part of your consortium. You can add or remove users to applications and roles.

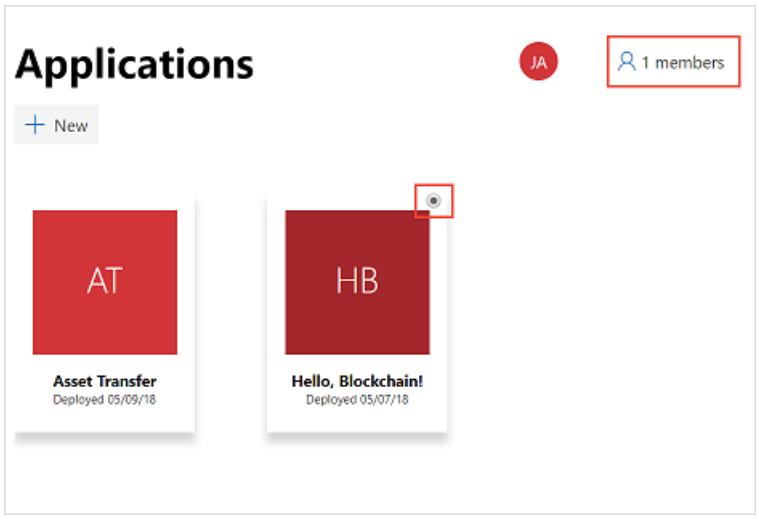
1. Open the Blockchain Workbench in your browser and sign in as an administrator.



   Members are added to each application. Members can have one or more application roles to initiate contracts or take actions.

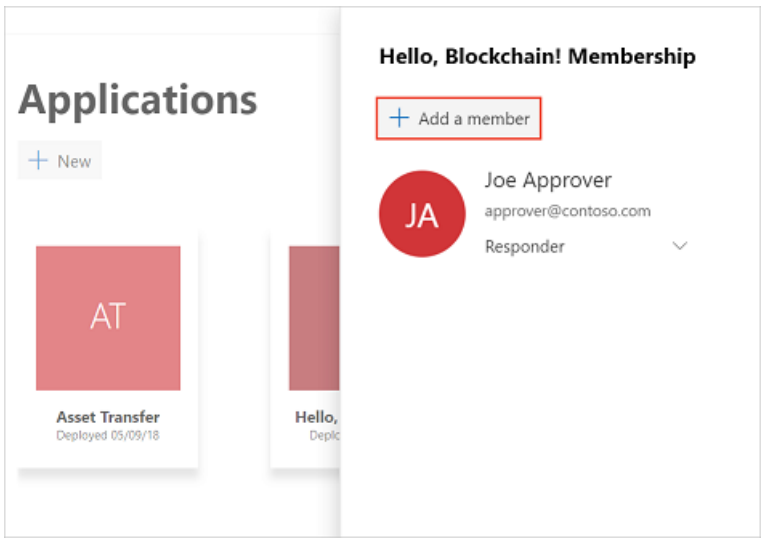
2. To manage members for an application, select an application tile in the **Applications** pane.

   The number of members associated to the selected application is reflected in the members tile.

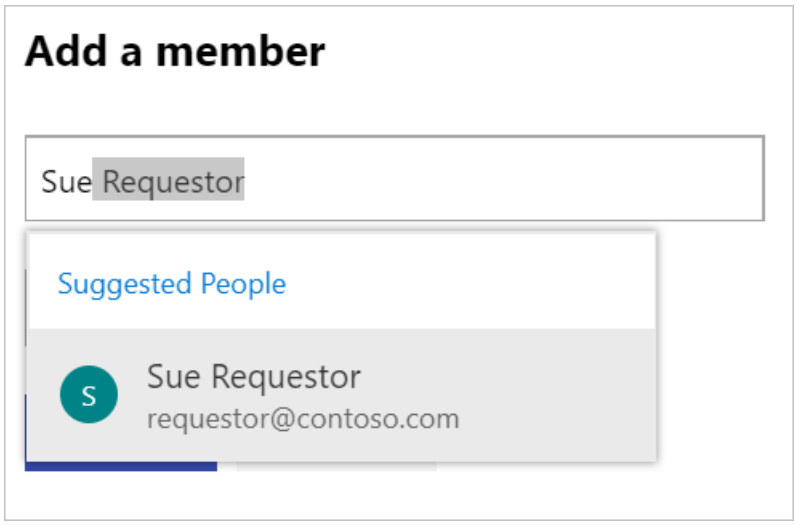


Add member to application

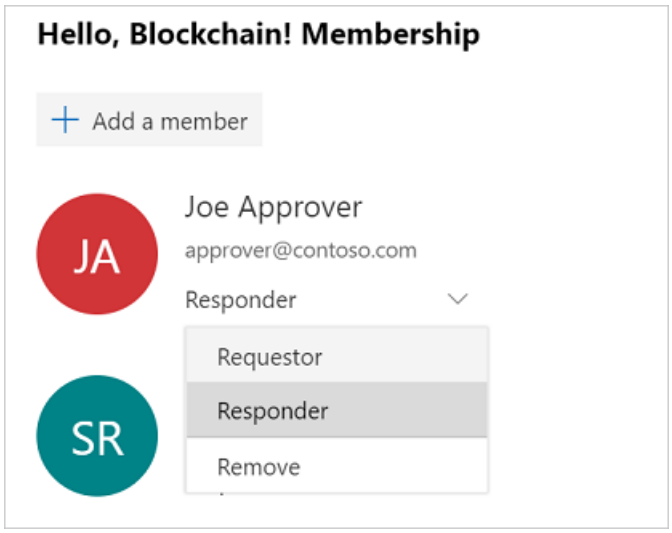
1. Select the member tile to display a list of the current members.
2. Select **Add members**.



3. Search for the user's name. Only Azure AD users that exist in the Blockchain Workbench tenant are listed. If the user is not found, you need to Add Azure AD users.



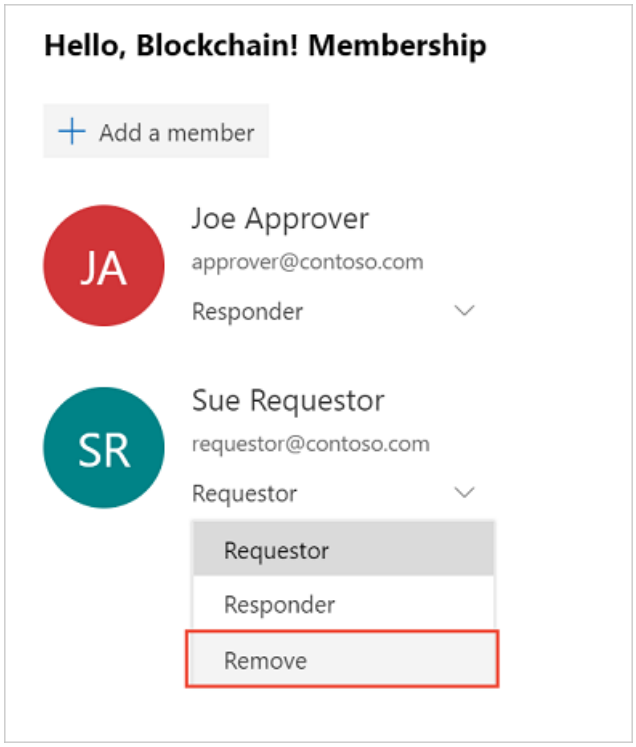
4. Select a **Role** from the drop-down.



5. Select **Add** to add the member with the associated role to the application.

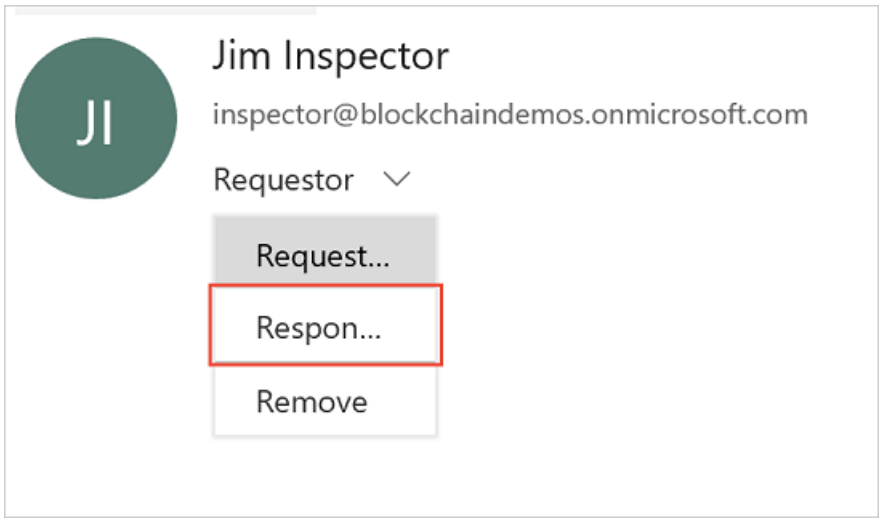
Remove member from application

1. Select the member tile to display a list of the current members.
2. For the user you want to remove, choose **Remove** from the role drop-down.



Change or add role

1. Select the member tile to display a list of the current members.
2. For the user you want to change, click the drop-down and select the new role.



Next steps

In this how-to article, you have learned how to manage users for Azure Blockchain Workbench. To learn how to create a blockchain application, continue to the next how-to article.

Create a blockchain application in Azure Blockchain Workbench

Azure Blockchain Workbench application versioning

4/15/2019 • 2 minutes to read • Edit Online

You can create and use multiple versions of an Azure Blockchain Workbench app. If multiple versions of the same application are uploaded, a version history is available and users can choose which version they want to use.

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

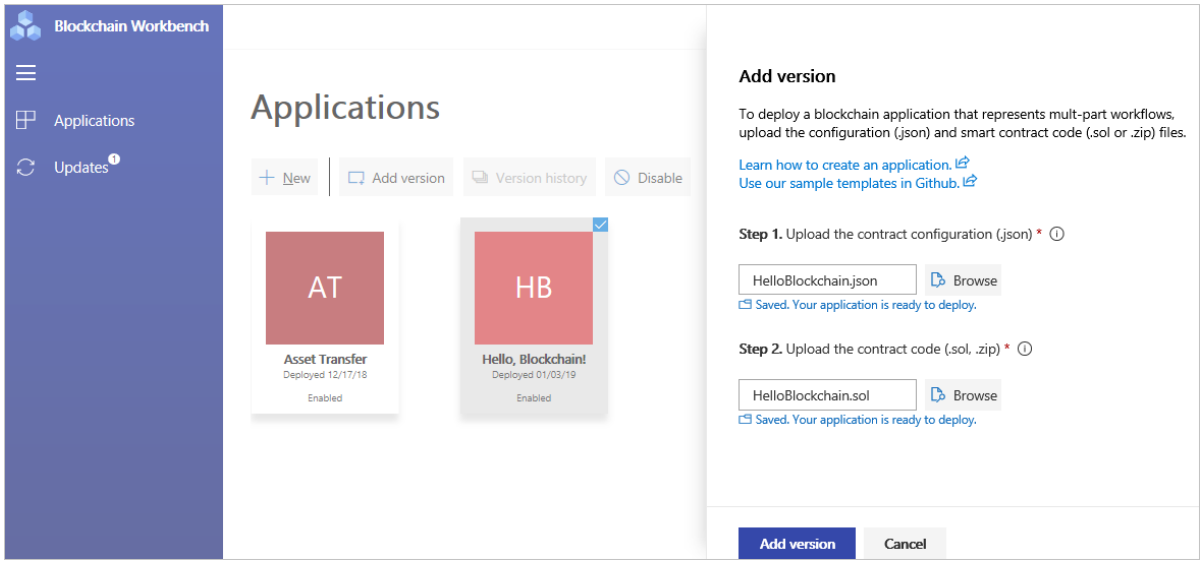
Prerequisites

- A Blockchain Workbench deployment. For more information, see Azure Blockchain Workbench deployment for details on deployment
- A deployed blockchain application in Blockchain Workbench. See Create a blockchain application in Azure Blockchain Workbench

Add an app version

To add a new version, upload the new configuration and smart contract files to Blockchain Workbench.

1. In a web browser, navigate to the Blockchain Workbench web address. For example, `https://{workbench URL}.azurewebsites.net/` For information on how to find your Blockchain Workbench web address, see Blockchain Workbench Web URL
2. Sign in as a Blockchain Workbench administrator.
3. Select the blockchain application you want to update with another version.
4. Select **Add version**. The **Add version** pane is displayed.
5. Choose the new version contract configuration and contract code files to upload. The configuration file is automatically validated. Fix any validation errors before you deploy the application.
6. Select **Add version** to add the new blockchain application version.



Deployment of the blockchain application can take a few minutes. When deployment is finished, refresh the application page. Choosing the application and selecting the **Version history** button, displays the version history of the application.

IMPORTANT

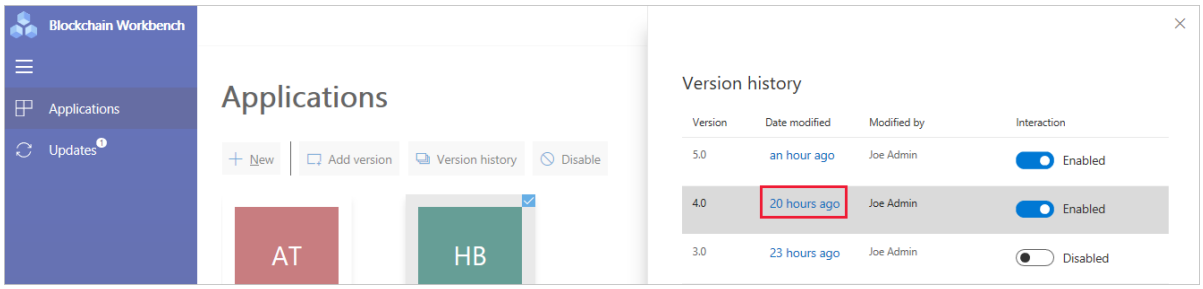
Previous versions of the application are disabled. You can individually re-enable past versions.

You may need to re-add members to application roles if changes were made to the application roles in the new version.

Using app versions

By default, the latest enabled version of the application is used in Blockchain Workbench. If you want to use a previous version of an application, you need to choose the version from the application page first.

1. In Blockchain Workbench application section, select the application checkbox that contains the contract you want to use. If previous versions are enabled, the version history button is available.
2. Select the **Version history** button.
3. In the version history pane, choose the version of the application by selecting the link in the *Date modified* column.



You can create new contracts or take actions on previous version contracts. The version of the application is displayed following the application name and a warning is displayed about the older version.

Next steps

- Azure Blockchain Workbench troubleshooting

Using the Azure Blockchain Workbench REST API

4/15/2019 • 5 minutes to read • Edit Online

Azure Blockchain Workbench REST API provides developers and information workers a way to build rich integrations to blockchain applications. This document walks you through several key methods of the Workbench REST API. For example, suppose a developer wants to create a custom blockchain client. This blockchain client allows signed in users to view and interact with their assigned blockchain applications. The client allows users to view contract instances and take actions on smart contracts. The client uses the Workbench REST API in the context of the signed-in user to do the following actions:

- List applications
- List workflows for an application
- List smart contract instances for a workflow
- List available actions for a contract
- Execute an action for a contract

The example blockchain applications used in the scenarios, can be downloaded from GitHub.

List applications

Once a user has signed into the blockchain client, the first task is to retrieve all Blockchain Workbench applications for the user. In this scenario, the user has access to two applications:

1. Asset transfer
2. Refrigerated transportation

Use the Applications GET API:

```
GET /api/v1/applications
Authorization : Bearer {access token}
```

The response lists all blockchain applications to which a user has access in Blockchain Workbench. Blockchain Workbench administrators get every blockchain application. Non-Workbench administrators get all blockchains for which they have at least one associated application role or an associated smart contract instance role.

```
HTTP/1.1 200 OK
Content-type: application/json
{
    "nextLink": "/api/v1/applications?skip=2",
    "applications": [
        {
            "id": 1,
            "name": "AssetTransfer",
            "description": "Allows transfer of assets between a buyer and a seller, with appraisal/inspection
functionality",
            "displayName": "Asset Transfer",
            "createdByUserId": 1,
            "createdDtTm": "2018-04-28T05:59:14.4733333",
            "enabled": true,
            "applicationRoles": null
        },
        {
            "id": 2,
            "name": "RefrigeratedTransportation",
            "description": "Application to track end-to-end transportation of perishable goods.",
            "displayName": "Refrigerated Transportation",
            "createdByUserId": 7,
            "createdDtTm": "2018-04-28T18:25:38.71",
            "enabled": true,
            "applicationRoles": null
        }
    ]
}
```

List workflows for an application

Once a user selects the applicable blockchain application (such as **Asset Transfer**), the blockchain client retrieves all workflows of the specific blockchain application. Users can then select the applicable workflow before being shown all smart contract instances for the workflow. Each blockchain application has one or more workflows and each workflow has zero or smart contract instances. For a blockchain client application that has only one workflow, we recommend skipping the user experience flow that allows users to select the appropriate workflow. In this case, **Asset Transfer** has only one workflow, also called **Asset Transfer**.

Use the Applications Workflows GET API:

```
GET /api/v1/applications/{applicationId}/workflows
Authorization: Bearer {access token}
```

Response lists all workflows of the specified blockchain application to which a user has access in Blockchain Workbench. Blockchain Workbench administrators get every blockchain workflow. Non-Workbench administrators get all workflows for which they have at least one associated application role or is associated with a smart contract instance role.

```
HTTP/1.1 200 OK
Content-type: application/json
{
    "nextLink": "/api/v1/applications/1/workflows?skip=1",
    "workflows": [
        {
            "id": 1,
            "name": "AssetTransfer",
            "description": "Handles the business logic for the asset transfer scenario",
            "displayName": "Asset Transfer",
            "applicationId": 1,
            "constructorId": 1,
            "startStateId": 1
        }
    ]
}
```

List smart contract instances for a workflow

Once a user selects the applicable workflow, this case **Asset Transfer**, the blockchain client will retrieve all smart contract instances for the specified workflow. You can use this information to show all smart contract instances for the workflow. Or you can allow users to deep dive into any of the shown smart contract instances. In this example, consider a user would like to interact with one of the smart contract instances to take action.

Use the Contracts GET API:

```
GET api/v1/contracts?workflowId={workflowId}
Authorization: Bearer {access token}
```

Response lists all smart contract instances of the specified workflow. Workbench administrators get all smart contract instances. Non-Workbench administrators get every smart contract instance for which they have at least one associated application role or is associated with a smart contract instance role.

```
HTTP/1.1 200 OK
Content-type: application/json
{
    "nextLink": "/api/v1/contracts?skip=3&workflowId=1",
    "contracts": [
        {
            "id": 1,
            "provisioningStatus": 2,
            "connectionID": 1,
            "ledgerIdentifier": "0xbcb6127be062acd37818af290c0e43479a153a1c",
            "deployedByUserId": 1,
            "workflowId": 1,
            "contractCodeId": 1,
            "contractProperties": [
                {
                    "workflowPropertyId": 1,
                    "value": "0"
                },
                {
                    "workflowPropertyId": 2,
                    "value": "My first car"
                },
                {
                    "workflowPropertyId": 3,
                    "value": "54321"
                },
                {
                    "workflowPropertyId": 4,
                    "value": "0"
```

```
                    "value": "0"
                },
                {
                    "workflowPropertyId": 5,
                    "value": "0x0000000000000000000000000000000000000000"
                },
                {
                    "workflowPropertyId": 6,
                    "value": "0x0000000000000000000000000000000000000000"
                },
                {
                    "workflowPropertyId": 7,
                    "value": "0x0000000000000000000000000000000000000000"
                },
                {
                    "workflowPropertyId": 8,
                    "value": "0xd882530eb3d6395e697508287900c7679dbe02d7"
                }
            ],
            "transactions": [
                {
                    "id": 1,
                    "connectionId": 1,
                    "transactionHash": "0xf3abb829884dc396e03ae9e115a770b230fcf41bb03d39457201449e077080f4",
                    "blockID": 241,
                    "from": "0xd882530eb3d6395e697508287900c7679dbe02d7",
                    "to": null,
                    "value": 0,
                    "isAppBuilderTx": true
                }
            ],
            "contractActions": [
                {
                    "id": 1,
                    "userId": 1,
                    "provisioningStatus": 2,
                    "timestamp": "2018-04-29T23:41:14.9333333",
                    "parameters": [
                        {
                            "name": "Description",
                            "value": "My first car"
                        },
                        {
                            "name": "Price",
                            "value": "54321"
                        }
                    ],
                    "workflowFunctionId": 1,
                    "transactionId": 1,
                    "workflowStateId": 1
                }
            ]
        }
    ]
}
```


List available actions for a contract

Once a user decides to deep dive into a contract, the blockchain client can then show the available user actions given the state of the contract. In this example, the user is looking at all available actions for a new smart contract they created:

- Modify: Allows the user to modify the description and price of an asset.
- Terminate: Allows the user to end the contract of the asset.

Use the Contract Action GET API:

```
GET /api/v1/contracts/{contractId}/actions
Authorization: Bearer {access token}
```

Response lists all actions to which a user can take given the current state of the specified smart contract instance. Users get all applicable actions if the user has an associated application role or is associated with a smart contract instance role for the current state of the specified smart contract instance.


```
HTTP/1.1 200 OK
Content-type: application/json
{
    "nextLink": "/api/v1/contracts/1/actions?skip=2",
    "workflowFunctions": [
        {
            "id": 2,
            "name": "Modify",
            "description": "Modify the description/price attributes of this asset transfer instance",
            "displayName": "Modify",
            "parameters": [
                {
                    "id": 1,
                    "name": "description",
                    "description": "The new description of the asset",
                    "displayName": "Description",
                    "type": {
                        "id": 2,
                        "name": "string",
                        "elementType": null,
                        "elementTypeId": 0
                    }
                },
                {
                    "id": 2,
                    "name": "price",
                    "description": "The new price of the asset",
                    "displayName": "Price",
                    "type": {
                        "id": 3,
                        "name": "money",
                        "elementType": null,
                        "elementTypeId": 0
                    }
                }
            ],
            "workflowId": 1
        },
        {
            "id": 3,
            "name": "Terminate",
            "description": "Used to cancel this particular instance of asset transfer",
            "displayName": "Terminate",
            "parameters": [],
            "workflowId": 1
        }
    ]
}
```


Execute an action for a contract

A user can then decide to take action for the specified smart contract instance. In this case, consider the scenario where a user would like to modify the description and price of an asset to the following action:

- Description: "My updated car"

- Price: 54321

Use the Contract Action POST API:

```
POST /api/v1/contracts/{contractId}/actions
Authorization: Bearer {access token}
actionInformation: {
    "workflowFunctionId": 2,
    "workflowActionParameters": [
        {
            "name": "description",
            "value": "My updated car"
        },
        {
            "name": "price",
            "value": "54321"
        }
    ]
}
```

Users are only able to execute the action given the current state of the specified smart contract instance and the user's associated application role or smart contract instance role. If the post is successful, an HTTP 200 OK response is returned with no response body.

```
HTTP/1.1 200 OK
Content-type: application/json
```

Next steps

Azure Blockchain Workbench REST API reference

Azure Blockchain Workbench troubleshooting

5/9/2019 • 2 minutes to read • Edit Online

A PowerShell script is available to assist with developer debugging or support. The script generates a summary and collects detailed logs for troubleshooting. Collected logs include:

- Blockchain network, such as Ethereum
- Blockchain Workbench microservices
- Application Insights
- Azure Monitoring (Azure Monitor logs)

You can use the information to determine next steps and determine root cause of issues.

Troubleshooting script

The PowerShell troubleshooting script is available on GitHub. Download a zip file or clone the sample from GitHub.

```
git clone https://github.com/Azure-Samples/blockchain.git
```

Run the script

If needed, install the Azure PowerShell module using the instructions found in the Azure PowerShell guide, and then run `Connect-AzAccount` to create a connection with Azure. Also, you need to have an SSH public key named `id_rsa.pub` in the .ssh directory of your user profile.

Run the `collectBlockchainWorkbenchTroubleshooting.ps1` script to collect logs and create a ZIP file containing a folder of troubleshooting information. For example:

```
collectBlockchainWorkbenchTroubleshooting.ps1 -SubscriptionID "<subscription_id>" -ResourceGroupName
"workbench-resource-group-name"
```

The script accepts the following parameters:

PARAMETER	DESCRIPTION	REQUIRED
SubscriptionID	SubscriptionID to create or locate all resources.	Yes
ResourceGroupName	Name of the Azure Resource Group where Blockchain Workbench has been deployed.	Yes
OutputDirectory	Path to create the output .ZIP file. If not specified, defaults to the current directory.	No
LookbackHours	Number of hours to use when pulling telemetry. Default value is 24 hours. Maximum value is 90 hours	No

PARAMETER	DESCRIPTION	REQUIRED
OmsSubscriptionId	The subscription ID where Azure Monitor logs is deployed. Only pass this parameter if the Azure Monitor logs for the blockchain network is deployed outside of Blockchain Workbench's resource group.	No
OmsResourceGroup	The resource group where Azure Monitor logs is deployed. Only pass this parameter if the Azure Monitor logs for the blockchain network is deployed outside of Blockchain Workbench's resource group.	No
OmsWorkspaceName	The Log Analytics workspace name. Only pass this parameter if the Azure Monitor logs for the blockchain network is deployed outside of Blockchain Workbench's resource group	No

What is collected?

The output ZIP file contains the following folder structure:

FOLDER OR FILE	DESCRIPTION
\Summary.txt	Summary of the system
\Metrics\blockchain	Metrics about the blockchain
\Metrics\Workbench	Metrics about the workbench
\Details\Blockchain	Detailed logs about the blockchain
\Details\Workbench	Detailed logs about the workbench

The summary file gives you a snapshot of the overall state of the application and health of the application. The summary provides recommended actions, highlights top errors, and metadata about running services.

The **Metrics** folder contains metrics of various system components over time. For example, the output file `\Details\Workbench\apiMetrics.txt` contains a summary of different response codes, and response times throughout the collection period. The **Details** folder contains detailed logs for troubleshooting specific issues with Workbench or the underlying blockchain network. For example, `\Details\Workbench\Exceptions.csv` contains a list of the most recent exceptions that have occurred in the system, which is useful for troubleshooting errors with smart contracts or interactions with the blockchain.

Next steps

Azure Blockchain Workbench Application Insights troubleshooting guide

Configure the Azure Blockchain Workbench database firewall

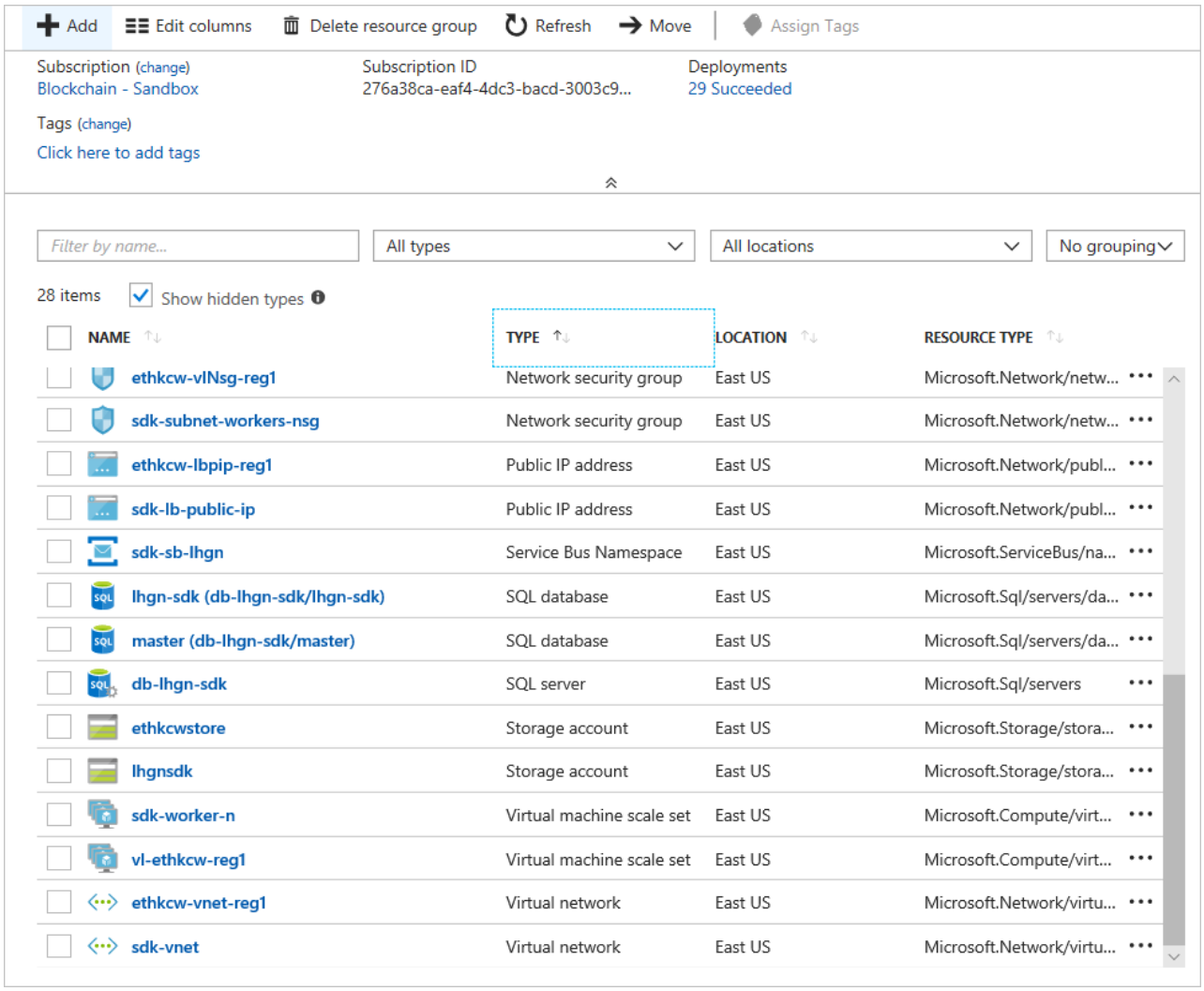
5/9/2019 • 2 minutes to read • Edit Online

This article shows how to configure a firewall rule using the Azure portal. Firewall rules let external clients or applications connect to your Azure Blockchain Workbench database.

Connect to the Blockchain Workbench database

To connect to the database where you want to configure a rule:

1. Sign in to the Azure Portal with an account that has **Owner** permissions for the Azure Blockchain Workbench resources.
2. In the left navigation pane, choose **Resource groups**.
3. Choose the name of the resource group for your Blockchain Workbench deployment.
4. Select **Type** to sort the list of resources, and then choose your **SQL server**.
5. The resource list example in the following screen capture shows two databases: *master* and *lsgn-sdk*. You configure the firewall rule on *lsgn-sdk*.

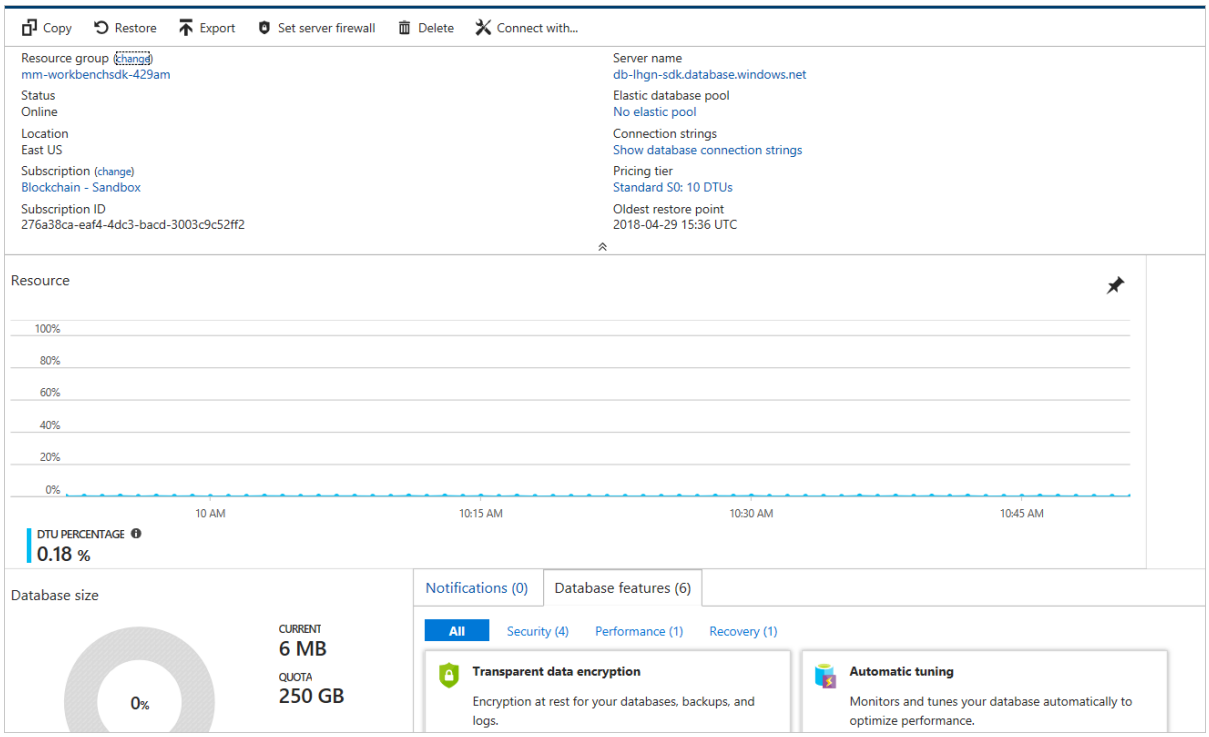


Create a database firewall rule

To create a firewall rule:

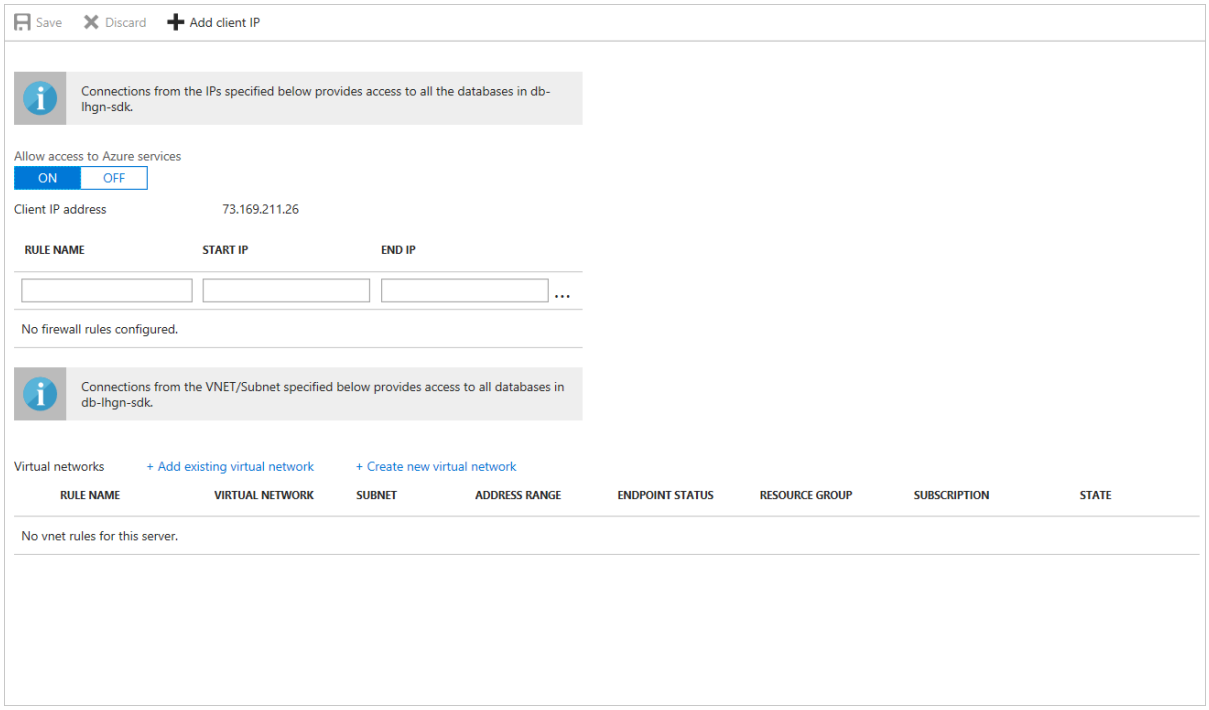
1. Choose the link to the "lsgn-sdk" database.

2. On the menu bar, select **Set server firewall**.



3. To create a rule for your organization:

   - Enter a **RULE NAME**
   - Enter an IP address for the **START IP** of the address range
   - Enter an IP address for the **END IP** of the address range



> **NOTE**
>
> If you only want to add the IP address of your computer, choose **+ Add client IP**.

4. To save your firewall configuration, select **Save**.
5. Test the IP address range you configured for the database by connecting from an application or tool. For example, SQL Server Management Studio.

Next steps

Database views in Azure Blockchain Workbench

Get information about your Azure Blockchain Workbench database

5/9/2019 • 2 minutes to read • Edit Online

This article shows how to get detailed information about your Azure Blockchain Workbench database.

Overview

Information about applications, workflows, and smart contract execution is provided using database views in the Blockchain Workbench SQL DB. Developers can use this information when using tools such as Microsoft Excel, Power BI, Visual Studio, and SQL Server Management Studio.

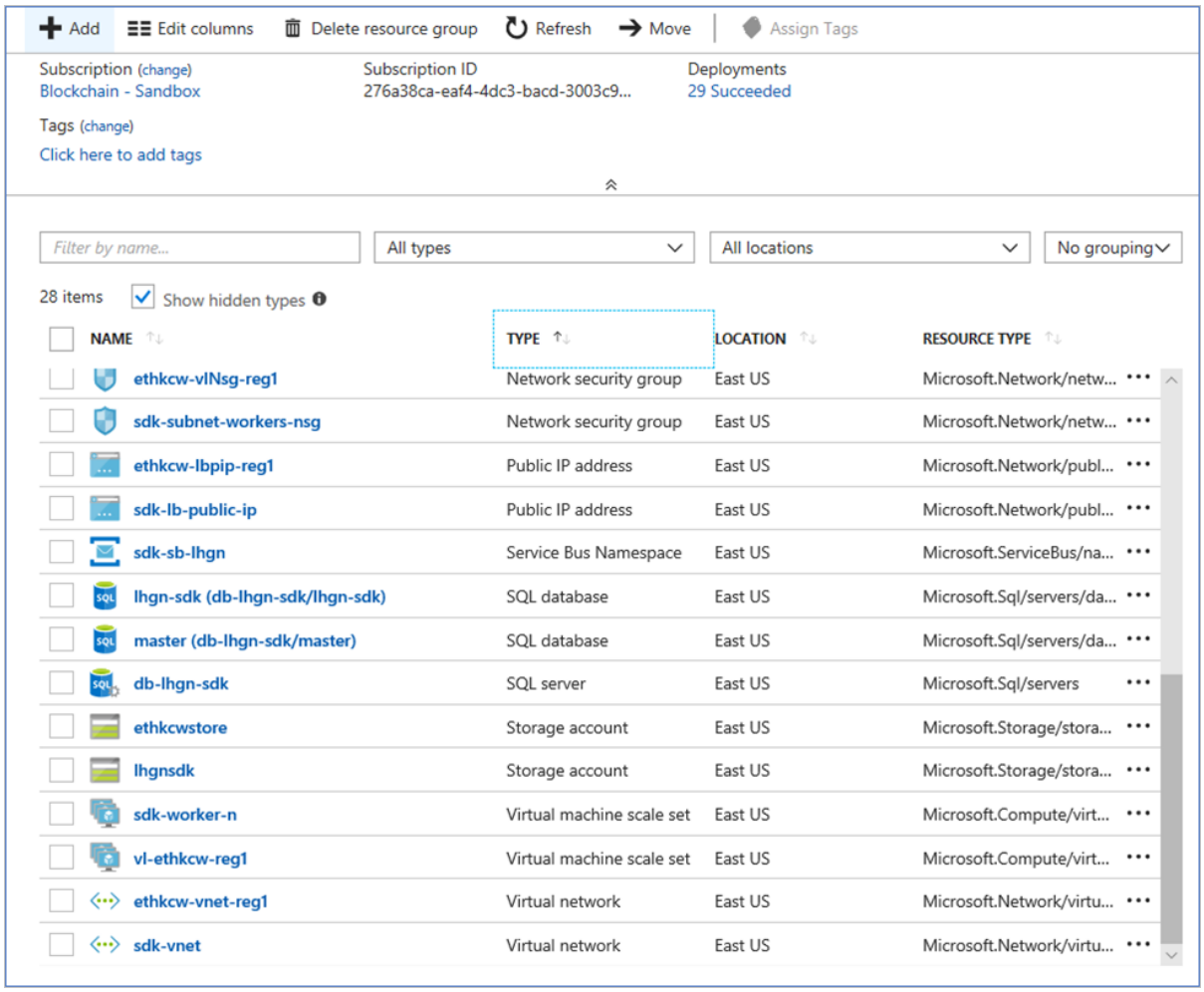
Before a developer can connect to the database, they need:

- External client access allowed in the database firewall. This article about configuring a database firewall article explains how to allow access.
- The database server name and database name.

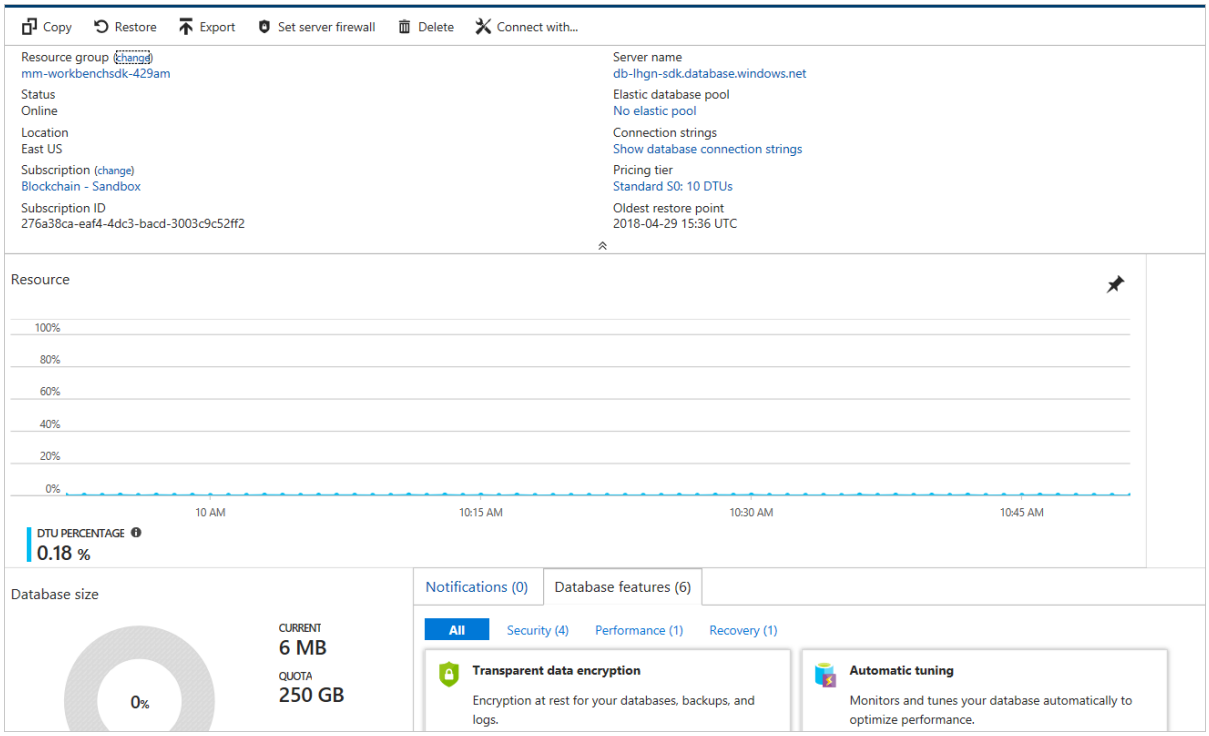
Connect to the Blockchain Workbench database

To connect to the database:

1. Sign in to the Azure portal with an account that has **Owner** permissions for the Azure Blockchain Workbench resources.
2. In the left navigation pane, choose **Resource groups**.
3. Choose the name of the resource group for your Blockchain Workbench deployment.
4. Select **Type** to sort the resource list, and then choose your **SQL server**. The sorted list in the next screen capture shows two SQL databases, "master" and one that uses "lhgn" as the **Resource prefix**.



5. To see detailed information about the Blockchain Workbench database, select the link for the database with the **Resource prefix** you provided for deploying Blockchain Workbench.



The database server name and database name let you connect to the Blockchain Workbench database using your development or reporting tool.

Next steps

Database views in Azure Blockchain Workbench

View Azure Blockchain Workbench data with Microsoft Excel

5/9/2019 • 2 minutes to read • Edit Online

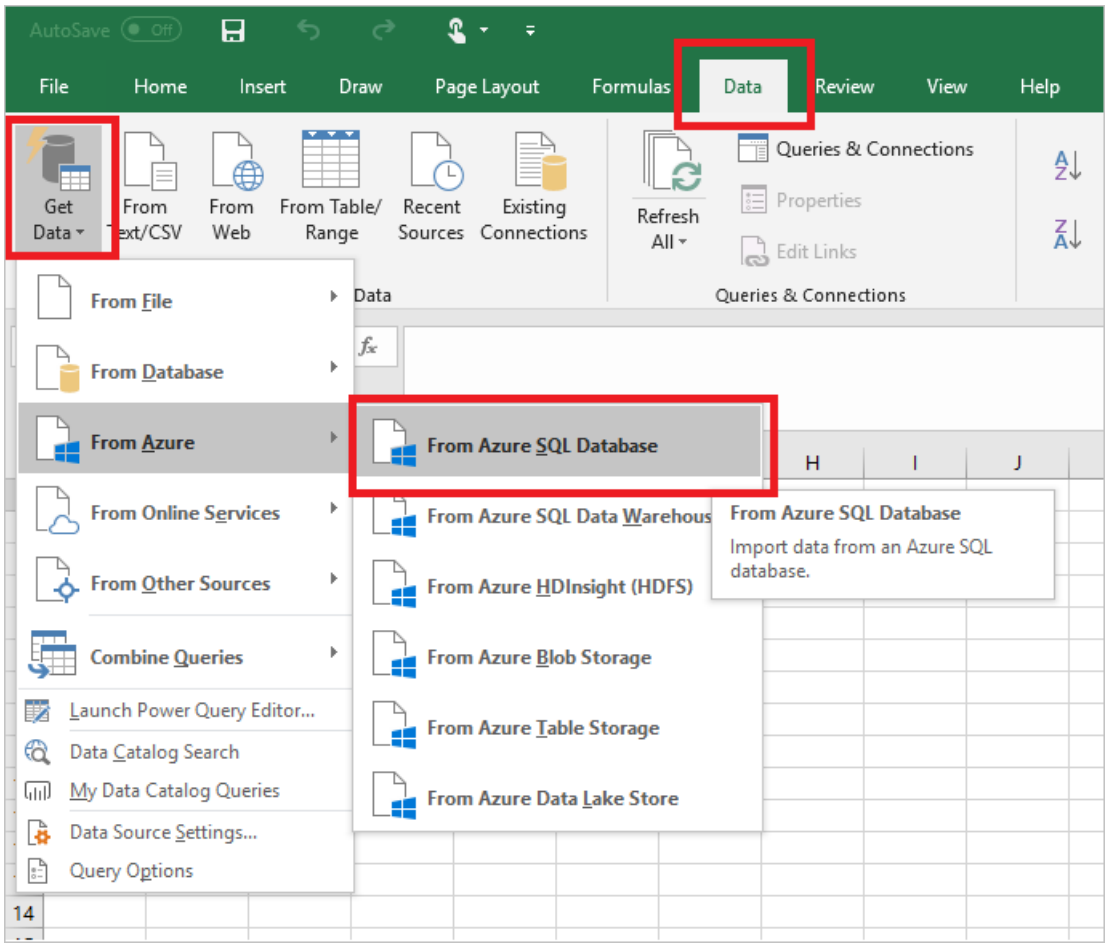
You can use Microsoft Excel to view data in Azure Blockchain Workbench's SQL DB. This article provides the steps you need to:

- Connect to the Blockchain Workbench database from Microsoft Excel
- Look at Blockchain Workbench database tables and views
- Load Blockchain Workbench view data into Excel

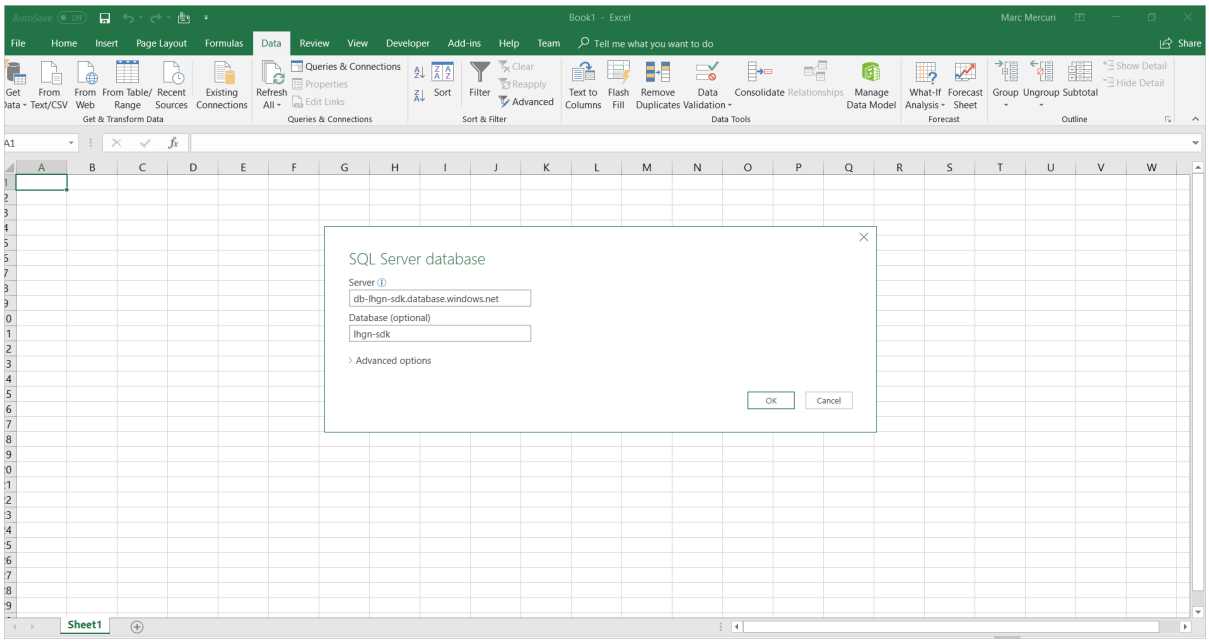
Connect to the Blockchain Workbench database

To connect to a Blockchain Workbench database:

1. Open Microsoft Excel.
2. On the **Data** tab, choose **Get Data**.
3. Select **From Azure** and then select **From Azure SQL Database**.

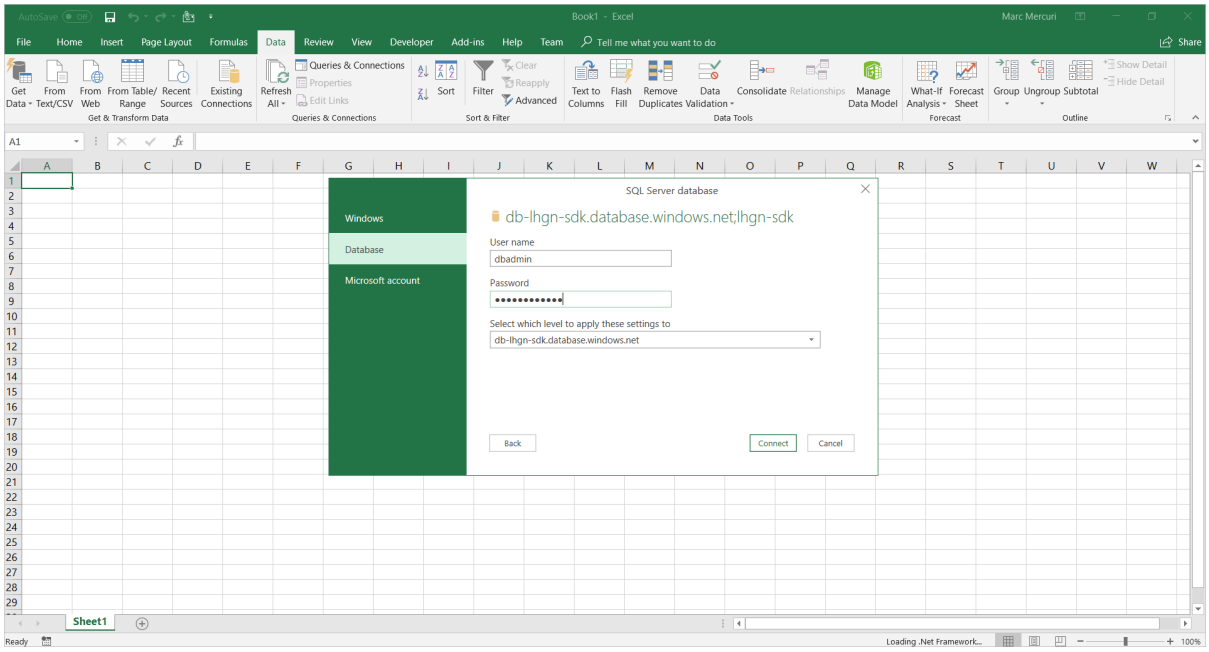


4. In the **SQL Server database** dialog box:
   - For **Server**, enter the name of the Blockchain Workbench server.
   - For **Database (optional)**, enter the name of the database.



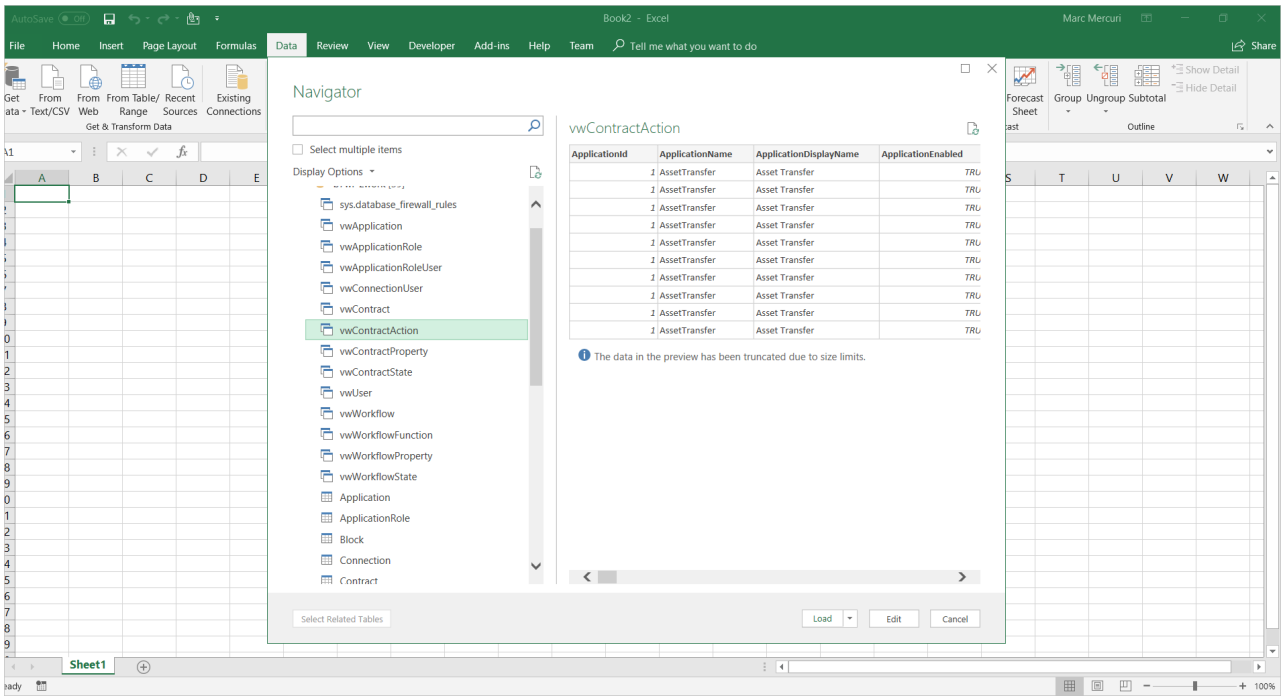
5. In the **SQL Server database** dialog navigation bar, select **Database**. Enter your **Username** and **Password**, and then select **Connect**.

> **NOTE**
>
> If you're using the credentials created during the Azure Blockchain Workbench deployment process, the **User name** is `dbadmin`. The **Password** is the one you created when you deployed the Blockchain Workbench.



Look at database tables and views

The Excel Navigator dialog opens after you connect to the database. You can use the Navigator to look at the tables and views in the database. The views are designed for reporting and their names are prefixed with **vw**.

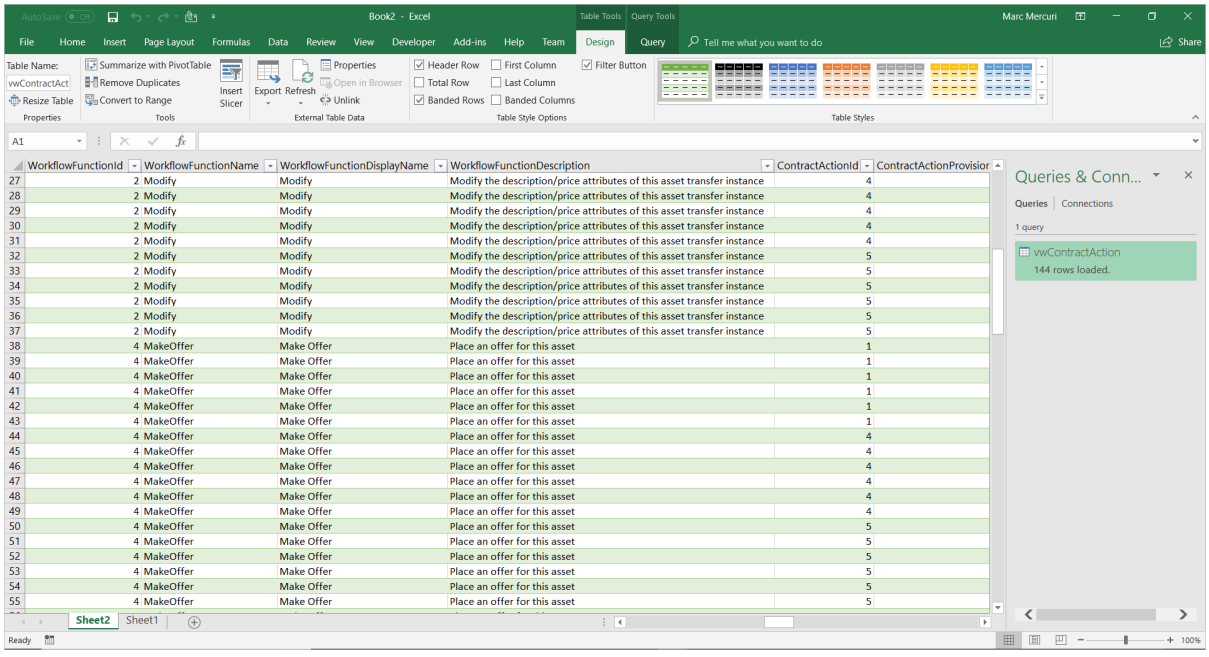


Load view data into an Excel workbook

The next example shows how you can load data from a view into an Excel workbook.

1. In the **Navigator** scroll bar, select the **vwContractAction** view. The **vwContractAction** preview shows all the actions related to a contract in the Blockchain Workbench database.

2. Select **Load** to retrieve all the data in the view and put it in your Excel workbook.



Now that you have the data loaded, you can use Excel features to create your own reports using the metadata and transaction data from the Azure Blockchain Workbench database.

Next steps

Database views in Azure Blockchain Workbench

Using Azure Blockchain Workbench data with Microsoft Power BI

5/9/2019 • 2 minutes to read • Edit Online

Microsoft Power BI provides the ability to easily generate powerful reports from SQL DB databases using Power BI Desktop and then publish them to https://www.powerbi.com.

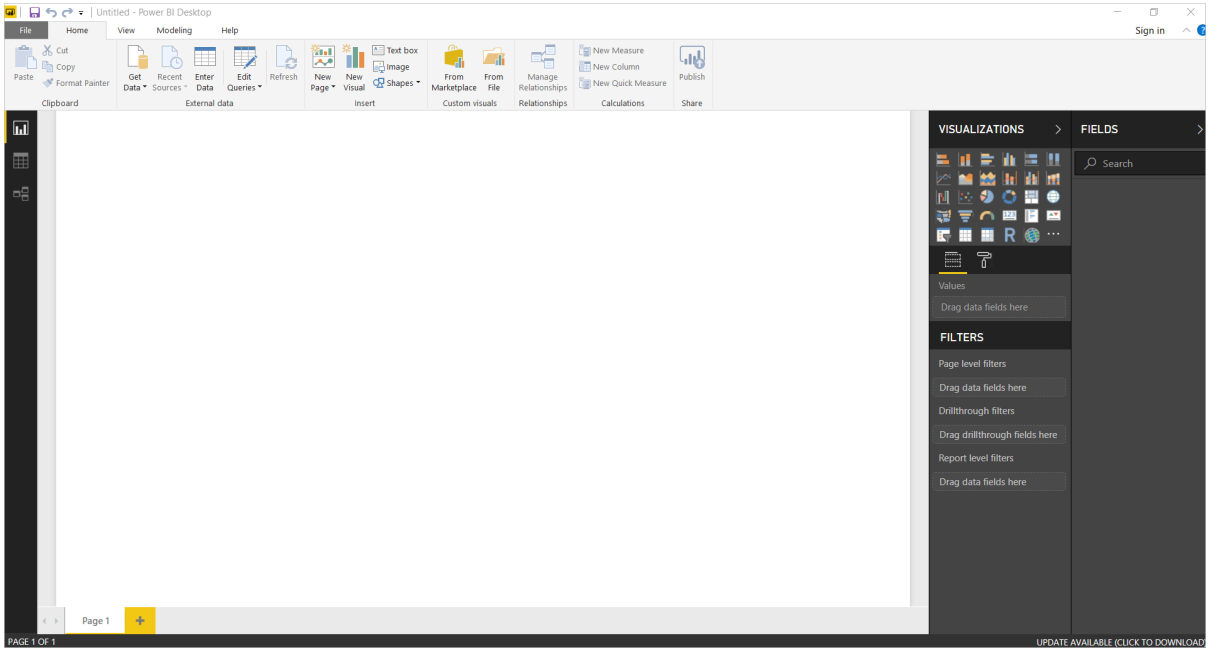
This article contains a step by step walkthrough of how to connect to Azure Blockchain Workbench's SQL Database from within PowerBI desktop, create a report, and deploy the report to powerbi.com.

Prerequisites

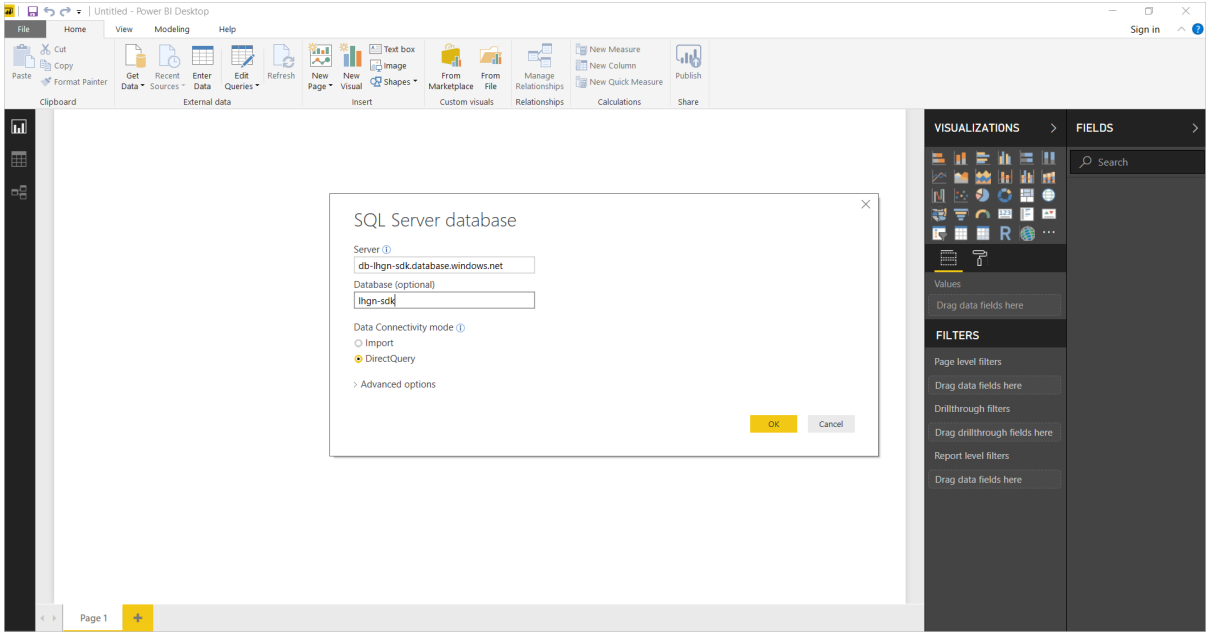
- Download Power BI Desktop.

Connecting Power BI to data in Azure Blockchain Workbench

1. Open Power BI Desktop.
2. Select **Get Data**.

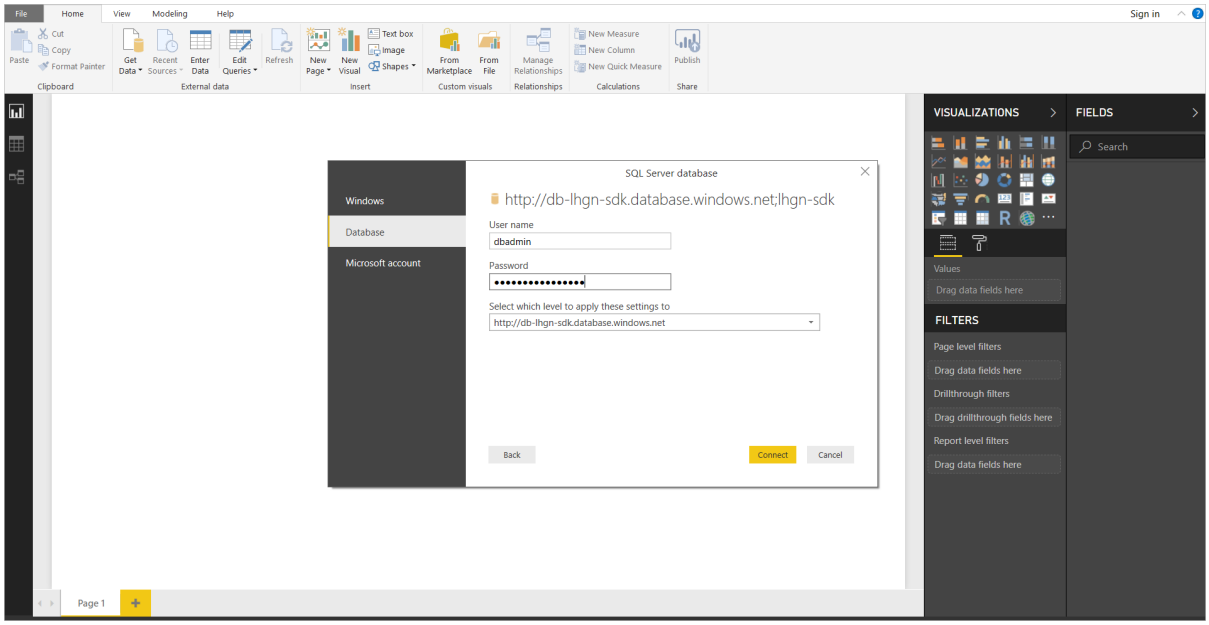


3. Select **SQL Server** from the list of data source types.
4. Provide the server and database name in the dialog. Specify if you want to import the data or perform a **DirectQuery**. Select **OK**.

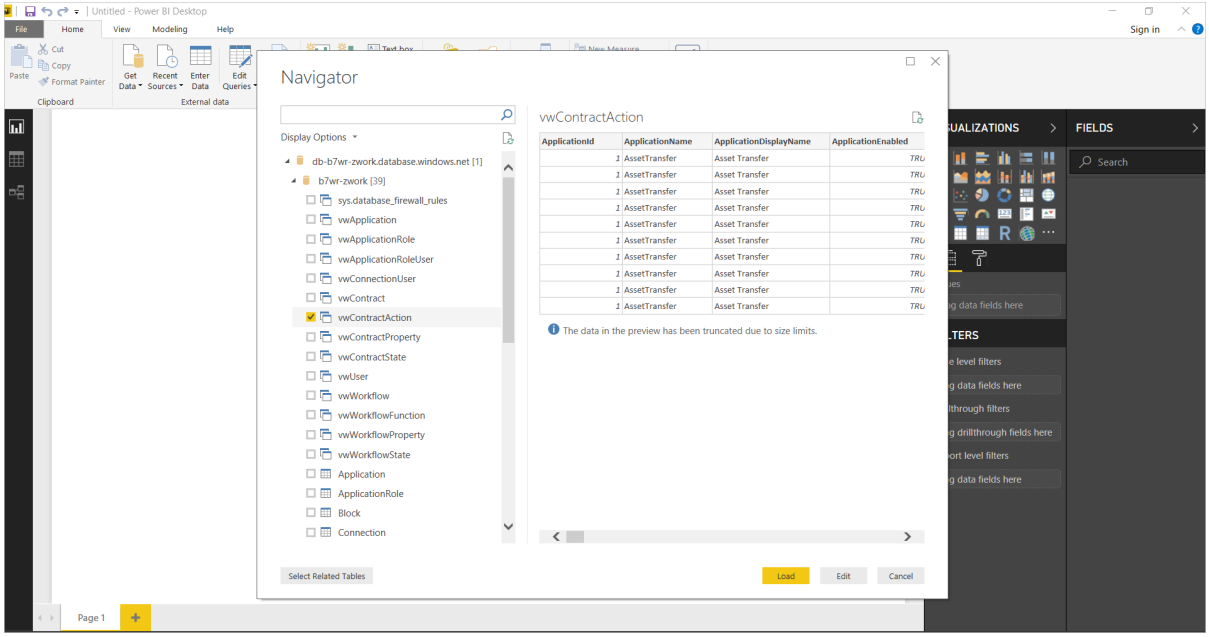


5. Provide the database credentials to access Azure Blockchain Workbench. Select **Database** and enter your credentials.

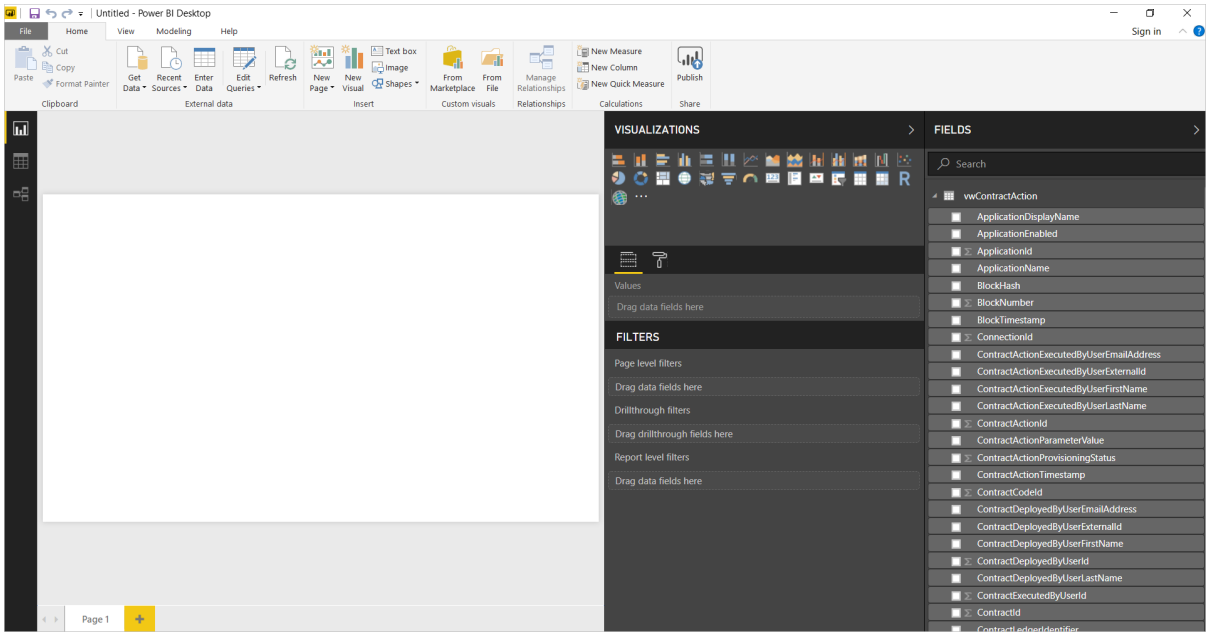
   If you are using the credentials created by the Azure Blockchain Workbench deployment process, the username is **dbadmin** and the password is the one you provided during deployment.



6. Once connected to the database, the **Navigator** dialog displays the tables and views available within the database. The views are designed for reporting and are all prefixed **vw**.



7. Select the views you wish to include. For demonstration purposes, we include **vwContractAction**, which provides details on the actions that have taken place on a contract.



You can now create and publish reports as you normally would with Power BI.

Next steps

[Database views in Azure Blockchain Workbench]

Using Azure Blockchain Workbench data with SQL Server Management Studio

5/9/2019 • 2 minutes to read • Edit Online

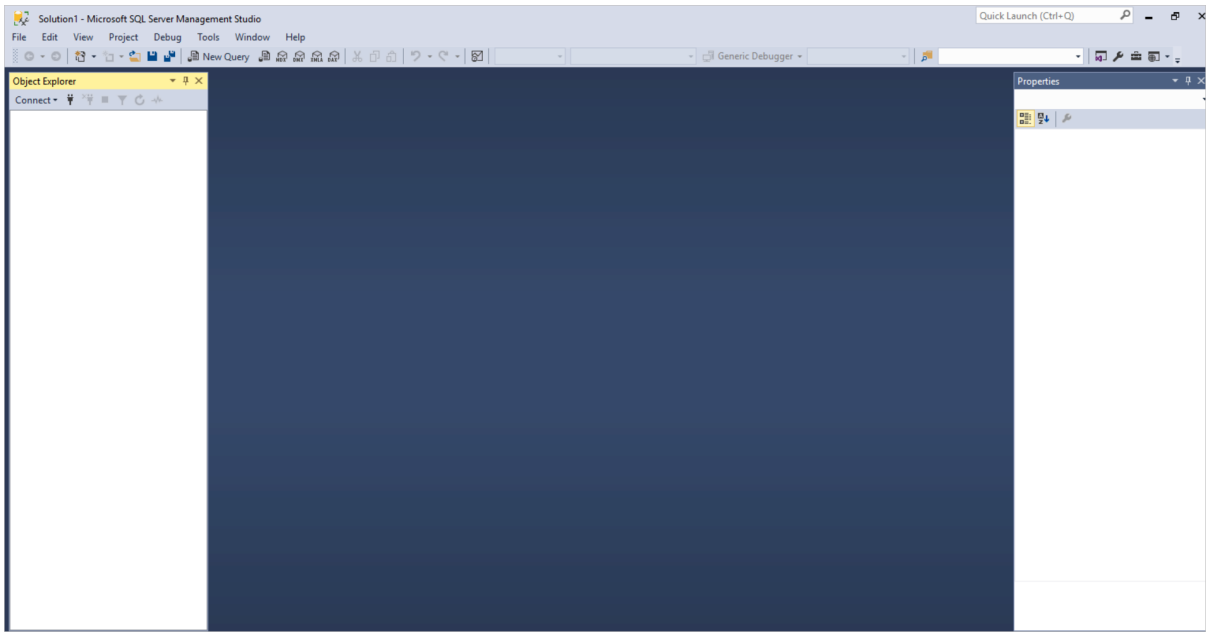
Microsoft SQL Server Management Studio provides the ability to rapidly write and test queries against Azure Blockchain Workbench's SQL DB. This section contains a step by step walkthrough of how to connect to Azure Blockchain Workbench's SQL Database from within SQL Server Management Studio.

Prerequisites

- Download SQL Server Management Studio.

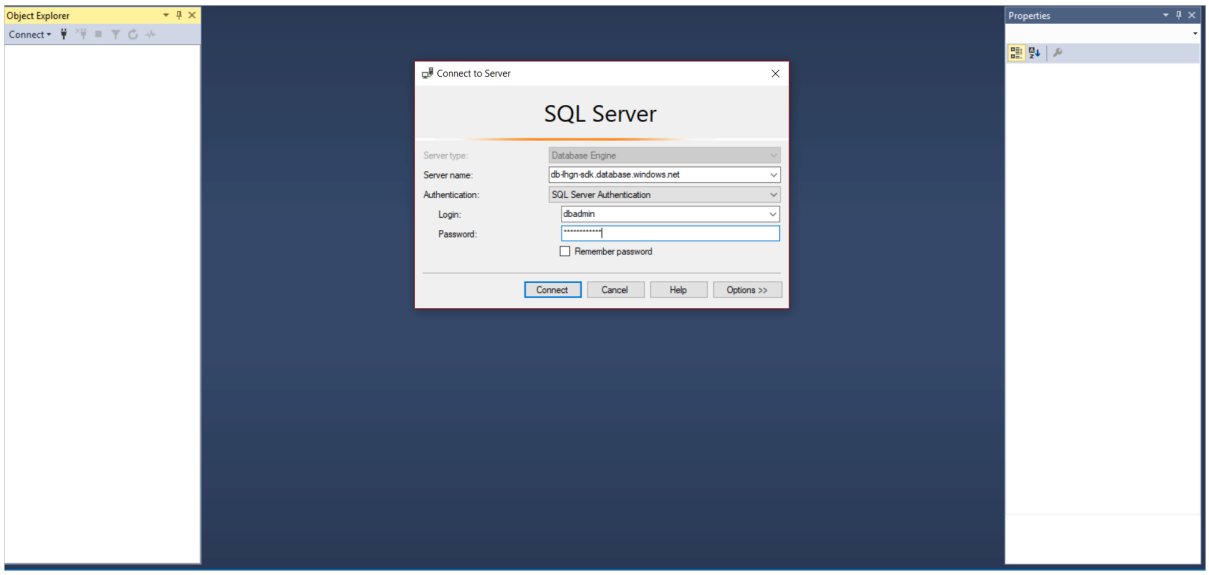
Connecting SQL Server Management Studio to data in Azure Blockchain Workbench

1. Open the SQL Server Management Studio and select **Connect**.
2. Select **Database Engine**.

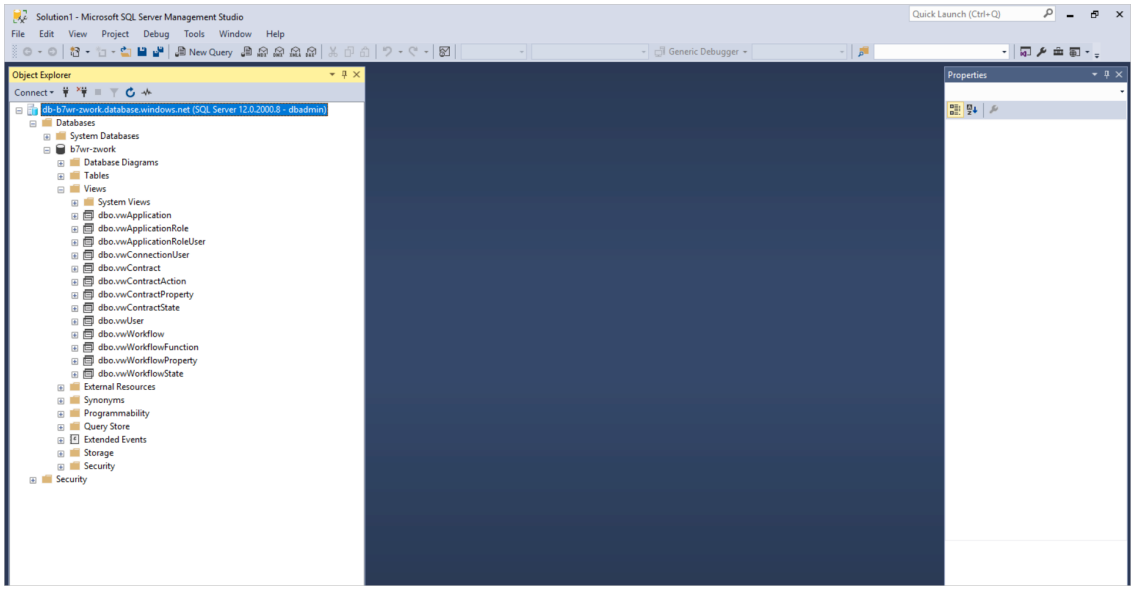


3. In the **Connect to Server** dialog, enter the server name and your database credentials.

   If you are using the credentials created by the Azure Blockchain Workbench deployment process, the username is **dbadmin** and the password is the one you provided during deployment.



a. SQL Server Management Studio displays the list of databases, database views, and stored procedures in the Azure Blockchain Workbench database.



4. To view the data associated with any of the database views, you can automatically generate a select statement using the following steps.
5. Right-click any of the database views in the Object Explorer.
6. Select **Script View as**.
7. Choose **SELECT to**.
8. Select **New Query Editor Window**.
9. A new query can be created by selecting **New Query**.

Next steps

Database views in Azure Blockchain Workbench

Azure Blockchain Workbench configuration reference

4/15/2019 • 15 minutes to read • Edit Online

Azure Blockchain Workbench applications are multi-party workflows defined by configuration metadata and smart contract code. Configuration metadata defines the high-level workflows and interaction model of the blockchain application. Smart contracts define the business logic of the blockchain application. Workbench uses configuration and smart contract code to generate blockchain application user experiences.

Configuration metadata specifies the following information for each blockchain application:

- Name and description of the blockchain application
- Unique roles for users who can act or participate within the blockchain application
- One or more workflows. Each workflow acts as a state machine to control the flow of the business logic. Workflows can be independent or interact with one another.

Each defined workflow specifies the following:

- Name and description of the workflow
- States of the workflow. Each state is a stage in the business logic's control flow.
- Actions to transition to the next state
- User roles permitted to initiate each action
- Smart contracts that represent business logic in code files

Application

A blockchain application contains configuration metadata, workflows, and user roles who can act or participate within the application.

FIELD	DESCRIPTION	REQUIRED
ApplicationName	Unique application name. The corresponding smart contract must use the same **ApplicationName** for the applicable contract class.	Yes
DisplayName	Friendly display name of the application.	Yes
Description	Description of the application.	No
ApplicationRoles	Collection of ApplicationRoles. User roles who can act or participate within the application.	Yes
Workflows	Collection of Workflows. Each workflow acts as a state machine to control the flow of the business logic.	Yes

For an example, see configuration file example.

Workflows

An application's business logic may be modeled as a state machine where taking an action causes the flow of the

business logic to move from one state to another. A workflow is a collection of such states and actions. Each workflow consists of one or more smart contracts, which represent the business logic in code files. An executable contract is an instance of a workflow.

FIELD	DESCRIPTION	REQUIRED	MAX LENGTH
Name	Unique workflow name. The corresponding smart contract must use the same **Name** for the applicable contract class.	Yes	50
DisplayName	Friendly display name of the workflow.	Yes	255
Description	Description of the workflow.	No	255
Initiators	Collection of ApplicationRoles. Roles that are assigned to users who are authorized to create contracts in the workflow.	Yes	
StartState	Name of the initial state of the workflow.	Yes	
Properties	Collection of identifiers. Represents data that can be read off-chain or visualized in a user experience tool.	Yes	
Constructor	Defines input parameters for creating an instance of the workflow.	Yes	
Functions	A collection of functions that can be executed in the workflow.	Yes	
States	A collection of workflow states.	Yes	

For an example, see configuration file example.

Type

Supported data types.

TYPE	DESCRIPTION
address	Blockchain address type, such as *contracts* or *users*.
array	Single level array of type integer, bool, money, or time. Arrays can be static or dynamic. Use **ElementType** to specify the datatype of the elements within the array. See example configuration.

TYPE	DESCRIPTION
bool	Boolean data type.
contract	Address of type contract.
enum	Enumerated set of named values. When using the enum type, you also specify a list of EnumValues. Each value is limited to 255 characters. Valid value characters include upper and lower case letters (A-Z, a-z) and numbers (0-9). See example configuration and use in Solidity.
int	Integer data type.
money	Money data type.
state	Workflow state.
string	String data type. 4000 character maximum. See example configuration.
user	Address of type user.
time	Time data type.
`[ Application Role Name ]`	Any name specified in application role. Limits users to be of that role type.

Example configuration of type array

```
{
  "Name": "Quotes",
  "Description": "Market quotes",
  "DisplayName": "Quotes",
  "Type": {
    "Name": "array",
    "ElementType": {
        "Name": "int"
    }
  }
}
```

Using a property of type array

If you define a property as type array in configuration, you need to include an explicit get function to return the public property of the array type in Solidity. For example:

```
function GetQuotes() public constant returns (int[]) {
     return Quotes;
}
```

Example configuration of type string

```
{
  "Name": "description",
  "Description": "Descriptive text",
  "DisplayName": "Description",
  "Type": {
    "Name": "string"
  }
}
```

Example configuration of type enum

```
{
  "Name": "PropertyType",
  "DisplayName": "Property Type",
  "Description": "The type of the property",
  "Type": {
    "Name": "enum",
    "EnumValues": ["House", "Townhouse", "Condo", "Land"]
  }
}
```

Using enumeration type in Solidity

Once an enum is defined in configuration, you can use enumeration types in Solidity. For example, you can define an enum called PropertyTypeEnum.

```
enum PropertyTypeEnum {House, Townhouse, Condo, Land} PropertyTypeEnum public PropertyType;
```

The list of strings needs to match between the configuration and smart contract to be valid and consistent declarations in Blockchain Workbench.

Assignment example:

```
PropertyType = PropertyTypeEnum.Townhouse;
```

Function parameter example:

```
function AssetTransfer(string description, uint256 price, PropertyTypeEnum propertyType) public
{
    InstanceOwner = msg.sender;
    AskingPrice = price;
    Description = description;
    PropertyType = propertyType;
    State = StateType.Active;
    ContractCreated();
}
```

Constructor

Defines input parameters for an instance of a workflow.

FIELD	DESCRIPTION	REQUIRED
Parameters	Collection of identifiers required to initiate a smart contract.	Yes

Constructor example

```
{
  "Parameters": [
    {
      "Name": "description",
      "Description": "The description of this asset",
      "DisplayName": "Description",
      "Type": {
        "Name": "string"
      }
    },
    {
      "Name": "price",
      "Description": "The price of this asset",
      "DisplayName": "Price",
      "Type": {
        "Name": "money"
      }
    }
  ]
}
```

Functions

Defines functions that can be executed on the workflow.

FIELD	DESCRIPTION	REQUIRED	MAX LENGTH
Name	The unique name of the function. The corresponding smart contract must use the same **Name** for the applicable function.	Yes	50
DisplayName	Friendly display name of the function.	Yes	255
Description	Description of the function	No	255
Parameters	Collection of identifiers corresponding to the parameters of the function.	Yes	

Functions example

```
"Functions": [
  {
    "Name": "Modify",
    "DisplayName": "Modify",
    "Description": "Modify the description/price attributes of this asset transfer instance",
    "Parameters": [
      {
        "Name": "description",
        "Description": "The new description of the asset",
        "DisplayName": "Description",
        "Type": {
          "Name": "string"
        }
      },
      {
        "Name": "price",
        "Description": "The new price of the asset",
        "DisplayName": "Price",
        "Type": {
          "Name": "money"
        }
      }
    ]
  },
  {
    "Name": "Terminate",
    "DisplayName": "Terminate",
    "Description": "Used to cancel this particular instance of asset transfer",
    "Parameters": []
  }
]
```

States

A collection of unique states within a workflow. Each state captures a step in the business logic's control flow.

FIELD	DESCRIPTION	REQUIRED	MAX LENGTH
Name	Unique name of the state. The corresponding smart contract must use the same **Name** for the applicable state.	Yes	50
DisplayName	Friendly display name of the state.	Yes	255
Description	Description of the state.	No	255
PercentComplete	An integer value displayed in the Blockchain Workbench user interface to show the progress within the business logic control flow.	Yes	
Style	Visual hint indicating whether the state represents a success or failure state. There are two valid values: `Success` or `Failure`.	Yes	

FIELD	DESCRIPTION	REQUIRED	MAX LENGTH
Transitions	Collection of available transitions from the current state to the next set of states.	No	

States example

```
"States": [
    {
      "Name": "Active",
      "DisplayName": "Active",
      "Description": "The initial state of the asset transfer workflow",
      "PercentComplete": 20,
      "Style": "Success",
      "Transitions": [
        {
          "AllowedRoles": [],
          "AllowedInstanceRoles": [ "InstanceOwner" ],
          "Description": "Cancels this instance of asset transfer",
          "Function": "Terminate",
          "NextStates": [ "Terminated" ],
          "DisplayName": "Terminate Offer"
        },
        {
          "AllowedRoles": [ "Buyer" ],
          "AllowedInstanceRoles": [],
          "Description": "Make an offer for this asset",
          "Function": "MakeOffer",
          "NextStates": [ "OfferPlaced" ],
          "DisplayName": "Make Offer"
        },
        {
          "AllowedRoles": [],
          "AllowedInstanceRoles": [ "InstanceOwner" ],
          "Description": "Modify attributes of this asset transfer instance",
          "Function": "Modify",
          "NextStates": [ "Active" ],
          "DisplayName": "Modify"
        }
      ]
    },
    {
      "Name": "Accepted",
      "DisplayName": "Accepted",
      "Description": "Asset transfer process is complete",
      "PercentComplete": 100,
      "Style": "Success",
      "Transitions": []
    },
    {
      "Name": "Terminated",
      "DisplayName": "Terminated",
      "Description": "Asset transfer has been canceled",
      "PercentComplete": 100,
      "Style": "Failure",
      "Transitions": []
    }
  ]
```

Transitions

Available actions to the next state. One or more user roles may perform an action at each state, where an action may transition a state to another state in the workflow.

FIELD	DESCRIPTION	REQUIRED
AllowedRoles	List of applications roles allowed to initiate the transition. All users of the specified role may be able to perform the action.	No

FIELD	DESCRIPTION	REQUIRED
AllowedInstanceRoles	List of user roles participating or specified in the smart contract allowed to initiate the transition. Instance roles are defined in **Properties** within workflows. AllowedInstanceRoles represent a user participating in an instance of a smart contract. AllowedInstanceRoles give you the ability to restrict taking an action to a user role in a contract instance. For example, you may only want to allow the user who created the contract (InstanceOwner) to be able to terminate rather than all users in role type (Owner) if you specified the role in AllowedRoles.	No
DisplayName	Friendly display name of the transition.	Yes
Description	Description of the transition.	No
Function	The name of the function to initiate the transition.	Yes
NextStates	A collection of potential next states after a successful transition.	Yes

Transitions example

```
"Transitions": [
  {
    "AllowedRoles": [],
    "AllowedInstanceRoles": [ "InstanceOwner" ],
    "Description": "Cancels this instance of asset transfer",
    "Function": "Terminate",
    "NextStates": [ "Terminated" ],
    "DisplayName": "Terminate Offer"
  },
  {
    "AllowedRoles": [ "Buyer" ],
    "AllowedInstanceRoles": [],
    "Description": "Make an offer for this asset",
    "Function": "MakeOffer",
    "NextStates": [ "OfferPlaced" ],
    "DisplayName": "Make Offer"
  },
  {
    "AllowedRoles": [],
    "AllowedInstanceRoles": [ "InstanceOwner" ],
    "Description": "Modify attributes of this asset transfer instance",
    "Function": "Modify",
    "NextStates": [ "Active" ],
    "DisplayName": "Modify"
  }
]
```

Application roles

Application roles define a set of roles that can be assigned to users who want to act or participate within the application. Application roles can be used to restrict actions and participation within the blockchain application and corresponding workflows.

FIELD	DESCRIPTION	REQUIRED	MAX LENGTH
Name	The unique name of the application role. The corresponding smart contract must use the same **Name** for the applicable role. Base type names are reserved. You cannot name an application role with the same name as Type	Yes	50
Description	Description of the application role.	No	255

Application roles example

```
"ApplicationRoles": [
  {
    "Name": "Appraiser",
    "Description": "User that signs off on the asset price"
  },
  {
    "Name": "Buyer",
    "Description": "User that places an offer on an asset"
  }
]
```

Identifiers

Identifiers represent a collection of information used to describe workflow properties, constructor, and function parameters.

FIELD	DESCRIPTION	REQUIRED	MAX LENGTH
Name	The unique name of the property or parameter. The corresponding smart contract must use the same **Name** for the applicable property or parameter.	Yes	50
DisplayName	Friendly display name for the property or parameter.	Yes	255
Description	Description of the property or parameter.	No	255

Identifiers example

```
"Properties": [
  {
    "Name": "State",
    "DisplayName": "State",
    "Description": "Holds the state of the contract",
    "Type": {
      "Name": "state"
    }
  },
  {
    "Name": "Description",
    "DisplayName": "Description",
    "Description": "Describes the asset being sold",
    "Type": {
      "Name": "string"
    }
  }
]
```

Configuration file example

Asset transfer is a smart contract scenario for buying and selling high value assets, which require an inspector and appraiser. Sellers can list their assets by instantiating an asset transfer smart contract. Buyers can make offers by taking an action on the smart contract, and other parties can take actions to inspect or appraise the asset. Once the asset is marked both inspected and appraised, the buyer and seller will confirm the sale again before the contract is set to complete. At each point in the process, all participants have visibility into the state of the contract as it is updated.

For more information including the code files, see asset transfer sample for Azure Blockchain Workbench

The following configuration file is for the asset transfer sample:

```
{
  "ApplicationName": "AssetTransfer",
  "DisplayName": "Asset Transfer",
  "Description": "Allows transfer of assets between a buyer and a seller, with appraisal/inspection
functionality",
  "ApplicationRoles": [
    {
      "Name": "Appraiser",
      "Description": "User that signs off on the asset price"
    },
    {
      "Name": "Buyer",
      "Description": "User that places an offer on an asset"
    },
    {
      "Name": "Inspector",
      "Description": "User that inspects the asset and signs off on inspection"
    },
    {
      "Name": "Owner",
      "Description": "User that signs off on the asset price"
    }
  ],
  "Workflows": [
    {
      "Name": "AssetTransfer",
      "DisplayName": "Asset Transfer",
      "Description": "Handles the business logic for the asset transfer scenario",
      "Initiators": [ "Owner" ],
      "StartState":  "Active",
      "Properties": [
```

```
      {
        "Name": "State",
        "DisplayName": "State",
        "Description": "Holds the state of the contract",
        "Type": {
          "Name": "state"
        }
      },
      {
        "Name": "Description",
        "DisplayName": "Description",
        "Description": "Describes the asset being sold",
        "Type": {
          "Name": "string"
        }
      },
      {
        "Name": "AskingPrice",
        "DisplayName": "Asking Price",
        "Description": "The asking price for the asset",
        "Type": {
          "Name": "money"
        }
      },
      {
        "Name": "OfferPrice",
        "DisplayName": "Offer Price",
        "Description": "The price being offered for the asset",
        "Type": {
          "Name": "money"
        }
      },
      {
        "Name": "InstanceAppraiser",
        "DisplayName": "Instance Appraiser",
        "Description": "The user that appraises the asset",
        "Type": {
          "Name": "Appraiser"
        }
      },
      {
        "Name": "InstanceBuyer",
        "DisplayName": "Instance Buyer",
        "Description": "The user that places an offer for this asset",
        "Type": {
          "Name": "Buyer"
        }
      },
      {
        "Name": "InstanceInspector",
        "DisplayName": "Instance Inspector",
        "Description": "The user that inspects this asset",
        "Type": {
          "Name": "Inspector"
        }
      },
      {
        "Name": "InstanceOwner",
        "DisplayName": "Instance Owner",
        "Description": "The seller of this particular asset",
        "Type": {
          "Name": "Owner"
        }
      }
    ],
    "Constructor": {
      "Parameters": [
        {
          "Name": "description",
```

```
        "Description": "The description of this asset",
        "DisplayName": "Description",
        "Type": {
          "Name": "string"
        }
      },
      {
        "Name": "price",
        "Description": "The price of this asset",
        "DisplayName": "Price",
        "Type": {
          "Name": "money"
        }
      }
    ]
  },
  "Functions": [
    {
      "Name": "Modify",
      "DisplayName": "Modify",
      "Description": "Modify the description/price attributes of this asset transfer instance",
      "Parameters": [
        {
          "Name": "description",
          "Description": "The new description of the asset",
          "DisplayName": "Description",
          "Type": {
            "Name": "string"
          }
        },
        {
          "Name": "price",
          "Description": "The new price of the asset",
          "DisplayName": "Price",
          "Type": {
            "Name": "money"
          }
        }
      ]
    },
    {
      "Name": "Terminate",
      "DisplayName": "Terminate",
      "Description": "Used to cancel this particular instance of asset transfer",
      "Parameters": []
    },
    {
      "Name": "MakeOffer",
      "DisplayName": "Make Offer",
      "Description": "Place an offer for this asset",
      "Parameters": [
        {
          "Name": "inspector",
          "Description": "Specify a user to inspect this asset",
          "DisplayName": "Inspector",
          "Type": {
            "Name": "Inspector"
          }
        },
        {
          "Name": "appraiser",
          "Description": "Specify a user to appraise this asset",
          "DisplayName": "Appraiser",
          "Type": {
            "Name": "Appraiser"
          }
        },
        {
          "Name": "offerPrice",
```

```
            "Description": "Specify your offer price for this asset",
            "DisplayName": "Offer Price",
            "Type": {
              "Name": "money"
            }
          }
        ]
      },
      {
        "Name": "Reject",
        "DisplayName": "Reject",
        "Description": "Reject the user's offer",
        "Parameters": []
      },
      {
        "Name": "AcceptOffer",
        "DisplayName": "Accept Offer",
        "Description": "Accept the user's offer",
        "Parameters": []
      },
      {
        "Name": "RescindOffer",
        "DisplayName": "Rescind Offer",
        "Description": "Rescind your placed offer",
        "Parameters": []
      },
      {
        "Name": "ModifyOffer",
        "DisplayName": "Modify Offer",
        "Description": "Modify the price of your placed offer",
        "Parameters": [
          {
            "Name": "offerPrice",
            "DisplayName": "Price",
            "Type": {
              "Name": "money"
            }
          }
        ]
      },
      {
        "Name": "Accept",
        "DisplayName": "Accept",
        "Description": "Accept the inspection/appraisal results",
        "Parameters": []
      },
      {
        "Name": "MarkInspected",
        "DisplayName": "Mark Inspected",
        "Description": "Mark the asset as inspected",
        "Parameters": []
      },
      {
        "Name": "MarkAppraised",
        "DisplayName": "Mark Appraised",
        "Description": "Mark the asset as appraised",
        "Parameters": []
      }
    ],
    "States": [
      {
        "Name": "Active",
        "DisplayName": "Active",
        "Description": "The initial state of the asset transfer workflow",
        "PercentComplete": 20,
        "Style": "Success",
        "Transitions": [
          {
            "AllowedRoles": [],
```

```
        "AllowedInstanceRoles": [ "InstanceOwner" ],
        "Description": "Cancels this instance of asset transfer",
        "Function": "Terminate",
        "NextStates": [ "Terminated" ],
        "DisplayName": "Terminate Offer"
      },
      {
        "AllowedRoles": [ "Buyer" ],
        "AllowedInstanceRoles": [],
        "Description": "Make an offer for this asset",
        "Function": "MakeOffer",
        "NextStates": [ "OfferPlaced" ],
        "DisplayName": "Make Offer"
      },
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceOwner" ],
        "Description": "Modify attributes of this asset transfer instance",
        "Function": "Modify",
        "NextStates": [ "Active" ],
        "DisplayName": "Modify"
      }
    ]
  },
  {
    "Name": "OfferPlaced",
    "DisplayName": "Offer Placed",
    "Description": "Offer has been placed for the asset",
    "PercentComplete": 30,
    "Style": "Success",
    "Transitions": [
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceOwner" ],
        "Description": "Accept the proposed offer for the asset",
        "Function": "AcceptOffer",
        "NextStates": [ "PendingInspection" ],
        "DisplayName": "Accept Offer"
      },
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceOwner" ],
        "Description": "Reject the proposed offer for the asset",
        "Function": "Reject",
        "NextStates": [ "Active" ],
        "DisplayName": "Reject"
      },
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceOwner" ],
        "Description": "Cancel this instance of asset transfer",
        "Function": "Terminate",
        "NextStates": [ "Terminated" ],
        "DisplayName": "Terminate"
      },
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceBuyer" ],
        "Description": "Rescind the offer you previously placed for this asset",
        "Function": "RescindOffer",
        "NextStates": [ "Active" ],
        "DisplayName": "Rescind Offer"
      },
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceBuyer" ],
        "Description": "Modify the price that you specified for your offer",
        "Function": "ModifyOffer",
        "NextStates": [ "OfferPlaced" ],
```

```
        "NextStates": [ "OfferPlaced" ],
        "DisplayName": "Modify Offer"
      }
    ]
  },
  {
    "Name": "PendingInspection",
    "DisplayName": "Pending Inspection",
    "Description": "Asset is pending inspection",
    "PercentComplete": 40,
    "Style": "Success",
    "Transitions": [
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceOwner" ],
        "Description": "Reject the offer",
        "Function": "Reject",
        "NextStates": [ "Active" ],
        "DisplayName": "Reject"
      },
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceOwner" ],
        "Description": "Cancel the offer",
        "Function": "Terminate",
        "NextStates": [ "Terminated" ],
        "DisplayName": "Terminate"
      },
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceBuyer" ],
        "Description": "Rescind the offer you placed for this asset",
        "Function": "RescindOffer",
        "NextStates": [ "Active" ],
        "DisplayName": "Rescind Offer"
      },
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceInspector" ],
        "Description": "Mark this asset as inspected",
        "Function": "MarkInspected",
        "NextStates": [ "Inspected" ],
        "DisplayName": "Mark Inspected"
      },
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceAppraiser" ],
        "Description": "Mark this asset as appraised",
        "Function": "MarkAppraised",
        "NextStates": [ "Appraised" ],
        "DisplayName": "Mark Appraised"
      }
    ]
  },
  {
    "Name": "Inspected",
    "DisplayName": "Inspected",
    "PercentComplete": 45,
    "Style": "Success",
    "Transitions": [
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceOwner" ],
        "Description": "Reject the offer",
        "Function": "Reject",
        "NextStates": [ "Active" ],
        "DisplayName": "Reject"
      },
      {
        "AllowedRoles": [],
```

```
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceOwner" ],
        "Description": "Cancel the offer",
        "Function": "Terminate",
        "NextStates": [ "Terminated" ],
        "DisplayName": "Terminate"
      },
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceBuyer" ],
        "Description": "Rescind the offer you placed for this asset",
        "Function": "RescindOffer",
        "NextStates": [ "Active" ],
        "DisplayName": "Rescind Offer"
      },
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceAppraiser" ],
        "Description": "Mark this asset as appraised",
        "Function": "MarkAppraised",
        "NextStates": [ "NotionalAcceptance" ],
        "DisplayName": "Mark Appraised"
      }
    ]
  },
  {
    "Name": "Appraised",
    "DisplayName": "Appraised",
    "Description": "Asset has been appraised, now awaiting inspection",
    "PercentComplete": 45,
    "Style": "Success",
    "Transitions": [
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceOwner" ],
        "Description": "Reject the offer",
        "Function": "Reject",
        "NextStates": [ "Active" ],
        "DisplayName": "Reject"
      },
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceOwner" ],
        "Description": "Cancel the offer",
        "Function": "Terminate",
        "NextStates": [ "Terminated" ],
        "DisplayName": "Terminate"
      },
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceBuyer" ],
        "Description": "Rescind the offer you placed for this asset",
        "Function": "RescindOffer",
        "NextStates": [ "Active" ],
        "DisplayName": "Rescind Offer"
      },
      {
        "AllowedRoles": [],
        "AllowedInstanceRoles": [ "InstanceInspector" ],
        "Description": "Mark the asset as inspected",
        "Function": "MarkInspected",
        "NextStates": [ "NotionalAcceptance" ],
        "DisplayName": "Mark Inspected"
      }
    ]
  },
  {
    "Name": "NotionalAcceptance",
    "DisplayName": "Notional Acceptance",
```

```
          "Description": "Asset has been inspected and appraised, awaiting final sign-off from buyer and
seller",
          "PercentComplete": 50,
          "Style": "Success",
          "Transitions": [
            {
              "AllowedRoles": [],
              "AllowedInstanceRoles": [ "InstanceOwner" ],
              "Description": "Sign-off on inspection and appraisal",
              "Function": "Accept",
              "NextStates": [ "SellerAccepted" ],
              "DisplayName": "SellerAccept"
            },
            {
              "AllowedRoles": [],
              "AllowedInstanceRoles": [ "InstanceOwner" ],
              "Description": "Reject the proposed offer for the asset",
              "Function": "Reject",
              "NextStates": [ "Active" ],
              "DisplayName": "Reject"
            },
            {
              "AllowedRoles": [],
              "AllowedInstanceRoles": [ "InstanceOwner" ],
              "Description": "Cancel this instance of asset transfer",
              "Function": "Terminate",
              "NextStates": [ "Terminated" ],
              "DisplayName": "Terminate"
            },
            {
              "AllowedRoles": [],
              "AllowedInstanceRoles": [ "InstanceBuyer" ],
              "Description": "Sign-off on inspection and appraisal",
              "Function": "Accept",
              "NextStates": [ "BuyerAccepted" ],
              "DisplayName": "BuyerAccept"
            },
            {
              "AllowedRoles": [],
              "AllowedInstanceRoles": [ "InstanceBuyer" ],
              "Description": "Rescind the offer you placed for this asset",
              "Function": "RescindOffer",
              "NextStates": [ "Active" ],
              "DisplayName": "Rescind Offer"
            }
          ]
        },
        {
          "Name": "BuyerAccepted",
          "DisplayName": "Buyer Accepted",
          "Description": "Buyer has signed-off on inspection and appraisal",
          "PercentComplete": 75,
          "Style": "Success",
          "Transitions": [
            {
              "AllowedRoles": [],
              "AllowedInstanceRoles": [ "InstanceOwner" ],
              "Description": "Sign-off on inspection and appraisal",
              "Function": "Accept",
              "NextStates": [ "SellerAccepted" ],
              "DisplayName": "Accept"
            },
            {
              "AllowedRoles": [],
              "AllowedInstanceRoles": [ "InstanceOwner" ],
              "Description": "Reject the proposed offer for the asset",
              "Function": "Reject",
              "NextStates": [ "Active" ],
              "DisplayName": "Reject"
```

```
            },
            {
              "AllowedRoles": [],
              "AllowedInstanceRoles": [ "InstanceOwner" ],
              "Description": "Cancel this instance of asset transfer",
              "Function": "Terminate",
              "NextStates": [ "Terminated" ],
              "DisplayName": "Terminate"
            }
          ]
        },
        {
          "Name": "SellerAccepted",
          "DisplayName": "Seller Accepted",
          "Description": "Seller has signed-off on inspection and appraisal",
          "PercentComplete": 75,
          "Style": "Success",
          "Transitions": [
            {
              "AllowedRoles": [],
              "AllowedInstanceRoles": [ "InstanceBuyer" ],
              "Description": "Sign-off on inspection and appraisal",
              "Function": "Accept",
              "NextStates": [ "Accepted" ],
              "DisplayName": "Accept"
            },
            {
              "AllowedRoles": [],
              "AllowedInstanceRoles": [ "InstanceBuyer" ],
              "Description": "Rescind the offer you placed for this asset",
              "Function": "RescindOffer",
              "NextStates": [ "Active" ],
              "DisplayName": "Rescind Offer"
            }
          ]
        },
        {
          "Name": "Accepted",
          "DisplayName": "Accepted",
          "Description": "Asset transfer process is complete",
          "PercentComplete": 100,
          "Style": "Success",
          "Transitions": []
        },
        {
          "Name": "Terminated",
          "DisplayName": "Terminated",
          "Description": "Asset transfer has been canceled",
          "PercentComplete": 100,
          "Style": "Failure",
          "Transitions": []
        }
      ]
    }
  ]
}
```

Next steps

Azure Blockchain Workbench REST API reference

Database views in Azure Blockchain Workbench

5/30/2019 • 23 minutes to read • Edit Online

Azure Blockchain Workbench delivers data from distributed ledgers to an *off-chain* SQL DB database. The off-chain database makes it possible to use SQL and existing tools, such as SQL Server Management Studio, to interact with blockchain data.

Azure Blockchain Workbench provides a set of database views that provide access to data that will be helpful when performing your queries. These views are heavily denormalized to make it easy to quickly get started building reports, analytics, and otherwise consume blockchain data with existing tools and without having to retrain database staff.

This section includes an overview of the database views and the data they contain.

> **NOTE**
> Any direct usage of database tables found in the database outside of these views, while possible, is not supported.

vwApplication

This view provides details on **Applications** that have been uploaded to Azure Blockchain Workbench.

NAME	TYPE	CAN BE NULL	DESCRIPTION
ApplicationId	int	No	A unique identifier for the application
ApplicationName	nvarchar(50)	No	The name of the application
ApplicationDescription	nvarchar(255)	Yes	A description of the application
ApplicationDisplayName	nvarchar(255)	No	The name to be displayed in a user interface
ApplicationEnabled	bit	No	Identifies if the application is currently enabled **Note:** Even though an application can be reflected as disabled in the database, associated contracts remain on the blockchain and data about those contracts remain in the database.
UploadedDtTm	datetime2(7)	No	The date and time a contract was uploaded
UploadedByUserId	int	No	The ID of the user who uploaded the application

NAME	TYPE	CAN BE NULL	DESCRIPTION
UploadedByUserExternalId	nvarchar(255)	No	The external identifier for the user who uploaded the application. By default, this ID is the user from the Azure Active Directory for the consortium.
UploadedByUserProvisioningStatus	int	No	Identifies the current status of provisioning process for the user. Possible values are: 0 – User has been created by the API 1 – A key has been associated with the user in the database 2 – The user is fully provisioned
UploadedByUserFirstName	nvarchar(50)	Yes	The first name of the user who uploaded the contract
UploadedByUserLastName	nvarchar(50)	Yes	The last name of the user who uploaded the contract
UploadedByUserEmailAddress	nvarchar(255)	Yes	The email address of the user who uploaded the contract

vwApplicationRole

This view provides details on the roles that have been defined in Azure Blockchain Workbench applications.

In an *Asset Transfer* application, for example, roles such as *Buyer* and *Seller* roles may be defined.

NAME	TYPE	CAN BE NULL	DESCRIPTION
ApplicationId	int	No	A unique identifier for the application
ApplicationName	nvarchar(50)	No	The name of the application
ApplicationDescription	nvarchar(255)	Yes	A description of the application
ApplicationDisplayName	nvarchar(255)	No	The name to be displayed in a user interface
RoleId	int	No	A unique identifier for a role in the application
RoleName	nvarchar50)	No	The name of the role
RoleDescription	description(255)	Yes	A description of the role

vwApplicationRoleUser

This view provides details on the roles that have been defined in Azure Blockchain Workbench applications and the users associated with them.

In an *Asset Transfer* application, for example, *John Smith* may be associated with the *Buyer* role.

NAME	TYPE	CAN BE NULL	DESCRIPTION
ApplicationId	int	No	A unique identifier for the application
ApplicationName	nvarchar(50)	No	The name of the application
ApplicationDescription	nvarchar(255)	Yes	A description of the application
ApplicationDisplayName	nvarchar(255)	No	The name to be displayed in a user interface
ApplicationRoleId	int	No	A unique identifier for a role in the application
ApplicationRoleName	nvarchar50)	No	The name of the role
ApplicationRoleDescription	nvarchar(255)	Yes	A description of the role
UserId	int	No	The ID of the user associated with the role
UserExternalId	nvarchar(255)	No	The external identifier for the user who is associated with the role. By default, this ID is the user from the Azure Active Directory for the consortium.
UserProvisioningStatus	int	No	Identifies the current status of provisioning process for the user. Possible values are: 0 – User has been created by the API 1 – A key has been associated with the user in the database 2 – The user is fully provisioned
UserFirstName	nvarchar(50)	Yes	The first name of the user who is associated with the role
UserLastName	nvarchar(255)	Yes	The last name of the user who is associated with the role

NAME	TYPE	CAN BE NULL	DESCRIPTION
UserEmailAddress	nvarchar(255)	Yes	The email address of the user who is associated with the role

vwConnectionUser

This view provides details on the connections defined in Azure Blockchain Workbench and the users associated with them. For each connection, this view contains the following data:

- Associated ledger details
- Associated user information

NAME	TYPE	CAN BE NULL	DESCRIPTION
ConnectionId	int	No	The unique identifier for a connection in Azure Blockchain Workbench
ConnectionEndpointUrl	nvarchar(50)	No	The endpoint url for a connection
ConnectionFundingAccount	nvarchar(255)	Yes	The funding account associated with a connection, if applicable
LedgerId	int	No	The unique identifier for a ledger
LedgerName	nvarchar(50)	No	The name of the ledger
LedgerDisplayName	nvarchar(255)	No	The name of the ledger to display in the UI
UserId	int	No	The ID of the user associated with the connection
UserExternalId	nvarchar(255)	No	The external identifier for the user who is associated with the connection. By default, this ID is the user from the Azure Active Directory for the consortium.
UserProvisioningStatus	int	No	Identifies the current status of provisioning process for the user. Possible values are: 0 – User has been created by the API 1 – A key has been associated with the user in the database 2 – The user is fully provisioned

NAME	TYPE	CAN BE NULL	DESCRIPTION
UserFirstName	nvarchar(50)	Yes	The first name of the user who is associated with the connection
UserLastName	nvarchar(255)	Yes	The last name of the user who is associated with the connection
UserEmailAddress	nvarchar(255)	Yes	The email address of the user who is associated with the connection

vwContract

This view provides details about deployed contracts. For each contract, this view contains the following data:

- Associated application definition
- Associated workflow definition
- Associated ledger implementation for the function
- Details for the user who initiated the action
- Details related to the blockchain block and transaction

NAME	TYPE	CAN BE NULL	DESCRIPTION
ConnectionId	int	No	The unique identifier for a connection in Azure Blockchain Workbench.
ConnectionEndpointUrl	nvarchar(50)	No	The endpoint url for a connection
ConnectionFundingAccount	nvarchar(255)	Yes	The funding account associated with a connection, if applicable
LedgerId	int	No	The unique identifier for a ledger
LedgerName	nvarchar(50)	No	The name of the ledger
LedgerDisplayName	nvarchar(255)	No	The name of the ledger to display in the UI
ApplicationId	int	No	A unique identifier for the application
ApplicationName	nvarchar (50)	No	The name of the application
ApplicationDisplayName	nvarchar (255)	No	The name to be displayed in a user interface

NAME	TYPE	CAN BE NULL	DESCRIPTION
ApplicationEnabled	bit	No	Identifies if the application is currently enabled. **Note:** Even though an application can be reflected as disabled in the database, associated contracts remain on the blockchain and data about those contracts remain in the database.
WorkflowId	int	No	A unique identifier for the workflow associated with a contract
WorkflowName	nvarchar(50)	No	The name of the workflow associated with a contract
WorkflowDisplayName	nvarchar(255)	No	The name of the workflow associated with the contract displayed in the user interface
WorkflowDescription	nvarchar(255)	Yes	The description of the workflow associated with a contract
ContractCodeId	int	No	A unique identifier for the contract code associated with the contract
ContractFileName	int	No	The name of the file containing the smart contract code for this workflow.
ContractUploadedDtTm	int	No	The date and time the contract code was uploaded
ContractId	int	No	The unique identifier for the contract

NAME	TYPE	CAN BE NULL	DESCRIPTION
ContractProvisioningStatus	int	No	Identifies the current status of the provisioning process for the contract. Possible values are: 0 – The contract has been created by the API in the database 1 – The contract has been sent to the ledger 2 – The contract has been successfully deployed to the ledger 3 or 4 - The contract failed to be deployed to the ledger 5 - The contract was successfully deployed to the ledger Beginning with version 1.5, values 0 through 5 are supported. For backwards compatibility in the current release, view **vwContractV0** is available that only supports values 0 through 2.
ContractLedgerIdentifier	nvarchar (255)		The email address of the user who deployed the contract
ContractDeployedByUserId	int	No	An external identifier for the user who deployed the contract. By default, this ID is the guid representing the Azure Active Directory ID for the user.
ContractDeployedByUserExternalId	nvarchar(255)	No	An external identifier for the user that deployed the contract. By default, this ID is the guid representing the Azure Active Directory ID for the user.
ContractDeployedByUserProvisioningStatus	int	No	Identifies the current status of the provisioning process for the user. Possible values are: 0 – user has been created by the API 1 – A key has been associated with the user in the database 2 – The user is fully provisioned
ContractDeployedByUserFirstName	nvarchar(50)	Yes	The first name of the user who deployed the contract

NAME	TYPE	CAN BE NULL	DESCRIPTION
ContractDeployedByUserLastName	nvarchar(255)	Yes	The last name of the user who deployed the contract
ContractDeployedByUserEmailAddress	nvarchar(255)	Yes	The email address of the user who deployed the contract

vwContractAction

This view represents the majority of information related to actions taken on contracts and is designed to readily facilitate common reporting scenarios. For each action taken, this view contains the following data:

- Associated application definition
- Associated workflow definition
- Associated smart contract function and parameter definition
- Associated ledger implementation for the function
- Specific instance values provided for parameters
- Details for the user who initiated the action
- Details related to the blockchain block and transaction

NAME	TYPE	CAN BE NULL	DESCRIPTION
ApplicationId	int	No	A unique identifier for the application
ApplicationName	nvarchar(50)	No	The name of the application
ApplicationDisplayName	nvarchar(255)	No	The name to be displayed in a user interface
ApplicationEnabled	bit	No	This field identifies if the application is currently enabled. Note – Even though an application can be reflected as disabled in the database, associated contracts remain on the blockchain and data about those contracts remain in the database.
WorkflowId	int	No	A unique identifier for a workflow
WorkflowName	nvarchar(50)	No	The name of the workflow
WorkflowDisplayName	nvarchar(255)	No	The name of the workflow to display in a user interface
WorkflowDescription	nvarchar(255)	Yes	The description of the workflow

NAME	TYPE	CAN BE NULL	DESCRIPTION
ContractId	int	No	A unique identifier for the contract
ContractProvisioningStatus	int	No	Identifies the current status of the provisioning process for the contract. Possible values are: 0 – The contract has been created by the API in the database 1 – The contract has been sent to the ledger 2 – The contract has been successfully deployed to the ledger 3 or 4 - The contract failed to be deployed to the ledger 5 - The contract was successfully deployed to the ledger Beginning with version 1.5, values 0 through 5 are supported. For backwards compatibility in the current release, view **vwContractActionV0** is available that only supports values 0 through 2.
ContractCodeId	int	No	A unique identifier for the code implementation of the contract
ContractLedgerIdentifier	nvarchar(255)	Yes	A unique identifier associated with the deployed version of a smart contract for a specific distributed ledger. For example, Ethereum.
ContractDeployedByUserId	int	No	The unique identifier of the user that deployed the contract
ContractDeployedByUserFirstName	nvarchar(50)	Yes	First name of the user who deployed the contract
ContractDeployedByUserLastName	nvarchar(255)	Yes	Last name of the user who deployed the contract
ContractDeployedByUserExternalId	nvarchar(255)	No	External identifier of the user who deployed the contract. By default, this ID is the guid that represents their identity in the consortium Azure Active Directory.

NAME	TYPE	CAN BE NULL	DESCRIPTION
ContractDeployedByUserEmailAddress	nvarchar(255)	Yes	The email address of the user who deployed the contract
WorkflowFunctionId	int	No	A unique identifier for a workflow function
WorkflowFunctionName	nvarchar(50)	No	The name of the function
WorkflowFunctionDisplayName	nvarchar(255)	No	The name of a function to be displayed in the user interface
WorkflowFunctionDescription	nvarchar(255)	No	The description of the function
ContractActionId	int	No	The unique identifier for a contract action
ContractActionProvisioningStatus	int	No	Identifies the current status of the provisioning process for the contract action. Possible values are: 0 – The contract action has been created by the API in the database 1 – The contract action has been sent to the ledger 2 – The contract action has been successfully deployed to the ledger 3 or 4 - The contract failed to be deployed to the ledger 5 - The contract was successfully deployed to the ledger Beginning with version 1.5, values 0 through 5 are supported. For backwards compatibility in the current release, view **vwContractActionV0** is available that only supports values 0 through 2.
ContractActionTimestamp	datetime(2,7)	No	The timestamp of the contract action
ContractActionExecutedByUserId	int	No	Unique identifier of the user that executed the contract action
ContractActionExecutedByUserFirstName	int	Yes	First name of the user who executed the contract action

NAME	TYPE	CAN BE NULL	DESCRIPTION
ContractActionExecutedByUserLastName	nvarchar(50)	Yes	Last name of the user who executed the contract action
ContractActionExecutedByUserExternalId	nvarchar(255)	Yes	External identifier of the user who executed the contract action. By default, this ID is the guid that represents their identity in the consortium Azure Active Directory.
ContractActionExecutedByUserEmailAddress	nvarchar(255)	Yes	The email address of the user who executed the contract action
WorkflowFunctionParameterId	int	No	A unique identifier for a parameter of the function
WorkflowFunctionParameterName	nvarchar(50)	No	The name of a parameter of the function
WorkflowFunctionParameterDisplayName	nvarchar(255)	No	The name of a function parameter to be displayed in the user interface
WorkflowFunctionParameterDataTypeId	int	No	The unique identifier for the data type associated with a workflow function parameter
WorkflowParameterDataTypeName	nvarchar(50)	No	The name of a data type associated with a workflow function parameter
ContractActionParameterValue	nvarchar(255)	No	The value for the parameter stored in the smart contract
BlockHash	nvarchar(255)	Yes	The hash of the block
BlockNumber	int	Yes	The number of the block on the ledger
BlockTimestamp	datetime(2,7)	Yes	The time stamp of the block
TransactionId	int	No	A unique identifier for the transaction
TransactionFrom	nvarchar(255)	Yes	The party that originated the transaction
TransactionTo	nvarchar(255)	Yes	The party that was transacted with
TransactionHash	nvarchar(255)	Yes	The hash of a transaction

NAME	TYPE	CAN BE NULL	DESCRIPTION
TransactionIsWorkbenchTransaction	bit	Yes	A bit that identifies if the transaction is an Azure Blockchain Workbench transaction
TransactionProvisioningStatus	int	Yes	Identifies the current status of the provisioning process for the transaction. Possible values are: 0 – The transaction has been created by the API in the database 1 – The transaction has been sent to the ledger 2 – The transaction has been successfully deployed to the ledger
TransactionValue	decimal(32,2)	Yes	The value of the transaction

vwContractProperty

This view represents the majority of information related to properties associated with a contract and is designed to readily facilitate common reporting scenarios. For each property taken, this view contains the following data:

- Associated application definition
- Associated workflow definition
- Details for the user who deployed the workflow
- Associated smart contract property definition
- Specific instance values for properties
- Details for the state property of the contract

NAME	TYPE	CAN BE NULL	DESCRIPTION
ApplicationId	int	No	A unique identifier for the application
ApplicationName	nvarchar(50)	No	The name of the application
ApplicationDisplayName	nvarchar(255)	No	The name to be displayed in a user interface
ApplicationEnabled	bit	No	Identifies if the application is currently enabled. **Note:** Even though an application can be reflected as disabled in the database, associated contracts remain on the blockchain and data about those contracts remain in the database.
WorkflowId	int	No	The unique identifier for the workflow

NAME	TYPE	CAN BE NULL	DESCRIPTION
WorkflowName	nvarchar(50)	No	The name of the workflow
WorkflowDisplayName	nvarchar(255)	No	The name of the workflow displayed in the user interface
WorkflowDescription	nvarchar(255)	Yes	The description of the workflow
ContractId	int	No	The unique identifier for the contract
ContractProvisioningStatus	int	No	Identifies the current status of the provisioning process for the contract. Possible values are: 0 – The contract has been created by the API in the database 1 – The contract has been sent to the ledger 2 – The contract has been successfully deployed to the ledger 3 or 4 - The contract failed to be deployed to the ledger 5 - The contract was successfully deployed to the ledger Beginning with version 1.5, values 0 through 5 are supported. For backwards compatibility in the current release, view **vwContractPropertyV0** is available that only supports values 0 through 2.
ContractCodeId	int	No	A unique identifier for the code implementation of the contract
ContractLedgerIdentifier	nvarchar(255)	Yes	A unique identifier associated with the deployed version of a smart contract for a specific distributed ledger. For example, Ethereum.
ContractDeployedByUserId	int	No	The unique identifier of the user that deployed the contract
ContractDeployedByUserFirstName	nvarchar(50)	Yes	First name of the user who deployed the contract

NAME	TYPE	CAN BE NULL	DESCRIPTION
ContractDeployedByUserLastName	nvarchar(255)	Yes	Last name of the user who deployed the contract
ContractDeployedByUserExternalId	nvarchar(255)	No	External identifier of the user who deployed the contract. By default, this ID is the guid that represents their identity in the consortium Azure Active Directory
ContractDeployedByUserEmailAddress	nvarchar(255)	Yes	The email address of the user who deployed the contract
WorkflowPropertyId	int		A unique identifier for a property of a workflow
WorkflowPropertyDataTypeId	int	No	The ID of the data type of the property
WorkflowPropertyDataTypeName	nvarchar(50)	No	The name of the data type of the property
WorkflowPropertyName	nvarchar(50)	No	The name of the workflow property
WorkflowPropertyDisplayName	nvarchar(255)	No	The display name of the workflow property
WorkflowPropertyDescription	nvarchar(255)	Yes	A description of the property
ContractPropertyValue	nvarchar(255)	No	The value for a property on the contract
StateName	nvarchar(50)	Yes	If this property contains the state of the contract, it is the display name for the state. If it is not associated with the state, the value will be null.
StateDisplayName	nvarchar(255)	No	If this property contains the state, it is the display name for the state. If it is not associated with the state, the value will be null.
StateValue	nvarchar(255)	Yes	If this property contains the state, it is the state value. If it is not associated with the state, the value will be null.

vwContractState

This view represents the majority of information related to the state of a specific contract and is designed to

readily facilitate common reporting scenarios. Each record in this view contains the following data:

- Associated application definition
- Associated workflow definition
- Details for the user who deployed the workflow
- Associated smart contract property definition
- Details for the state property of the contract

NAME	TYPE	CAN BE NULL	DESCRIPTION
ApplicationId	int	No	A unique identifier for the application
ApplicationName	nvarchar(50)	No	The name of the application
ApplicationDisplayName	nvarchar(255)	No	The name to be displayed in a user interface
ApplicationEnabled	bit	No	Identifies if the application is currently enabled. **Note:** Even though an application can be reflected as disabled in the database, associated contracts remain on the blockchain and data about those contracts remain in the database.
WorkflowId	int	No	A unique identifier for the workflow
WorkflowName	nvarchar(50)	No	The name of the workflow
WorkflowDisplayName	nvarchar(255)	No	The name displayed in the user interface
WorkflowDescription	nvarchar(255)	Yes	The description of the workflow
ContractLedgerImplementationId	nvarchar(255)	Yes	A unique identifier associated with the deployed version of a smart contract for a specific distributed ledger. For example, Ethereum.
ContractId	int	No	A unique identifier for the contract

NAME	TYPE	CAN BE NULL	DESCRIPTION
ContractProvisioningStatus	int	No	Identifies the current status of the provisioning process for the contract. Possible values are: 0 – The contract has been created by the API in the database 1 – The contract has been sent to the ledger 2 – The contract has been successfully deployed to the ledger 3 or 4 - The contract failed to be deployed to the ledger 5 - The contract was successfully deployed to the ledger Beginning with version 1.5, values 0 through 5 are supported. For backwards compatibility in the current release, view **vwContractStateV0** is available that only supports values 0 through 2.
ConnectionId	int	No	A unique identifier for the blockchain instance the workflow is deployed to
ContractCodeId	int	No	A unique identifier for the code implementation of the contract
ContractDeployedByUserId	int	No	Unique identifier of the user that deployed the contract
ContractDeployedByUserExternalId	nvarchar(255)	No	External identifier of the user who deployed the contract. By default, this ID is the guid that represents their identity in the consortium Azure Active Directory.
ContractDeployedByUserFirstName	nvarchar(50)	Yes	First name of the user who deployed the contract
ContractDeployedByUserLastName	nvarchar(255)	Yes	Last name of the user who deployed the contract
ContractDeployedByUserEmailAddress	nvarchar(255)	Yes	The email address of the user who deployed the contract
WorkflowPropertyId	int	No	A unique identifier for a workflow property

NAME	TYPE	CAN BE NULL	DESCRIPTION
WorkflowPropertyDataTypeId	int	No	The ID of the data type of the workflow property
WorkflowPropertyDataTypeName	nvarchar(50)	No	The name of the data type of the workflow property
WorkflowPropertyName	nvarchar(50)	No	The name of the workflow property
WorkflowPropertyDisplayName	nvarchar(255)	No	The display name of the property to show in a UI
WorkflowPropertyDescription	nvarchar(255)	Yes	The description of the property
ContractPropertyValue	nvarchar(255)	No	The value for a property stored in the contract
StateName	nvarchar(50)	Yes	If this property contains the state, it the display name for the state. If it is not associated with the state, the value will be null.
StateDisplayName	nvarchar(255)	No	If this property contains the state, it is the display name for the state. If it is not associated with the state, the value will be null.
StateValue	nvarchar(255)	Yes	If this property contains the state, it is the state value. If it is not associated with the state, the value will be null.

vwUser

This view provides details on the consortium members that are provisioned to use Azure Blockchain Workbench. By default, data is populated through the initial provisioning of the user.

NAME	TYPE	CAN BE NULL	DESCRIPTION
ID	int	No	A unique identifier for a user
ExternalID	nvarchar(255)	No	An external identifier for a user. By default, this ID is the guid representing the Azure Active Directory ID for the user.

NAME	TYPE	CAN BE NULL	DESCRIPTION
ProvisioningStatus	int	No	Identifies the current status of provisioning process for the user. Possible values are: 0 – User has been created by the API 1 – A key has been associated with the user in the database 2 – The user is fully provisioned
FirstName	nvarchar(50)	Yes	The first name of the user
LastName	nvarchar(50)	Yes	The last name of the user
EmailAddress	nvarchar(255)	Yes	The email address of the user

vwWorkflow

This view represents the details core workflow metadata as well as the workflow's functions and parameters. Designed for reporting, it also contains metadata about the application associated with the workflow. This view contains data from multiple underlying tables to facilitate reporting on workflows. For each workflow, this view contains the following data:

- Associated application definition
- Associated workflow definition
- Associated workflow start state information

NAME	TYPE	CAN BE NULL	DESCRIPTION
ApplicationId	int	No	A unique identifier for the application
ApplicationName	nvarchar(50)	No	The name of the application
ApplicationDisplayName	nvarchar(255)	No	The name to be displayed in a user interface
ApplicationEnabled	bit	No	Identifies if the application is enabled
WorkflowId	int	Yes	A unique identifier for a workflow
WorkflowName	nvarchar(50)	No	The name of the workflow
WorkflowDisplayName	nvarchar(255)	No	The name displayed in the user interface
WorkflowDescription	nvarchar(255)	Yes	The description of the workflow.

NAME	TYPE	CAN BE NULL	DESCRIPTION
WorkflowConstructorFunctionId	int	No	The identifier of the workflow function that serves as the constructor for the workflow
WorkflowStartStateId	int	No	A unique identifier for the state
WorkflowStartStateName	nvarchar(50)	No	The name of the state
WorkflowStartStateDisplayName	nvarchar(255)	No	The name to be displayed in the user interface for the state
WorkflowStartStateDescription	nvarchar(255)	Yes	A description of the workflow state
WorkflowStartStateStyle	nvarchar(50)	Yes	This value identifies the percentage complete that the workflow is when in this state
WorkflowStartStateValue	int	No	The value of the state
WorkflowStartStatePercentComplete	int	No	A text description that provides a hint to clients on how to render this state in the UI. Supported states include *Success* and *Failure*

vwWorkflowFunction

This view represents the details core workflow metadata as well as the workflow's functions and parameters. Designed for reporting, it also contains metadata about the application associated with the workflow. This view contains data from multiple underlying tables to facilitate reporting on workflows. For each workflow function, this view contains the following data:

- Associated application definition
- Associated workflow definition
- Workflow function details

NAME	TYPE	CAN BE NULL	DESCRIPTION
ApplicationId	int	No	A unique identifier for the application
ApplicationName	nvarchar(50)	No	The name of the application
ApplicationDisplayName	nvarchar(255)	No	The name to be displayed in a user interface
ApplicationEnabled	bit	No	Identifies if the application is enabled

NAME	TYPE	CAN BE NULL	DESCRIPTION
WorkflowId	int	No	A unique identifier for a workflow
WorkflowName	nvarchar(50)	No	The name of the workflow
WorkflowDisplayName	nvarchar(255)	No	The name of the workflow displayed in the user interface
WorkflowDescription	nvarchar(255)	Yes	The description of the workflow
WorkflowFunctionId	int	No	A unique identifier for a function
WorkflowFunctionName	nvarchar(50)	Yes	The name of the function
WorkflowFunctionDisplayName	nvarchar(255)	No	The name of a function to be displayed in the user interface
WorkflowFunctionDescription	nvarchar(255)	Yes	The description of the workflow function
WorkflowFunctionIsConstructor	bit	No	Identifies if the workflow function is the constructor for the workflow
WorkflowFunctionParameterId	int	No	A unique identifier for a parameter of a function
WorkflowFunctionParameterName	nvarchar(50)	No	The name of a parameter of the function
WorkflowFunctionParameterDisplayName	nvarchar(255)	No	The name of a function parameter to be displayed in the user interface
WorkflowFunctionParameterDataTypeId	int	No	A unique identifier for the data type associated with a workflow function parameter
WorkflowParameterDataTypeName	nvarchar(50)	No	The name of a data type associated with a workflow function parameter

vwWorkflowProperty

This view represents the properties defined for a workflow. For each property, this view contains the following data:

- Associated application definition
- Associated workflow definition
- Workflow property details

NAME	TYPE	CAN BE NULL	DESCRIPTION
ApplicationId	int	No	A unique identifier for the application
ApplicationName	nvarchar(50)	No	The name of the application
ApplicationDisplayName	nvarchar(255)	No	The name to be displayed in a user interface
ApplicationEnabled	bit	No	Identifies if the application is currently enabled. **Note:** Even though an application can be reflected as disabled in the database, associated contracts remain on the blockchain and data about those contracts remain in the database.
WorkflowId	int	No	A unique identifier for the workflow
WorkflowName	nvarchar(50)	No	The name of the workflow
WorkflowDisplayName	nvarchar(255)	No	The name to be displayed for the workflow in a user interface
WorkflowDescription	nvarchar(255)	Yes	A description of the workflow
WorkflowPropertyID	int	No	A unique identifier for a property of a workflow
WorkflowPropertyName	nvarchar(50)	No	The name of the property
WorkflowPropertyDescription	nvarchar(255)	Yes	A description of the property
WorkflowPropertyDisplayName	nvarchar(255)	No	The name to be displayed in a user interface
WorkflowPropertyWorkflowId	int	No	The ID of the workflow to which this property is associated
WorkflowPropertyDataTypeId	int	No	The ID of the data type defined for the property
WorkflowPropertyDataTypeName	nvarchar(50)	No	The name of the data type defined for the property
WorkflowPropertyIsState	bit	No	This field identifies if this workflow property contains the state of the workflow

vwWorkflowState

This view represents the properties associated with a workflow. For each contract, this view contains the following data:

- Associated application definition
- Associated workflow definition
- Workflow state information

NAME	TYPE	CAN BE NULL	DESCRIPTION
ApplicationId	int	No	A unique identifier for the application
ApplicationName	nvarchar(50)	No	The name of the application
ApplicationDisplayName	nvarchar(255)	No	A description of the application
ApplicationEnabled	bit	No	Identifies if the application is currently enabled. **Note:** Even though an application can be reflected as disabled in the database, associated contracts remain on the blockchain and data about those contracts remain in the database.
WorkflowId	int	No	The unique identifier for the workflow
WorkflowName	nvarchar(50)	No	The name of the workflow
WorkflowDisplayName	nvarchar(255)	No	The name displayed in the user interface for the workflow
WorkflowDescription	nvarchar(255)	Yes	The description of the workflow
WorkflowStateID	int	No	The unique identifier for the state
WorkflowStateName	nvarchar(50)	No	The name of the state
WorkflowStateDisplayName	nvarchar(255)	No	The name to be displayed in the user interface for the state
WorkflowStateDescription	nvarchar(255)	Yes	A description of the workflow state
WorkflowStatePercentComplete	int	No	This value identifies the percentage complete that the workflow is when in this state

NAME	TYPE	CAN BE NULL	DESCRIPTION
WorkflowStateValue	nvarchar(50)	No	Value of the state
WorkflowStateStyle	nvarchar(50)	No	A text description that provides a hint to clients on how to render this state in the UI. Supported states include *Success* and *Failure*

Azure Blockchain Workbench messaging integration

6/16/2019 • 14 minutes to read • Edit Online

In addition to providing a REST API, Azure Blockchain Workbench also provides messaging-based integration. Workbench publishes ledger-centric events via Azure Event Grid, enabling downstream consumers to ingest data or take action based on these events. For those clients that require reliable messaging, Azure Blockchain Workbench delivers messages to an Azure Service Bus endpoint as well.

Input APIs

If you want to initiate transactions from external systems to create users, create contracts, and update contracts, you can use messaging input APIs to perform transactions on a ledger. See messaging integration samples for a sample that demonstrates input APIs.

The following are the currently available input APIs.

Create user

Creates a new user.

The request requires the following fields:

NAME	DESCRIPTION
requestId	Client supplied GUID
firstName	First name of the user
lastName	Last name of the user
emailAddress	Email address of the user
externalId	Azure AD object ID of the user
connectionId	Unique identifier for the blockchain connection
messageSchemaVersion	Messaging schema version
messageName	**CreateUserRequest**

Example:

```
{
    "requestId": "e2264523-6147-41fc-bbbb-edba8e44562d",
    "firstName": "Ali",
    "lastName": "Alio",
    "emailAddress": "aa@contoso.com",
    "externalId": "6a9b7f65-ffff-442f-b3b8-58a35abd1bcd",
    "connectionId": 1,
    "messageSchemaVersion": "1.0.0",
    "messageName": "CreateUserRequest"
}
```

Blockchain Workbench returns a response with the following fields:

NAME	DESCRIPTION
requestId	Client supplied GUID
userId	ID of the user that was created
userChainIdentifier	Address of the user that was created on the blockchain network. In Ethereum, the address is the user's **on-chain** address.
connectionId	Unique identifier for the blockchain connection
messageSchemaVersion	Messaging schema version
messageName	**CreateUserUpdate**
status	Status of the user creation request. If successful, value is **Success**. On failure, value is **Failure**.
additionalInformation	Additional information provided based on the status

Example successful **create user** response from Blockchain Workbench:

```
{
    "requestId": "e2264523-6147-41fc-bb59-edba8e44562d",
    "userId": 15,
    "userChainIdentifier": "0x9a8DDaCa9B7488683A4d62d0817E965E8f248398",
    "connectionId": 1,
    "messageSchemaVersion": "1.0.0",
    "messageName": "CreateUserUpdate",
    "status": "Success",
    "additionalInformation": { }
}
```

If the request was unsuccessful, details about the failure are included in additional information.

```
{
    "requestId": "e2264523-6147-41fc-bb59-edba8e44562d",
    "userId": 15,
    "userChainIdentifier": null,
    "connectionId": 1,
    "messageSchemaVersion": "1.0.0",
    "messageName": "CreateUserUpdate",
    "status": "Failure",
    "additionalInformation": {
        "errorCode": 4000,
        "errorMessage": "User cannot be provisioned on connection."
    }
}
```

Create contract

Creates a new contract.

The request requires the following fields:

NAME	DESCRIPTION
requestId	Client supplied GUID
userChainIdentifier	Address of the user that was created on the blockchain network. In Ethereum, this address is the user's **on chain** address.
applicationName	Name of the application
version	Version of the application. Required if you have multiple versions of the application enabled. Otherwise, version is optional. For more information on application versioning, see Azure Blockchain Workbench application versioning.
workflowName	Name of the workflow
parameters	Parameters input for contract creation
connectionId	Unique identifier for the blockchain connection
messageSchemaVersion	Messaging schema version
messageName	**CreateContractRequest**

Example:

```
{
    "requestId": "ce3c429b-a091-4baa-b29b-5b576162b211",
    "userChainIdentifier": "0x9a8DDaCa9B7488683A4d62d0817E965E8f248398",
    "applicationName": "AssetTransfer",
    "version": "1.0",
    "workflowName": "AssetTransfer",
    "parameters": [
        {
            "name": "description",
            "value": "a 1969 dodge charger"
        },
        {
            "name": "price",
            "value": "12345"
        }
    ],
    "connectionId": 1,
    "messageSchemaVersion": "1.0.0",
    "messageName": "CreateContractRequest"
}
```

Blockchain Workbench returns a response with the following fields:

NAME	DESCRIPTION
requestId	Client supplied GUID
contractId	Unique identifier for the contract inside Azure Blockchain Workbench

NAME	DESCRIPTION
contractLedgerIdentifier	Address of the contract on the ledger
connectionId	Unique identifier for the blockchain connection
messageSchemaVersion	Messaging schema version
messageName	**CreateContractUpdate**
status	Status of the contract creation request. Possible values: **Submitted**, **Committed**, **Failure**.
additionalInformation	Additional information provided based on the status

Example of a submitted **create contract** response from Blockchain Workbench:

```
{
    "requestId": "ce3c429b-a091-4baa-b29b-5b576162b211",
    "contractId": 55,
    "contractLedgerIdentifier": "0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe",
    "connectionId": 1,
    "messageSchemaVersion": "1.0.0",
    "messageName": "CreateContractUpdate",
    "status": "Submitted",
    "additionalInformation": { }
}
```

Example of a committed **create contract** response from Blockchain Workbench:

```
{
    "requestId": "ce3c429b-a091-4baa-b29b-5b576162b211",
    "contractId": 55,
    "contractLedgerIdentifier": "0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe",
    "connectionId": 1,
    "messageSchemaVersion": "1.0.0",
    "messageName": "CreateContractUpdate",
    "status": "Committed",
    "additionalInformation": { }
}
```

If the request was unsuccessful, details about the failure are included in additional information.

```
{
    "requestId": "ce3c429b-a091-4baa-b29b-5b576162b211",
    "contractId": 55,
    "contractLedgerIdentifier": null,
    "connectionId": 1,
    "messageSchemaVersion": "1.0.0",
    "messageName": "CreateContractUpdate",
    "status": "Failure",
    "additionalInformation": {
        "errorCode": 4000,
        "errorMessage": "Contract cannot be provisioned on connection."
    }
}
```

Create contract action

Creates a new contract action.

The request requires the following fields:

NAME	DESCRIPTION
requestId	Client supplied GUID
userChainIdentifier	Address of the user that was created on the blockchain network. In Ethereum, this address is the user's **on chain** address.
contractLedgerIdentifier	Address of the contract on the ledger
version	Version of the application. Required if you have multiple versions of the application enabled. Otherwise, version is optional. For more information on application versioning, see Azure Blockchain Workbench application versioning.
workflowFunctionName	Name of the workflow function
parameters	Parameters input for contract creation
connectionId	Unique identifier for the blockchain connection
messageSchemaVersion	Messaging schema version
messageName	**CreateContractActionRequest**

Example:

```
{
    "requestId": "a5530932-9d6b-4eed-8623-441a647741d3",
    "userChainIdentifier": "0x9a8DDaCa9B7488683A4d62d0817E965E8f248398",
    "contractLedgerIdentifier": "0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe",
    "version": "1.0",
    "workflowFunctionName": "modify",
    "parameters": [
        {
            "name": "description",
            "value": "a 1969 dodge charger"
        },
        {
            "name": "price",
            "value": "12345"
        }
    ],
    "connectionId": 1,
    "messageSchemaVersion": "1.0.0",
    "messageName": "CreateContractActionRequest"
}
```

Blockchain Workbench returns a response with the following fields:

NAME	DESCRIPTION
requestId	Client supplied GUID

NAME	DESCRIPTION
contractId	Unique identifier for the contract inside Azure Blockchain Workbench
connectionId	Unique identifier for the blockchain connection
messageSchemaVersion	Messaging schema version
messageName	**CreateContractActionUpdate**
status	Status of the contract action request. Possible values: **Submitted**, **Committed**, **Failure**.
additionalInformation	Additional information provided based on the status

Example of a submitted **create contract action** response from Blockchain Workbench:

```
{
    "requestId": "a5530932-9d6b-4eed-8623-441a647741d3",
    "contractId": 105,
    "connectionId": 1,
    "messageSchemaVersion": "1.0.0",
    "messageName": "CreateContractActionUpdate",
    "status": "Submitted",
    "additionalInformation": { }
}
```

Example of a committed **create contract action** response from Blockchain Workbench:

```
{
    "requestId": "a5530932-9d6b-4eed-8623-441a647741d3",
    "contractId": 105,
    "connectionId": 1,
    "messageSchemaVersion": "1.0.0",
    "messageName": "CreateContractActionUpdate",
    "status": "Committed",
    "additionalInformation": { }
}
```

If the request was unsuccessful, details about the failure are included in additional information.

```
{
    "requestId": "a5530932-9d6b-4eed-8623-441a647741d3",
    "contractId": 105,
    "connectionId": 1,
    "messageSchemaVersion": "1.0.0",
    "messageName": "CreateContractActionUpdate",
    "status": "Failure",
    "additionalInformation": {
        "errorCode": 4000,
        "errorMessage": "Contract action cannot be provisioned on connection."
    }
}
```

Input API error codes and messages

Error code 4000: Bad request error

- Invalid connectionId
- CreateUserRequest deserialization failed
- CreateContractRequest deserialization failed
- CreateContractActionRequest deserialization failed
- Application {identified by application name} does not exist
- Application {identified by application name} does not have workflow
- UserChainIdentifier does not exist
- Contract {identified by ledger identifier} does not exist
- Contract {identified by ledger identifier} does not have function {workflow function name}
- UserChainIdentifier does not exist

Error code 4090: Conflict error

- User already exists
- Contract already exists
- Contract action already exists

Error code 5000: Internal server error

- Exception messages

Event notifications

Event notifications can be used to notify users and downstream systems of events that happen in Blockchain Workbench and the blockchain network it is connected to. Event notifications can be consumed directly in code or used with tools such as Logic Apps and Flow to trigger flow of data to downstream systems.

See Notification message reference for details of various messages that can be received.

Consuming Event Grid events with Azure Functions

If a user wants to use Event Grid to be notified about events that happen in Blockchain Workbench, you can consume events from Event Grid by using Azure Functions.

1. Create an **Azure Function App** in the Azure portal.
2. Create a new function.
3. Locate the template for Event Grid. Basic template code for reading the message is shown. Modify the code as needed.
4. Save the Function.
5. Select the Event Grid from Blockchain Workbench's resource group.

Consuming Event Grid events with Logic Apps

1. Create a new **Azure Logic App** in the Azure portal.
2. When opening the Azure Logic App in the portal, you will be prompted to select a trigger. Select **Azure Event Grid -- When a resource event occurs**.
3. When the workflow designer is displayed, you will be prompted to sign in.
4. Select the Subscription. Resource as **Microsoft.EventGrid.Topics**. Select the **Resource Name** from the name of the resource from the Azure Blockchain Workbench resource group.
5. Select the Event Grid from Blockchain Workbench's resource group.

Using Service Bus Topics for notifications

Service Bus Topics can be used to notify users about events that happen in Blockchain Workbench.

1. Browse to the Service Bus within the Workbench's resource group.
2. Select **Topics**.
3. Select **egress-topic**.
4. Create a new subscription to this topic. Obtain a key for it.
5. Create a program, which subscribes to events from this subscription.

Consuming Service Bus Messages with Logic Apps

1. Create a new **Azure Logic App** in the Azure portal.
2. When opening the Azure Logic App in the portal, you will be prompted to select a trigger. Type **Service Bus** into the search box and select the trigger appropriate for the type of interaction you want to have with the Service Bus. For example, **Service Bus -- When a message is received in a topic subscription (auto-complete)**.
3. When the workflow designer is displayed, specify the connection information for the Service Bus.
4. Select your subscription and specify the topic of **workbench-external**.
5. Develop the logic for your application that utilizes the message from this trigger.

Notification message reference

Depending on the **messageName**, the notification messages have one of the following message types.

Block message

Contains information about individual blocks. The *BlockMessage* includes a section with block level information and a section with transaction information.

NAME	DESCRIPTION
block	Contains block information
transactions	Contains a collection transaction information for the block
connectionId	Unique identifier for the connection
messageSchemaVersion	Messaging schema version
messageName	**BlockMessage**
additionalInformation	Additional information provided

Block information

NAME	DESCRIPTION
blockId	Unique identifier for the block inside Azure Blockchain Workbench
blockNumber	Unique identifier for a block on the ledger
blockHash	The hash of the block
previousBlockHash	The hash of the previous block
blockTimestamp	The timestamp of the block

Transaction information

NAME	DESCRIPTION
transactionId	Unique identifier for the transaction inside Azure Blockchain Workbench
transactionHash	The hash of the transaction on the ledger
from	Unique identifier on the ledger for the transaction origin
to	Unique identifier on the ledger for the transaction destination
provisioningStatus	Identifies the current status of the provisioning process for the transaction. Possible values are: 0 – The transaction has been created by the API in the database 1 – The transaction has been sent to the ledger 2 – The transaction has been successfully committed to the ledger 3 or 4 - The transaction failed to be committed to the ledger 5 - The transaction was successfully committed to the ledger

Example of a *BlockMessage* from Blockchain Workbench:


```
{
    "block": {
        "blockId": 123,
        "blockNumber": 1738312,
        "blockHash": "0x03a39411e25e25b47d0ec6433b73b488554a4a5f6b1a253e0ac8a200d13fffff",
        "previousBlockHash": null,
        "blockTimestamp": "2018-10-09T23:35:58Z",
    },
    "transactions": [
        {
            "transactionId": 234,
            "transactionHash": "0xa4d9c95b581f299e41b8cc193dd742ef5a1d3a4ddf97bd11b80d123fec27ffff",
            "from": "0xd85e7262dd96f3b8a48a8aaf3dcdda90f60dffff",
            "to": null,
            "provisioningStatus": 1
        },
        {
            "transactionId": 235,
            "transactionHash": "0x5c1fddea83bf19d719e52a935ec8620437a0a6bdaa00ecb7c3d852cf92e1ffff",
            "from": "0xadd97e1e595916e29ea94fda894941574000ffff",
            "to": "0x9a8DDaCa9B7488683A4d62d0817E965E8f24ffff",
            "provisioningStatus": 2
        }
    ],
    "connectionId": 1,
    "messageSchemaVersion": "1.0.0",
    "messageName": "BlockMessage",
    "additionalInformation": {}
}
```


Contract message

Contains information about a contract. The message includes a section with contract properties and a section with transaction information. All transactions that have modified the contract for the particular block are included in the transaction section.

NAME	DESCRIPTION
blockId	Unique identifier for the block inside Azure Blockchain Workbench
blockHash	Hash of the block
modifyingTransactions	Transactions that modified the contract
contractId	Unique identifier for the contract inside Azure Blockchain Workbench
contractLedgerIdentifier	Unique identifier for the contract on the ledger
contractProperties	Properties of the contract
isNewContract	Indicates whether or not this contract was newly created. Possible values are: true: this contract was a new contract created. false: this contract is a contract update.
connectionId	Unique identifier for the connection
messageSchemaVersion	Messaging schema version
messageName	**ContractMessage**
additionalInformation	Additional information provided

Modifying transaction information

NAME	DESCRIPTION
transactionId	Unique identifier for the transaction inside Azure Blockchain Workbench
transactionHash	The hash of the transaction on the ledger
from	Unique identifier on the ledger for the transaction origin
to	Unique identifier on the ledger for the transaction destination

Contract properties

NAME	DESCRIPTION
workflowPropertyId	Unique identifier for the workflow property inside Azure Blockchain Workbench
name	Name of the workflow property
value	Value of the workflow property

Example of a *ContractMessage* from Blockchain Workbench:

```
{
```

```
"blockId": 123,
"blockhash": "0x03a39411e25e25b47d0ec6433b73b488554a4a5f6b1a253e0ac8a200d13fffff",
"modifyingTransactions": [
    {
        "transactionId": 234,
        "transactionHash": "0x5c1fddea83bf19d719e52a935ec8620437a0a6bdaa00ecb7c3d852cf92e1ffff",
        "from": "0xd85e7262dd96f3b8a48a8aaf3dcdda90f60dffff",
        "to": "0xf8559473b3c7197d59212b401f5a9f07ffff"
    },
    {
        "transactionId": 235,
        "transactionHash": "0xa4d9c95b581f299e41b8cc193dd742ef5a1d3a4ddf97bd11b80d123fec27ffff",
        "from": "0xd85e7262dd96f3b8a48a8aaf3dcdda90f60dffff",
        "to": "0xf8559473b3c7197d59212b401f5a9f07b429ffff"
    }
],
"contractId": 111,
"contractLedgerIdentifier": "0xf8559473b3c7197d59212b401f5a9f07b429ffff",
"contractProperties": [
    {
        "workflowPropertyId": 1,
        "name": "State",
        "value": "0"
    },
    {
        "workflowPropertyId": 2,
        "name": "Description",
        "value": "1969 Dodge Charger"
    },
    {
        "workflowPropertyId": 3,
        "name": "AskingPrice",
        "value": "30000"
    },
    {
        "workflowPropertyId": 4,
        "name": "OfferPrice",
        "value": "0"
    },
    {
        "workflowPropertyId": 5,
        "name": "InstanceAppraiser",
        "value": "0x0000000000000000000000000000000000000000"
    },
    {
        "workflowPropertyId": 6,
        "name": "InstanceBuyer",
        "value": "0x0000000000000000000000000000000000000000"
    },
    {
        "workflowPropertyId": 7,
        "name": "InstanceInspector",
        "value": "0x0000000000000000000000000000000000000000"
    },
    {
        "workflowPropertyId": 8,
        "name": "InstanceOwner",
        "value": "0x9a8DDaCa9B7488683A4d62d0817E965E8f24ffff"
    },
    {
        "workflowPropertyId": 9,
        "name": "ClosingDayOptions",
        "value": "[21,48,69]"
    }
],
"isNewContract": false,
"connectionId": 1,
"messageSchemaVersion": "1.0.0",
"messageName": "ContractMessage",
```

```
    "additionalInformation": {}
}
```

Event message: Contract function invocation

Contains information when a contract function is invoked, such as the function name, parameters input, and the caller of the function.

NAME	DESCRIPTION
eventName	**ContractFunctionInvocation**
caller	Caller information
contractId	Unique identifier for the contract inside Azure Blockchain Workbench
contractLedgerIdentifier	Unique identifier for the contract on the ledger
functionName	Name of the function
parameters	Parameter information
transaction	Transaction information
inTransactionSequenceNumber	The sequence number of the transaction in the block
connectionId	Unique identifier for the connection
messageSchemaVersion	Messaging schema version
messageName	**EventMessage**
additionalInformation	Additional information provided

Caller information

NAME	DESCRIPTION
type	Type of the caller, like a user or a contract
id	Unique identifier for the caller inside Azure Blockchain Workbench
ledgerIdentifier	Unique identifier for the caller on the ledger

Parameter information

NAME	DESCRIPTION
name	Parameter name
value	Parameter value

Event message transaction information

NAME	DESCRIPTION
transactionId	Unique identifier for the transaction inside Azure Blockchain Workbench
transactionHash	The hash of the transaction on the ledger
from	Unique identifier on the ledger for the transaction origin
to	Unique identifier on the ledger for the transaction destination

Example of an *EventMessage ContractFunctionInvocation* from Blockchain Workbench:

```
{
    "eventName": "ContractFunctionInvocation",
    "caller": {
        "type": "User",
        "id": 21,
        "ledgerIdentifier": "0xd85e7262dd96f3b8a48a8aaf3dcdda90f60ffff"
    },
    "contractId": 34,
    "contractLedgerIdentifier": "0xf8559473b3c7197d59212b401f5a9f07b429ffff",
    "functionName": "Modify",
    "parameters": [
        {
            "name": "description",
            "value": "a new description"
        },
        {
            "name": "price",
            "value": "4567"
        }
    ],
    "transaction": {
        "transactionId": 234,
        "transactionHash": "0x5c1fddea83bf19d719e52a935ec8620437a0a6bdaa00ecb7c3d852cf92e1ffff",
        "from": "0xd85e7262dd96f3b8a48a8aaf3dcdda90f60dffff",
        "to": "0xf8559473b3c7197d59212b401f5a9f07b429ffff"
    },
    "inTransactionSequenceNumber": 1,
    "connectionId": 1,
    "messageSchemaVersion": "1.0.0",
    "messageName": "EventMessage",
    "additionalInformation": { }
}
```

Event message: Application ingestion

Contains information when an application is uploaded to Workbench, such as the name and version of the application uploaded.

NAME	DESCRIPTION
eventName	**ApplicationIngestion**
applicationId	Unique identifier for the application inside Azure Blockchain Workbench
applicationName	Application name

NAME	DESCRIPTION
applicationDisplayName	Application display name
applicationVersion	Application version
applicationDefinitionLocation	URL where the application configuration file is located
contractCodes	Collection of contract codes for the application
applicationRoles	Collection of application roles for the application
applicationWorkflows	Collection of application workflows for the application
connectionId	Unique identifier for the connection
messageSchemaVersion	Messaging schema version
messageName	**EventMessage**
additionalInformation	Additional information provided here includes the application workflow states and transition information.

Contract code information

NAME	DESCRIPTION
id	Unique identifier for the contract code file inside Azure Blockchain Workbench
ledgerId	Unique identifier for the ledger inside Azure Blockchain Workbench
location	URL where the contract code file is located

Application role information

NAME	DESCRIPTION
id	Unique identifier for the application role inside Azure Blockchain Workbench
name	Name of the application role

Application workflow information

NAME	DESCRIPTION
id	Unique identifier for the application workflow inside Azure Blockchain Workbench
name	Application workflow name
displayName	Application workflow display name

NAME	DESCRIPTION
functions	Collection of functions for the application workflow
states	Collection of states for the application workflow
properties	Application workflow properties information

Workflow function information

NAME	DESCRIPTION
id	Unique identifier for the application workflow function inside Azure Blockchain Workbench
name	Function name
parameters	Parameters for the function

Workflow state information

NAME	DESCRIPTION
name	State name
displayName	State display name
style	State style (success or failure)

Workflow property information

NAME	DESCRIPTION
id	Unique identifier for the application workflow property inside Azure Blockchain Workbench
name	Property name
type	Property type

Example of an *EventMessage ApplicationIngestion* from Blockchain Workbench:

```
{
    "eventName": "ApplicationIngestion",
    "applicationId": 31,
    "applicationName": "AssetTransfer",
    "applicationDisplayName": "Asset Transfer",
    "applicationVersion": “1.0”,
    "applicationDefinitionLocation": "http://url",
    "contractCodes": [
        {
            "id": 23,
            "ledgerId": 1,
            "location": "http://url"
        }
    ],
    "applicationRoles": [
            {
                "id": 134,
```

```
                "name": "Buyer"
            },
            {
                "id": 135,
                "name": "Seller"
            }
        ],
    "applicationWorkflows": [
        {
            "id": 89,
            "name": "AssetTransfer",
            "displayName": "Asset Transfer",
            "functions": [
                {
                    "id": 912,
                    "name": "",
                    "parameters": [
                        {
                            "name": "description",
                            "type": {
                                "name": "string"
                             }
                        },
                        {
                            "name": "price",
                            "type": {
                                "name": "int"
                            }
                        }
                    ]
                },
                {
                    "id": 913,
                    "name": "modify",
                    "parameters": [
                        {
                            "name": "description",
                            "type": {
                                "name": "string"
                             }
                        },
                        {
                            "name": "price",
                            "type": {
                                "name": "int"
                            }
                        }
                    ]
                }
            ],
            "states": [
                 {
                      "name": "Created",
                      "displayName": "Created",
                      "style" : "Success"
                 },
                 {
                      "name": "Terminated",
                      "displayName": "Terminated",
                      "style" : "Failure"
                 }
            ],
            "properties": [
                {
                    "id": 879,
                    "name": "Description",
                    "type": {
                                "name": "string"
                     }
```

```
                },
                {
                    "id": 880,
                    "name": "Price",
                    "type": {
                                "name": "int"
                     }
                }
            ]
        }
    ],
    "connectionId": [ ],
    "messageSchemaVersion": "1.0.0",
    "messageName": "EventMessage",
    "additionalInformation":
        {
            "states" :
            [
                {
                    "Name": "BuyerAccepted",
                    "Transitions": [
                        {
                            "DisplayName": "Accept",
                            "AllowedRoles": [ ],
                            "AllowedInstanceRoles": [ "InstanceOwner" ],
                            "Function": "Accept",
                            "NextStates": [ "SellerAccepted" ]
                        }
                    ]
                }
            ]
        }
}
```

Event message: Role assignment

Contains information when a user is assigned a role in Workbench, such as who performed the role assignment and the name of the role and corresponding application.

NAME	DESCRIPTION
eventName	**RoleAssignment**
applicationId	Unique identifier for the application inside Azure Blockchain Workbench
applicationName	Application name
applicationDisplayName	Application display name
applicationVersion	Application version
applicationRole	Information about the application role
assigner	Information about the assigner
assignee	Information about the assignee
connectionId	Unique identifier for the connection

NAME	DESCRIPTION
messageSchemaVersion	Messaging schema version
messageName	**EventMessage**
additionalInformation	Additional information provided

RoleAssignment application role

NAME	DESCRIPTION
id	Unique identifier for the application role inside Azure Blockchain Workbench
name	Name of the application role

RoleAssignment assigner

NAME	DESCRIPTION
id	Unique identifier of the user inside Azure Blockchain Workbench
type	Type of the assigner
chainIdentifier	Unique identifier of the user on the ledger

RoleAssignment assignee

NAME	DESCRIPTION
id	Unique identifier of the user inside Azure Blockchain Workbench
type	Type of the assignee
chainIdentifier	Unique identifier of the user on the ledger

Example of an *EventMessage RoleAssignment* from Blockchain Workbench:

```
{
    "eventName": "RoleAssignment",
    "applicationId": 31,
    "applicationName": "AssetTransfer",
    "applicationDisplayName": "Asset Transfer",
    "applicationVersion": “1.0”,
    "applicationRole": {
        "id": 134,
        "name": "Buyer"
    },
    "assigner": {
        "id": 1,
        "type": null,
        "chainIdentifier": "0xeFFC7766d38aC862d79706c3C5CEEf089564ffff"
    },
    "assignee": {
        "id": 3,
        "type": null,
        "chainIdentifier": "0x9a8DDaCa9B7488683A4d62d0817E965E8f24ffff"
    },
    "connectionId": [ ],
    "messageSchemaVersion": "1.0.0",
    "messageName": "EventMessage",
    "additionalInformation": { }
}
```


Next steps

- Smart contract integration patterns

Ethereum proof-of-authority consortium

4/8/2019 • 29 minutes to read • Edit Online

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

Overview

This solution is designed to make it easier to deploy, configure, and govern a multi-member consortium Proof-of-authority Ethereum network with minimal Azure and Ethereum knowledge.

With a handful of user inputs and a single-click deployment through the Azure portal, each member can provision a network footprint, using Microsoft Azure Compute, networking, and storage services across the globe. Each member's network footprint consists of a set of load-balanced validator nodes with which an application or user can interact to submit Ethereum transactions.

Concepts

Terminology

- **Consensus** - The act of synchronizing data across the distributed network through block validation and creation.
- **Consortium member** - An entity that participates in consensus on the Blockchain network.
- **Admin** - An Ethereum account that is used to manage participation for a given consortium member.
- **Validator** - A machine associated with an Ethereum account that participates in consensus on behalf of an Admin.

Proof-of-authority

For those of you who are new to the blockchain community, the release of this solution is a great opportunity to learn about the technology in an easy and configurable manner on Azure. Proof-of-work is a Sybil-resistance mechanism that leverages computation costs to self-regulate the network and allow fair participation. This works great in anonymous, open blockchain networks where competition for cryptocurrency promotes security on the network. However, in private/consortium networks the underlying Ether has no value. An alternative protocol, proof-of-authority, is more suitable for permitted networks where all consensus participants are known and reputable. Without the need for mining, Proof-of-authority is more efficient while still retaining Byzantine fault tolerance.

Consortium governance

Since proof-of-authority relies upon a permitted list of network authorities to keep the network healthy, it's important to provide a fair mechanism to make modifications to this permission list. Each deployment comes with a set of smart-contracts and portal for on-chain governance of this permitted list. Once a proposed change reaches a majority vote by consortium members, the change is enacted. This allows new consensus participants to be added or compromised participants to be removed in a transparent way that encourages an honest network.

Admin account

During the deployment of the proof-of-authority nodes, you'll be asked for an Admin Ethereum address. You may use several different mechanisms to generate and secure this Ethereum account. Once this address is added as an authority on the network, you can use this account to participate in governance. This admin account will also be used to delegate consensus participation to the validator nodes that are created as part of this deployment. Since only the public Ethereum address is used, each admin has the flexibility to secure their private keys in a way that follows their wanted security model.

Validator node

In the proof-of-authority protocol, validator nodes take the place of traditional miner nodes. Each validator has a unique Ethereum identity that gets added to a smart-contract permission list. Once a validator is on this list, it can participate in the block creation process. To learn more about this process, see Parity's documentation on Authority Round consensus. Each consortium member can provision two or more validator nodes across five regions, for geo-redundancy. Validator nodes communicate with other validator nodes to come to consensus on the state of the underlying distributed ledger. To ensure fair participation on the network, each consortium member is prohibited from using more validators than the first member on the network (if the first member deploys three validators, each member can only have up to three validators).

Identity store

Since each member will have multiple validator nodes running simultaneously and each node must have a permitted identity, it's important that the validators can safely acquire a unique active identity on the network. To make this easier, we've built an Identity Store that gets deployed in each member's subscription that securely holds the generated Ethereum identities. Upon deployment, the orchestration container will generate an Ethereum private key for each validator and store it in Azure Key Vault. Before the parity node starts up, it first acquires a lease on an unused identity to ensure the identity isn't picked up by another node. The identity is provided to the client which gives it the authority to start creating blocks. If the hosting VM experiences an outage, the identity lease will be released, allowing a replacement node to resume its identity in the future.

Bootnode registrar

To enable the ease of connectivity, each member will host a set of connection information at the data API endpoint. This data includes a list of bootnodes that are provided as peering nodes for the joining member. As part of this data API, we keep this bootnode list up-to-date

Bring your own operator

Often a consortium member will want to participate in network governance but don't want to operate and maintain their infrastructure. Unlike traditional systems, having a single operator across the network works against the decentralized model of blockchain systems. Instead of hiring a centralized intermediary to operate a network, each consortium member can delegate infrastructure management to the operator of their choosing. This allows a hybrid model where each member can choose to operate their own infrastructure or delegate operation to a different partner. The delegated operation workflow works as follows:

1. **Consortium Member** generates an Ethereum address (holds private key)
2. **Consortium Member** provides public Ethereum address to **Operator**
3. **Operator** deploys and configures the PoA validator nodes using our Azure Resource Manager solution
4. **Operator** provides the RPC and management endpoint to **Consortium Member**
5. **Consortium Member** uses their private key to sign a request accepting the validator nodes **Operator** has deployed to participate on their behalf

Azure Monitor

This solution also comes with Azure Monitor to track node and network statistics. For application developers, this provides visibility into the underlying blockchain to track block generation statistics. Network operators can use Azure Monitor to quickly detect and prevent network outages through infrastructure statistics and queryable logs.

For more information, see Service monitoring.

Deployment architecture

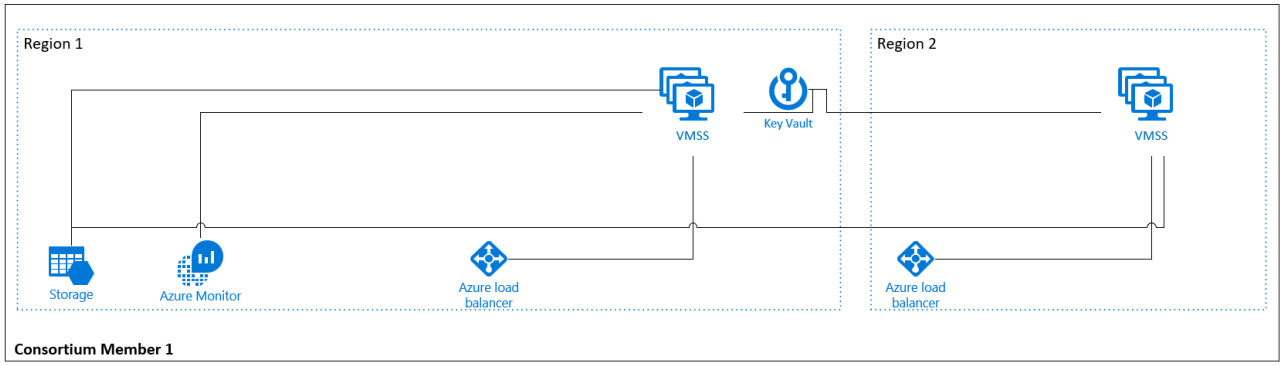
Description

This solution can deploy a single or multi-region based multi-member Ethereum consortium network. By default, the RPC and peering endpoints are accessible over public IP to enable simplified connectivity across subscriptions and clouds. We recommend leveraging Parity's permissioning contracts for application level access-controls. We also support networks deployed behind VPNs, which leverage VNet gateways for cross-subscription connectivity. These deployments are more complex, so it is recommended to start with the public IP model first.

Consortium member overview

Each consortium member deployment includes:

- Virtual Machines for running the PoA validators
- Azure Load Balancer for distributing RPC, peering, and Governance DApp requests
- Azure Key Vault for securing the validator identities
- Azure Storage for hosting persistent network information and coordinating leasing
- Azure Monitor for aggregating logs and performance statistics
- VNet Gateway (optional) for allowing VPN connections across private VNets



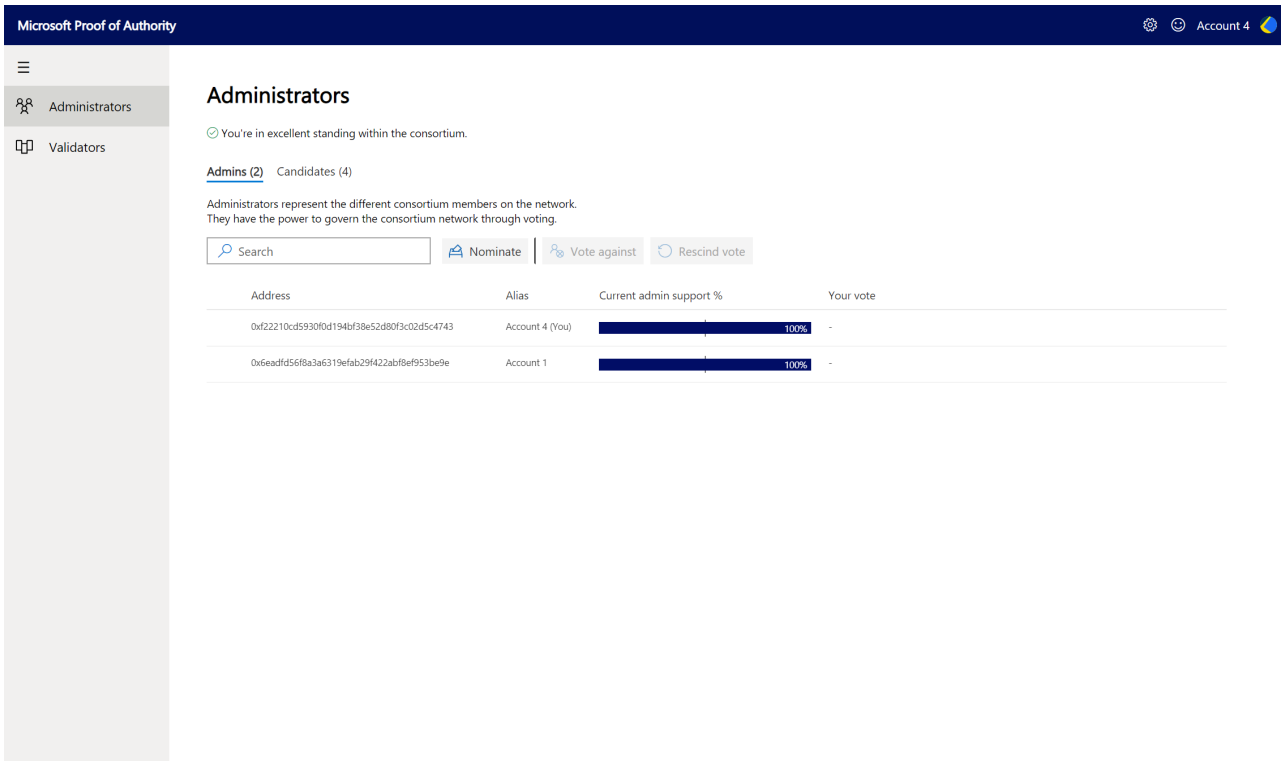
We leverage Docker containers for reliability and modularity. We use Azure Container Registry to host and serve versioned images as part of each deployment. The container images consist of:

- Orchestrator
  - Runs once during deployment
  - Generates identities and governance contracts
  - Stores identities in Identity Store
- Parity Client
  - Leases identity from Identity Store
  - Discovers and connects to peers
- EthStats Agent
  - Collects local logs and stats via RPC and pushes to Azure Monitor
- Governance DApp
  - Web interface for interacting with Governance contracts

How-to guides

Governance DApp

At the heart of proof-of-authority is decentralized governance. The governance DApp is a set of pre-deployed smart contracts and a web application that are used to govern the authorities on the network. Authorities are broken up into Admin identities and Validator nodes. Admins have the power to delegate consensus participation to a set of Validator nodes. Admins also may vote other admins into or out of the network.



- **Decentralized Governance -** Changes in network authorities are administered through on-chain voting by select administrators.
- **Validator Delegation -** Authorities can manage their validator nodes that are set up in each PoA deployment.
- **Auditable Change History -** Each change is recorded on the blockchain providing transparency and auditability.

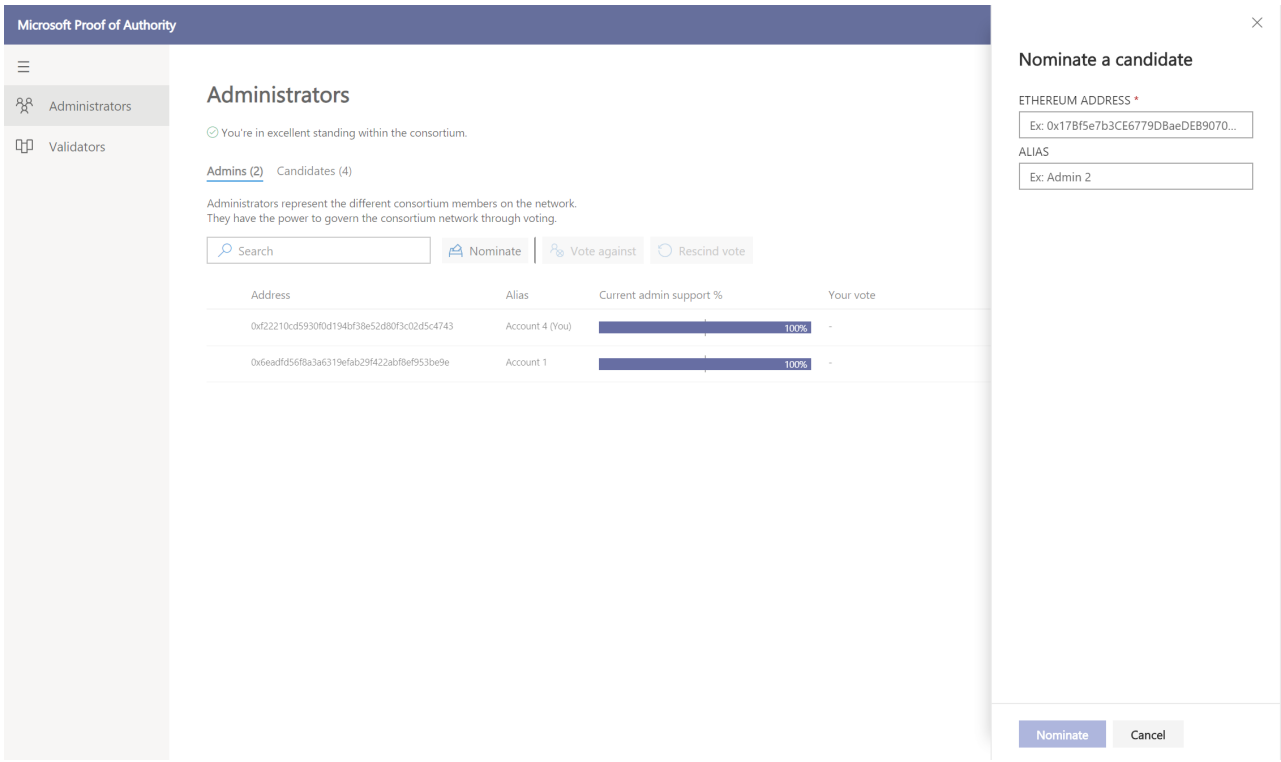
Getting started with governance

To perform any kind of transactions through the Governance DApp, you'll need to leverage an Ethereum wallet. The most straightforward approach is to use an in-browser wallet such as MetaMask; however, because these are smart contracts deployed on the network you may also automate your interactions to the Governance contract.

After installing MetaMask, navigate to the Governance DApp in the browser. You can locate the URL in the deployment confirmation email or through Azure portal in the deployment output. If you don't have an in-browser wallet installed you'll not be able to perform any actions; however, you still can read the administrator state.

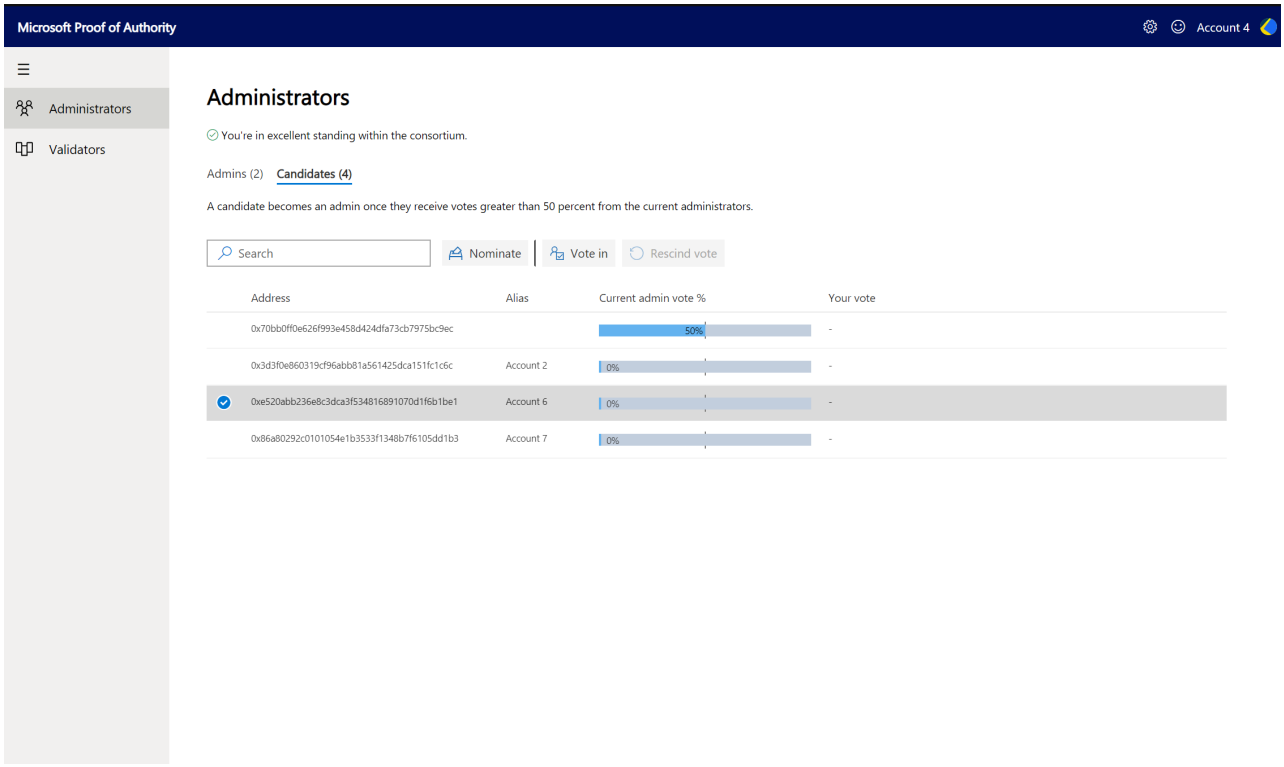
Becoming an admin

If you're the first member that deployed on the network, then you'll automatically become an Admin and your Parity nodes will be listed as Validators. If you're joining the network, you'll need to get voted in as an Admin by a majority (greater than 50%) of the existing Admin set. If you choose not to become an Admin then your nodes will still sync and validate the blockchain; however, they will not participate in the block creation process. To start the voting process to become an Admin, click **Nominate** and enter your Ethereum address and alias.



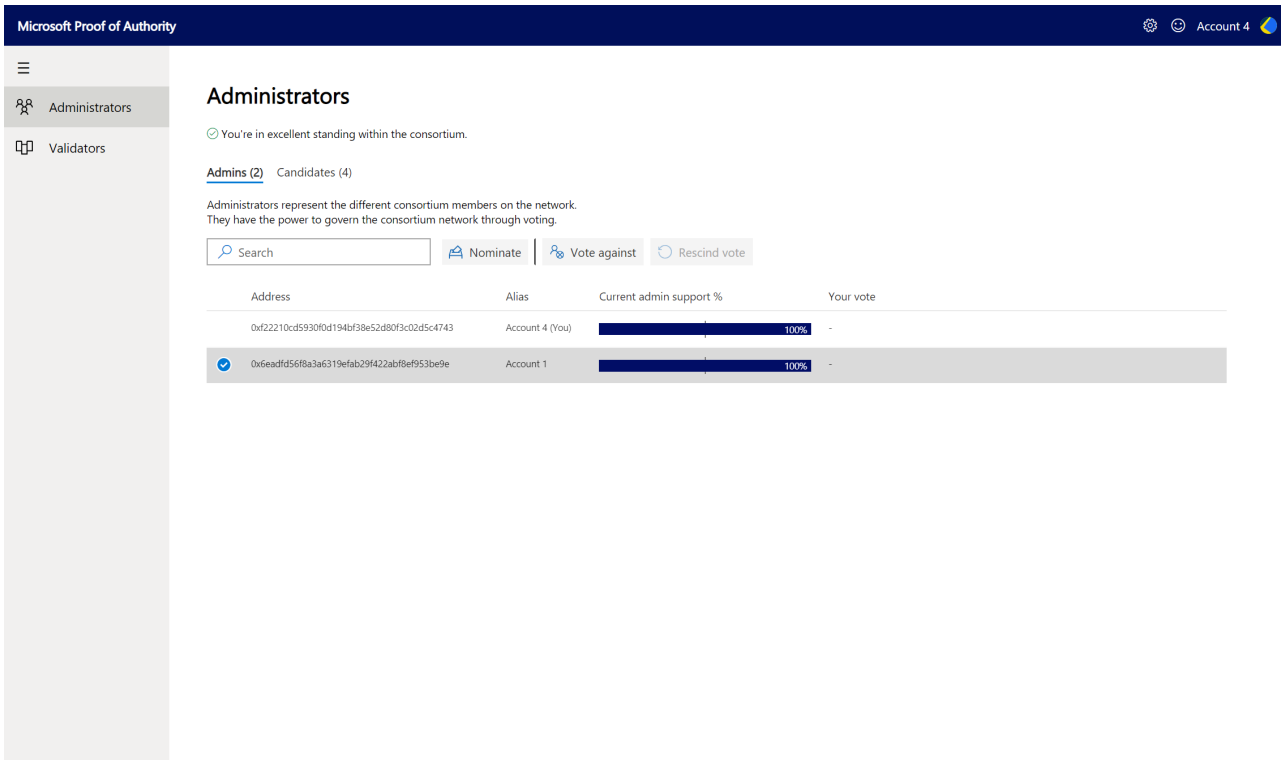
Candidates

Selecting the **Candidates** tab will show you the current set of candidate administrators. Once a Candidate reaches a majority vote by the current Admins, the Candidate will get promoted to an Admin. To vote on a Candidate, select the row and click "Vote in" at the top. If you change your mind on a vote, you may select the candidate and click "Rescind vote".



Admins

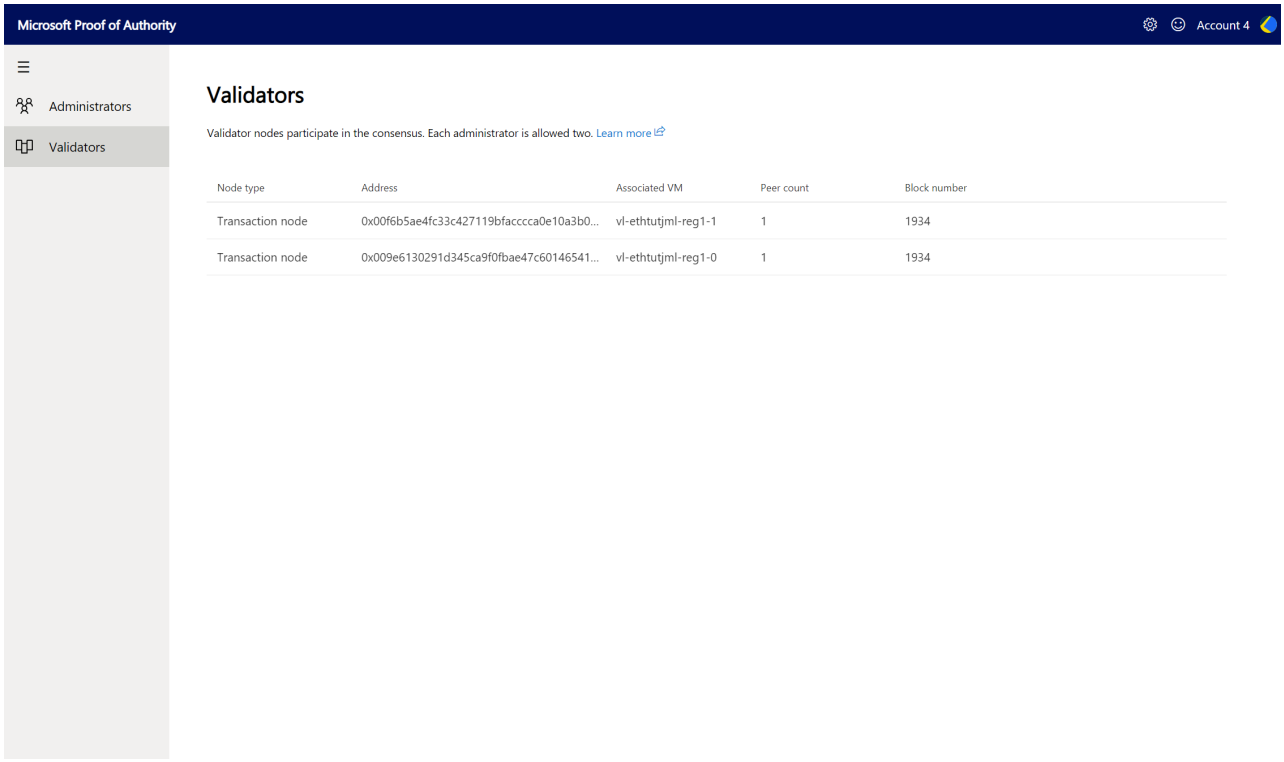
The **Admins** tab will show the current set of Admins and provide you the ability to vote against. Once an Admin loses more than 50% support, they'll be removed as an Admin on the network. Any validator nodes that this Admin owns will lose validator status and become transaction nodes on the network. An Admin may be removed for any number of reasons; however, it's up to the consortium to agree on a policy in advance.



Validators

Selecting the **Validators** tab in the left menu will display the current deployed Parity nodes for this instance and their current status (Node type). Each consortium member will have a different set of validators in this list, since this view represents the current deployed consortium member. If this is a newly deployed instance and you haven't yet added your validators, you'll be shown the option to 'Add Validators'. Selecting this will automatically choose a regionally balanced set of Parity nodes and assign them to your validator set. If you have deployed more nodes than the allowed capacity, the remaining nodes will become transaction nodes on the network.

The address of each validator is automatically assigned via the identity store in Azure. If a node goes down, it will relinquish its identity, allowing another node in your deployment to take its place. This ensures that your consensus participation is highly available.



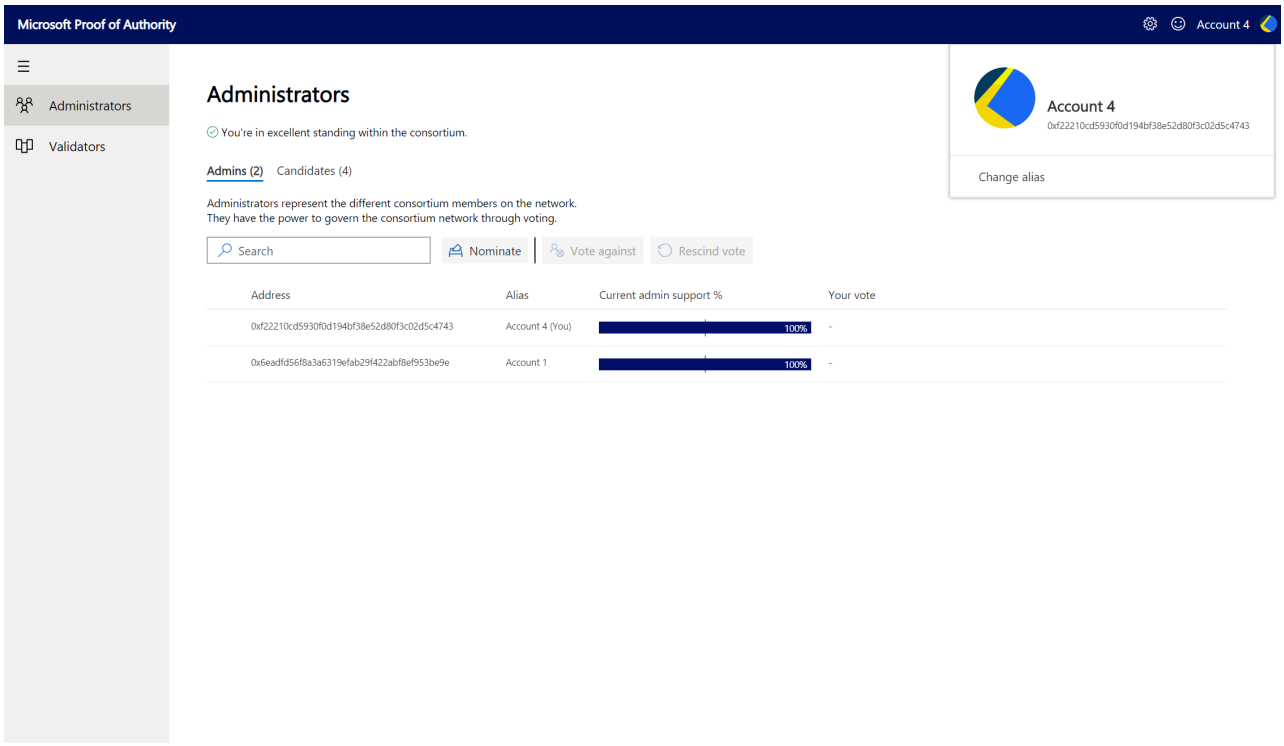
Consortium name

Any Admin may update the Consortium Name, displayed at the top of the page. Select the gear icon in the top left

to update the Consortium Name.

Account menu

In the top-right is your Ethereum account alias and identicon. If you're an Admin you'll have the ability to update your alias.



Deploy Ethereum Proof-of-Authority

Here's an example of a multi-party deployment flow:

1. Three members each generate an Ethereum account using MetaMask
2. *Member A* deploys Ethereum PoA, providing their Ethereum Public Address
3. *Member A* provides the consortium URL to *Member B* and *Member C*
4. *Member B* and *Member C* deploy, Ethereum PoA, providing their Ethereum Public Address and *Member A*'s consortium URL
5. *Member A* votes in *Member B* as an admin
6. *Member A* and *Member B* both vote *Member C* as an admin

This process requires an Azure subscription that can support deploying several virtual machines and managed disks. If necessary, create a free Azure account to begin.

Once a subscription is secured, go to Azure portal. Select '+', Marketplace ('See all'), and search for Ethereum PoA Consortium.

The following section will walk you through configuring the first member's footprint in the network. The deployment flow is divided into five steps: Basics, Deployment regions, Network size and performance, Ethereum settings, Azure Monitor.

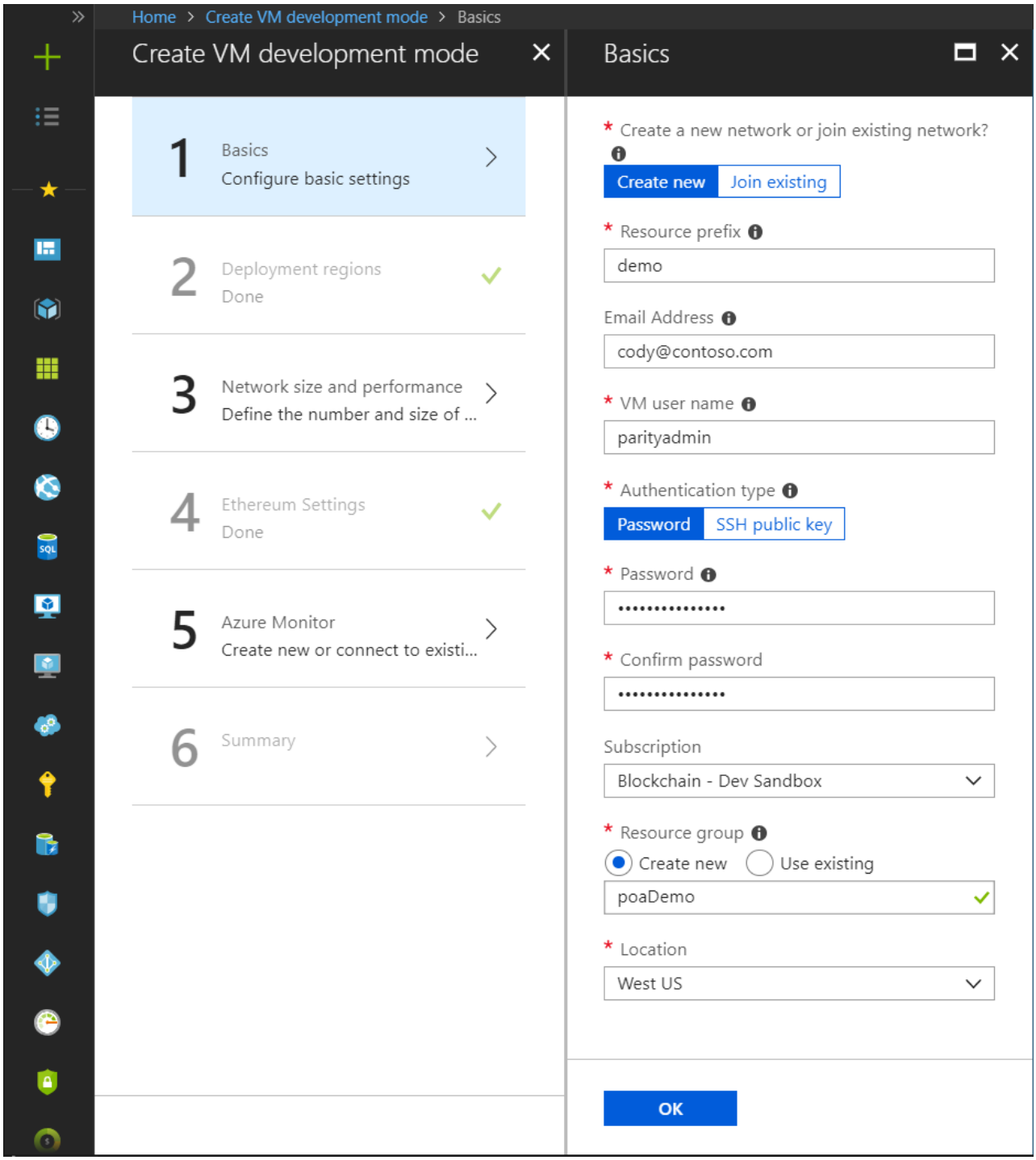
Basics

Under **Basics**, specify values for standard parameters for any deployment, such as subscription, resource group and basic virtual machine properties.

A detailed description of each parameter follows:

PARAMETER NAME	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUES
Create a new network or join existing network?	Create a new network or join a pre-existing consortium network	Create New Join Existing	Create New
Email Address (Optional)	You'll receive an email notification when your deployment completes with information about your deployment.	Valid email address	NA
VM user name	Administrator username of each deployed VM (alphanumeric characters only)	1-64 characters	NA
Authentication type	The method to authenticate to the virtual machine.	Password or SSH public key	Password
Password (Authentication type = Password)	The password for the administrator account for each of the virtual machines deployed. The password must contain 3 of the following: 1 upper case character, 1 lower case character, 1 number, and 1 special character. While all VMs initially have the same password, you can change the password after provisioning.	12-72 characters	NA
SSH Key (Authentication type = Public Key)	The secure shell key used for remote login.		NA
Subscription	The subscription to which to deploy the consortium network		NA
Resource Group	The resource group to which to deploy the consortium network.		NA
Location	The Azure region for resource group.		NA

A sample deployment is shown below:



Deployment regions

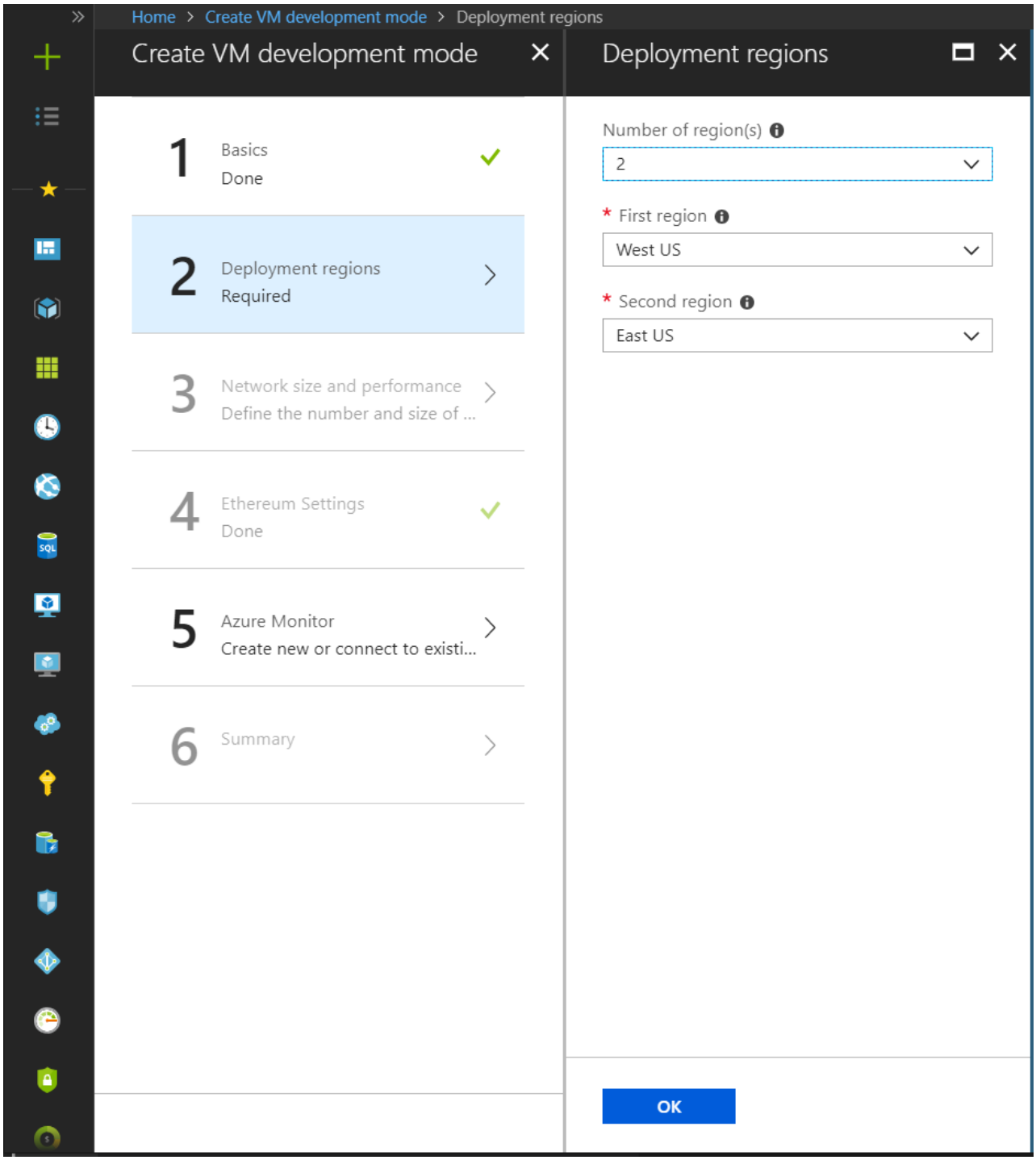
Next, under Deployment regions, specify inputs for number of region(s) to deploy the consortium network and selection of Azure regions based on the number of regions given. User can deploy in maximum of 5 regions. We recommend choosing the first region to match the resource group location from Basics section. For development or test networks, a single region per member is recommended. For production, we recommend deploying across two or more regions for high-availability.

A detailed description of each parameter follows:

PARAMETER NAME	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUES
Number of region(s)	Number of regions to deploy the consortium network	1, 2, 3, 4, 5	1
First region	First region to deploy the consortium network	All allowed Azure regions	NA

PARAMETER NAME	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUES
Second region	Second region to deploy the consortium network (Visible only when number of regions is selected as 2)	All allowed Azure regions	NA
Third region	Third region to deploy the consortium network (Visible only when number of regions is selected as 3)	All allowed Azure regions	NA
Fourth region	Fourth region to deploy the consortium network (Visible only when number of regions is selected as 4)	All allowed Azure regions	NA
Fifth region	Fifth region to deploy the consortium network (Visible only when number of regions is selected as 5)	All allowed Azure regions	NA

A sample deployment is shown below:



Network size and performance

Next, under 'Network size and performance' specify inputs for the size of the consortium network, such as number and size of validator nodes. The validator node storage size will dictate the potential size of the blockchain. This can be changed after deployment.

A detailed description of each parameter follows:

PARAMETER NAME	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUES
Number of load balanced validator nodes	The number of validator nodes to provision as part of the network	2-15	2
Validator node storage performance	The type of managed disk backing each of the deployed validator nodes.	Standard SSD or Premium	Standard SSD

PARAMETER NAME	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUES
Validator node virtual machine size	The virtual machine size used for validator nodes.	Standard A, Standard D, Standard D-v2, Standard F series, Standard DS, and Standard FS	Standard D1 v2

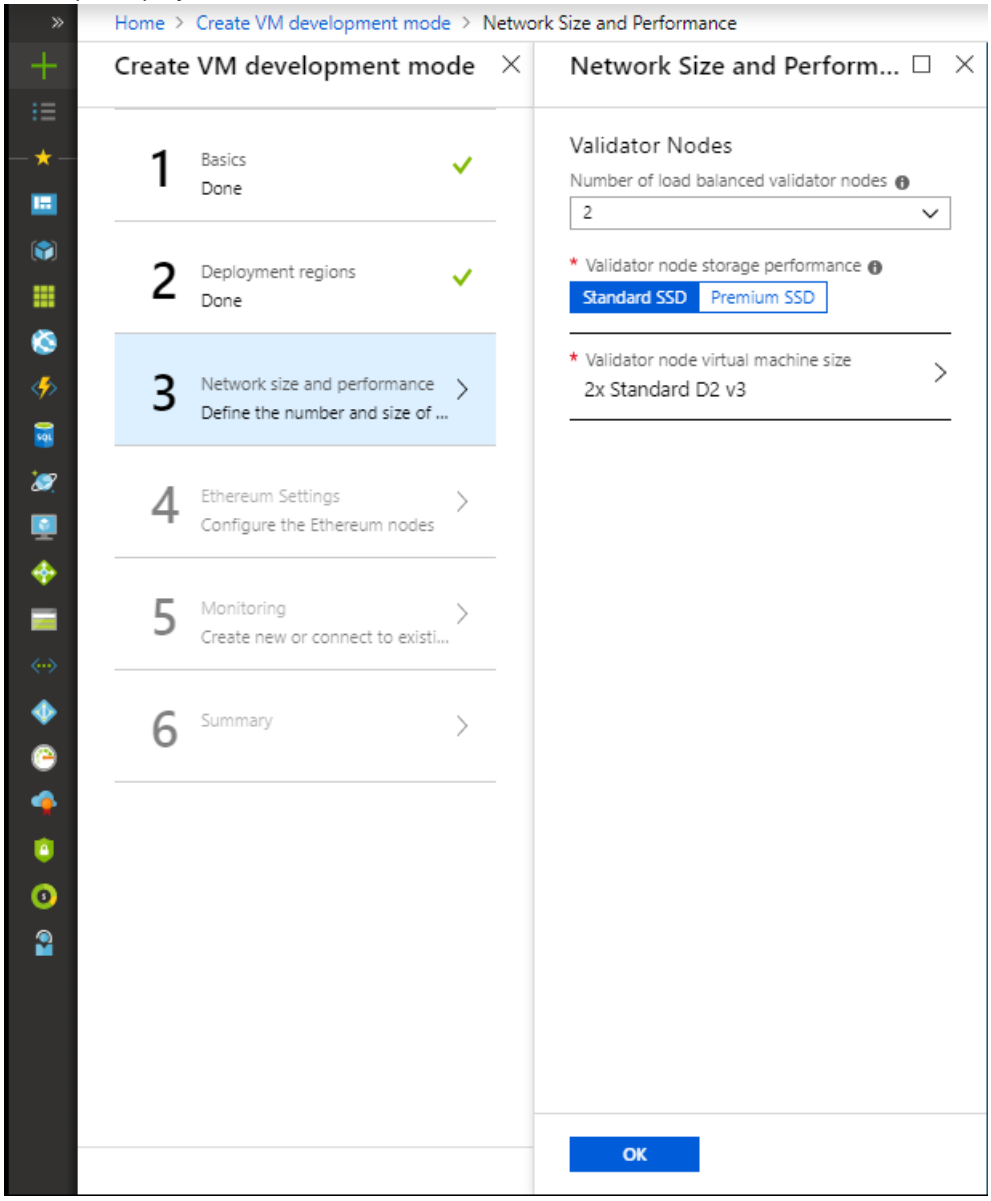
Storage Pricing Details

Virtual Machine Pricing Details

Virtual Machine and Storage Tier will affect network performance. We recommend the following SKUs based on desired cost-efficiency:

VIRTUAL MACHINE SKU	STORAGE TIER	PRICE	THROUGHPUT	LATENCY
F1	Standard SSD	low	low	high
D2_v3	Standard SSD	medium	medium	medium
F16s	Premium SSD	high	high	low

A sample deployment is shown below:



Ethereum settings

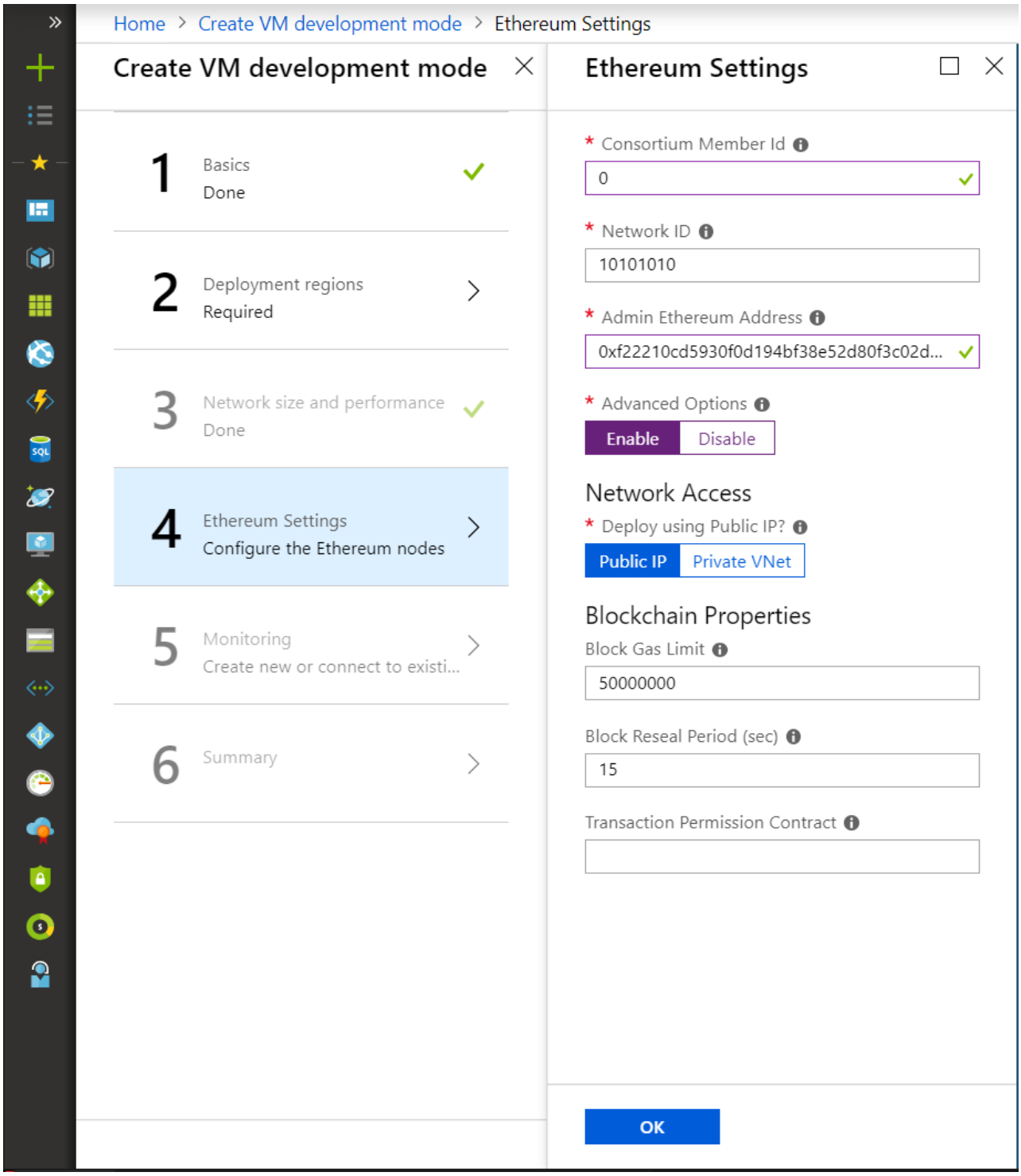
Next, under Ethereum settings, specify Ethereum-related configuration settings, like the network ID and Ethereum account password or genesis block.

A detailed description of each parameter follows:

PARAMETER NAME	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUES
Consortium Member ID	The ID associated with each member participating in the consortium network used to configure IP address spaces to avoid collision. In the case of a private network, Member ID should be unique across different organizations in the same network. A unique member ID is needed even when the same organization deploys to multiple regions. Make note of the value of this parameter since you'll need to share it with other joining members to ensure there's no collision.	0-255	NA
Network ID	The network ID for the consortium Ethereum network being deployed. Each Ethereum network has its own Network ID, with 1 being the ID for the public network.	5 - 999,999,999	10101010
Admin Ethereum Address	Ethereum account address that is used for participating in PoA governance. We recommend using MetaMask for generating an Ethereum address.	42 alphanumeric characters starting with 0x	NA
Advanced Options	Advanced options for Ethereum settings	Enable or Disable	Disable
Public IP (Advanced Options = Enable)	Deploys the network behind a VNet Gateway and removes peering access. If this option is selected, all members must use a VNet Gateway for the connection to be compatible.	Public IP Private VNet	Public IP
Block Gas Limit (Advanced Options = Enable)	The starting block gas limit of the network	Any numeric	50000000

PARAMETER NAME	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUES
Block Reseal Period (sec)	The frequency at which empty blocks will be created when there are no transactions on the network. A higher frequency will have faster finality but increased storage costs.	Any numeric	15
Transaction Permission Contract (Advanced Options = Enable)	Bytecode for the Transaction Permissioning contract. Restricts smart contract deployment and execution to a permitted list of Ethereum accounts.	Contract bytecode	NA

A sample deployment is shown below:



Monitoring

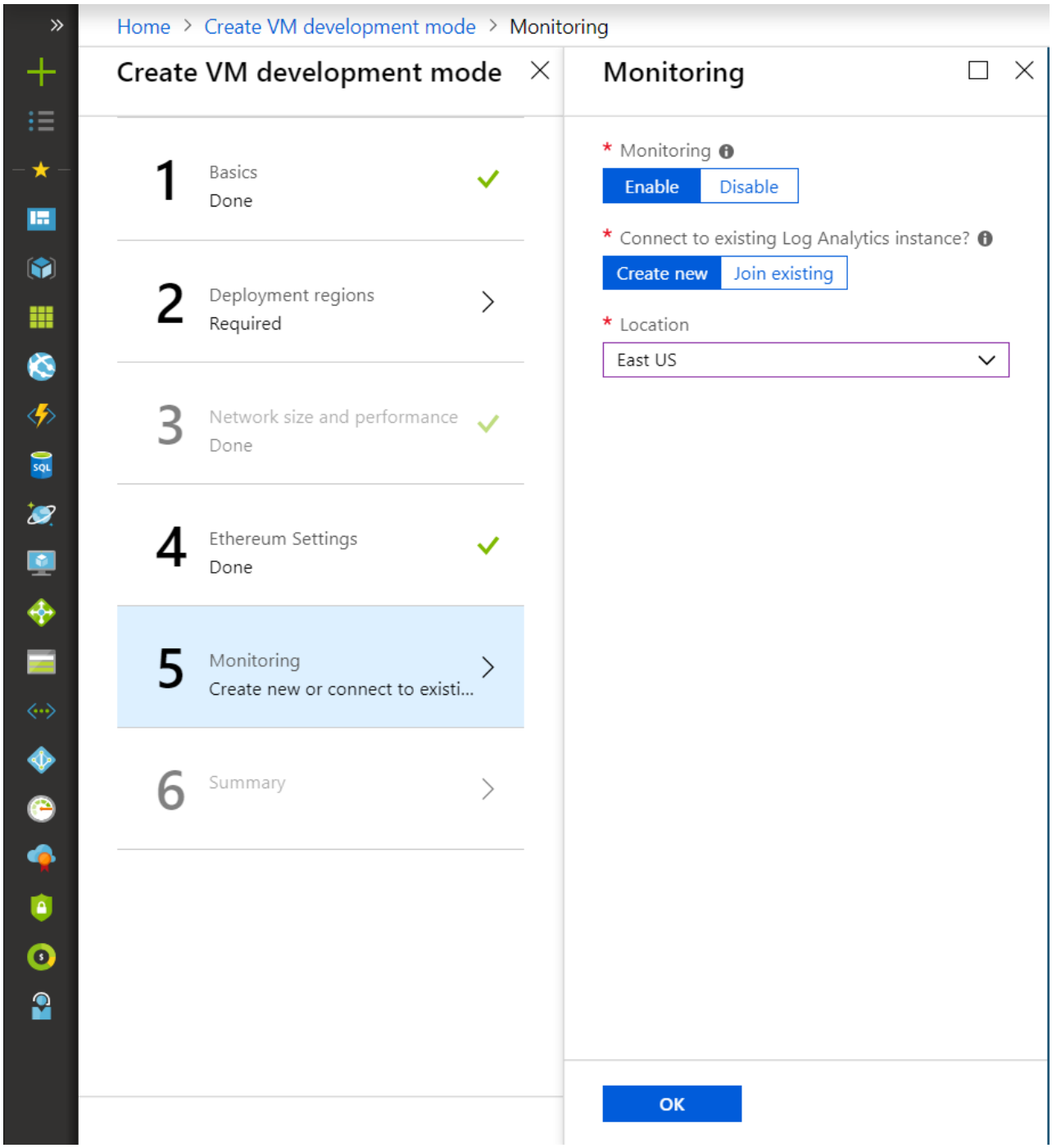
The Monitoring blade allows you to configure an Azure Monitor logs resource for your network. The monitoring agent will collect and surface useful metrics and logs from your network, providing the ability to quickly check the network health or debug issues.

> **NOTE**
>
> This article was recently updated to use the term Azure Monitor logs instead of Log Analytics. Log data is still stored in a Log Analytics workspace and is still collected and analyzed by the same Log Analytics service. We are updating the terminology to better reflect the role of logs in Azure Monitor. See Azure Monitor terminology changes for details.

PARAMETER NAME	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUES
Monitoring	Option to enable Monitoring	Enable or Disable	Enable

PARAMETER NAME	DESCRIPTION	ALLOWED VALUES	DEFAULT VALUES
Connect to existing Azure Monitor logs	Create a new Azure Monitor logs instance or join an existing instance	Create new or Join existing	Create new
Monitor Location(Connect to existing Azure Monitor logs= Create new)	The region where the new Azure Monitor logs instance will be deployed	All Azure Monitor logs regions	NA
Existing log analytics workspace ID (Connect to existing Azure Monitor logs = Join Existing)	Workspace ID of the existing Azure Monitor logs instance		NA
Existing log analytics primary key (Connect to existing Azure Monitor logs = Join Existing)	The primary key used to connect to the existing Azure Monitor logs instance		NA

A sample deployment is shown below:



Summary

Click through the summary blade to review the inputs specified and to run basic pre-deployment validation. Before deploying you may download the template and parameters.

Review legal and privacy terms and click 'Purchase' to deploy. If the deployment includes VNet Gateways, the deployment will take up 45 to 50 minutes.

Post deployment

Deployment output

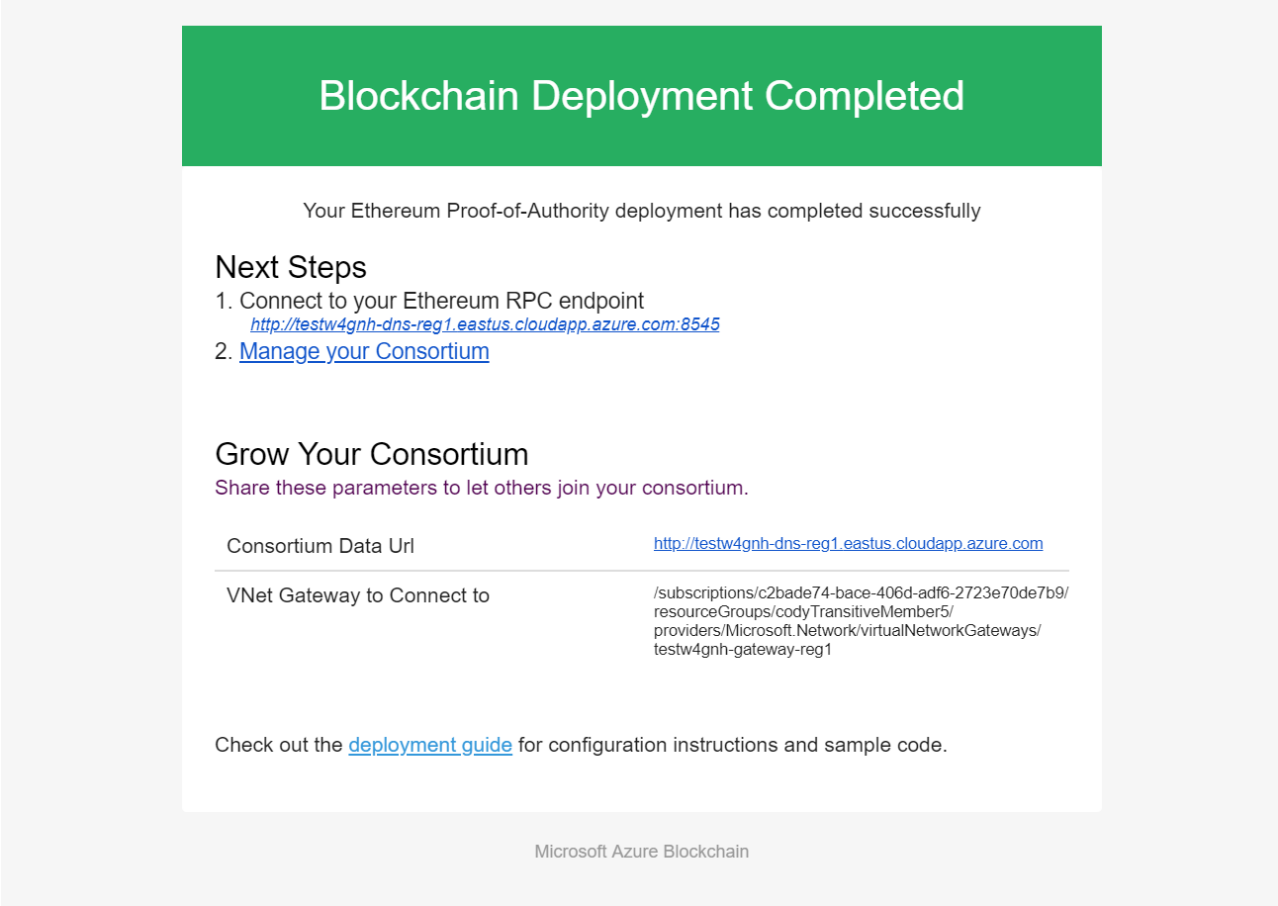
Once the deployment has completed, you can access the necessary parameters via the confirmation email or through the Azure portal. In these parameters you'll find:

- Ethereum RPC endpoint
- Governance Dashboard URL
- Azure Monitor URL
- Data URL

- VNet Gateway Resource ID (optional)

Confirmation email

If you provide an email address (Basics Section), an email would be sent to the email address with the deployment output information.



Portal

Once the deployment has completed successfully and all resources have been provisioned you can view the output parameters in your resource group.

1. Locate your resource group in the portal
2. Navigate to *Deployments*
3. Select the top deployment with the same name as your resource group
4. Select *Outputs*

Growing the consortium

To expand your consortium, you must first connect the physical network. Using the Public IP-based deployment this first step is seamless. If deploying behind a VPN, see the section Connecting VNet Gateway to do the network connection as part of the new member deployment. Once your deployment completes use the Governance DApp to become a network Admin.

New member deployment

1. Share the following information with the joining member. This information can be found in your post-deployment email or in the portal deployment output.
   - Consortium Data Url
   - The number of nodes you've deployed
   - VNet Gateway Resource ID (if using VPN)

2. The deploying member should use the same solution when deploying their network presence with keeping the following in mind:

    - Select *Join Existing*
    - Choose the same number of validator nodes as the rest of the members on the network to ensure fair representation
    - Use the same Ethereum address that was provided in the previous step
    - Pass in the provided *Consortium Data Url* on the *Ethereum Settings* tab
    - If the rest of the network is behind a VPN, select *Private VNet* under the advanced section

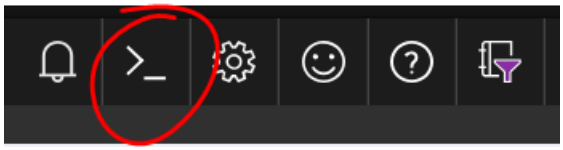
Connecting VNet gateways

You may ignore this step if you've deployed using the default Public IP settings. In the case of a private network, the different members are connected via VNet gateway connections. Before a member can join the network and see transaction traffic, an existing member must do a final configuration on their VPN gateway to accept the connection. This means that the Ethereum nodes of the joining member won't run until a connection is established. It's recommended to create redundant network connections (mesh) into the consortium to reduce chances of a single point of failure.

After the new member deploys, the existing member must complete the bi-directional connection by setting up a VNet gateway connection to the new member. To achieve this, existing member will need:

1. The VNet gateway ResourceID of the connecting member (see deployment output)
2. The shared connection key

The existing member must run the following PowerShell script to complete the connection. We recommend using Azure Cloud Shell located in the top-right navigation bar in the portal.

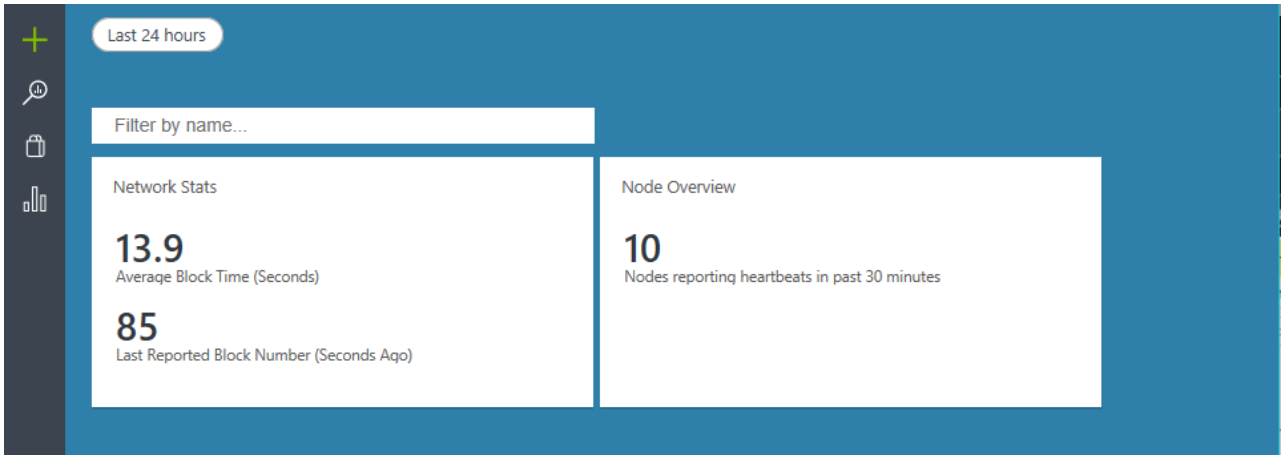


```
$MyGatewayResourceId = "<EXISTING_MEMBER_RESOURCEID>"
$OtherGatewayResourceId = "<NEW_MEMBER_RESOURCEID]"
$ConnectionName = "Leader2Member"
$SharedKey = "<NEW_MEMBER_KEY>"

## $myGatewayResourceId tells me what subscription I am in, what ResourceGroup and the VNetGatewayName
$splitValue = $MyGatewayResourceId.Split('/')
$MySubscriptionid = $splitValue[2]
$MyResourceGroup = $splitValue[4]
$MyGatewayName = $splitValue[8]

## $otherGatewayResourceid tells me what the subscription and VNet GatewayName are
$OtherGatewayName = $OtherGatewayResourceId.Split('/')[8]
$Subscription=Select-AzSubscription -SubscriptionId $MySubscriptionid

## create a PSVirtualNetworkGateway instance for the gateway I want to connect to
$OtherGateway=New-Object Microsoft.Azure.Commands.Network.Models.PSVirtualNetworkGateway
$OtherGateway.Name = $OtherGatewayName
$OtherGateway.Id = $OtherGatewayResourceId
$OtherGateway.GatewayType = "Vpn"
$OtherGateway.VpnType = "RouteBased"

## get a PSVirtualNetworkGateway instance for my gateway
$MyGateway = Get-AzVirtualNetworkGateway -Name $MyGatewayName -ResourceGroupName $MyResourceGroup

## create the connection
New-AzVirtualNetworkGatewayConnection -Name $ConnectionName -ResourceGroupName $MyResourceGroup -
VirtualNetworkGateway1 $MyGateway -VirtualNetworkGateway2 $OtherGateway -Location $MyGateway.Location -
ConnectionType Vnet2Vnet -SharedKey $SharedKey -EnableBgp $True
```

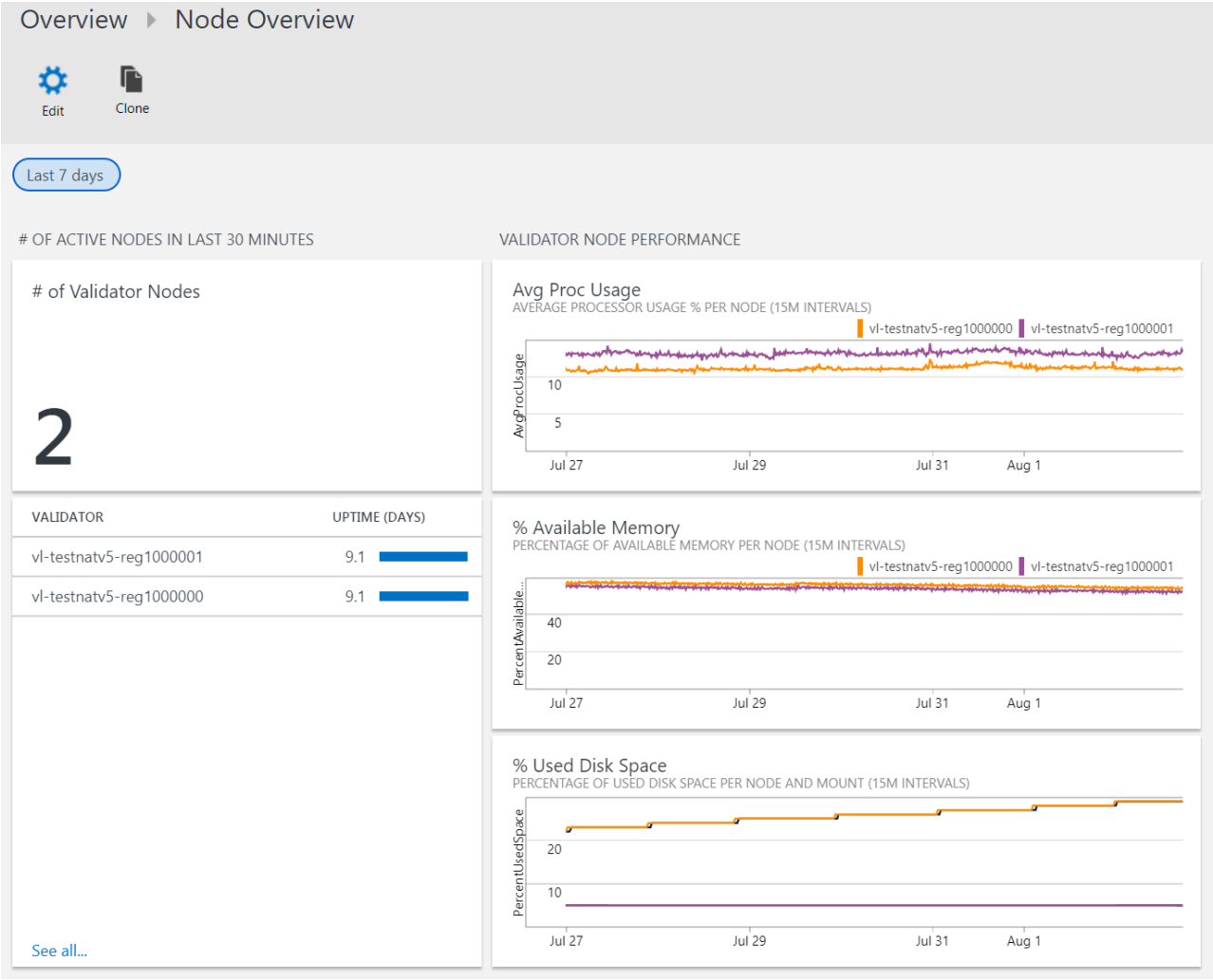
Service monitoring

You can locate your Azure Monitor portal either by following the link in the deployment email or locating the parameter in the deployment output [OMS_PORTAL_URL].

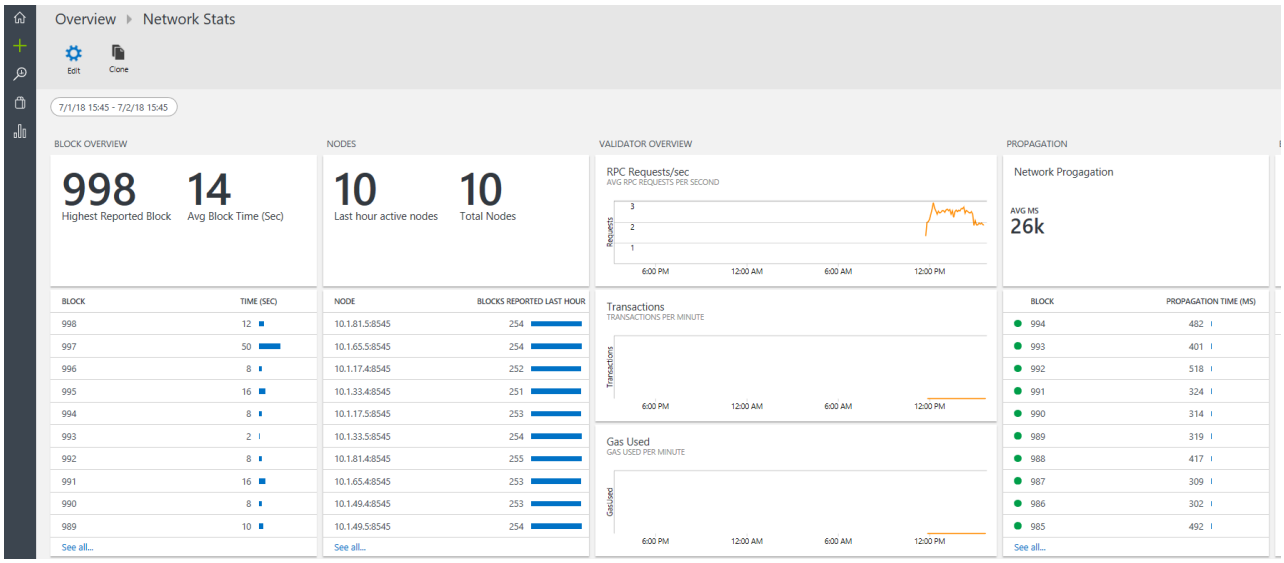
The portal will first display high-level network statistics and node overview.



Selecting **Node Overview** will direct you to a portal to view per-node infrastructure statistics.



Selecting **Network Stats** will direct you to view Ethereum network statistics.



Sample Kusto queries

Behind these dashboards is a set of queryable raw logs. You can use these raw logs to customize the dashboards, investigate failures, or setup threshold alerting. Below you'll find a set of example queries that can be ran in the Log Search tool:

Lists blocks that have been reported by more than one validator. Useful to help find chain forks.

```
MinedBlock_CL
| summarize DistinctMiners = dcount(BlockMiner_s) by BlockNumber_d, BlockMiner_s
| where DistinctMiners > 1
```

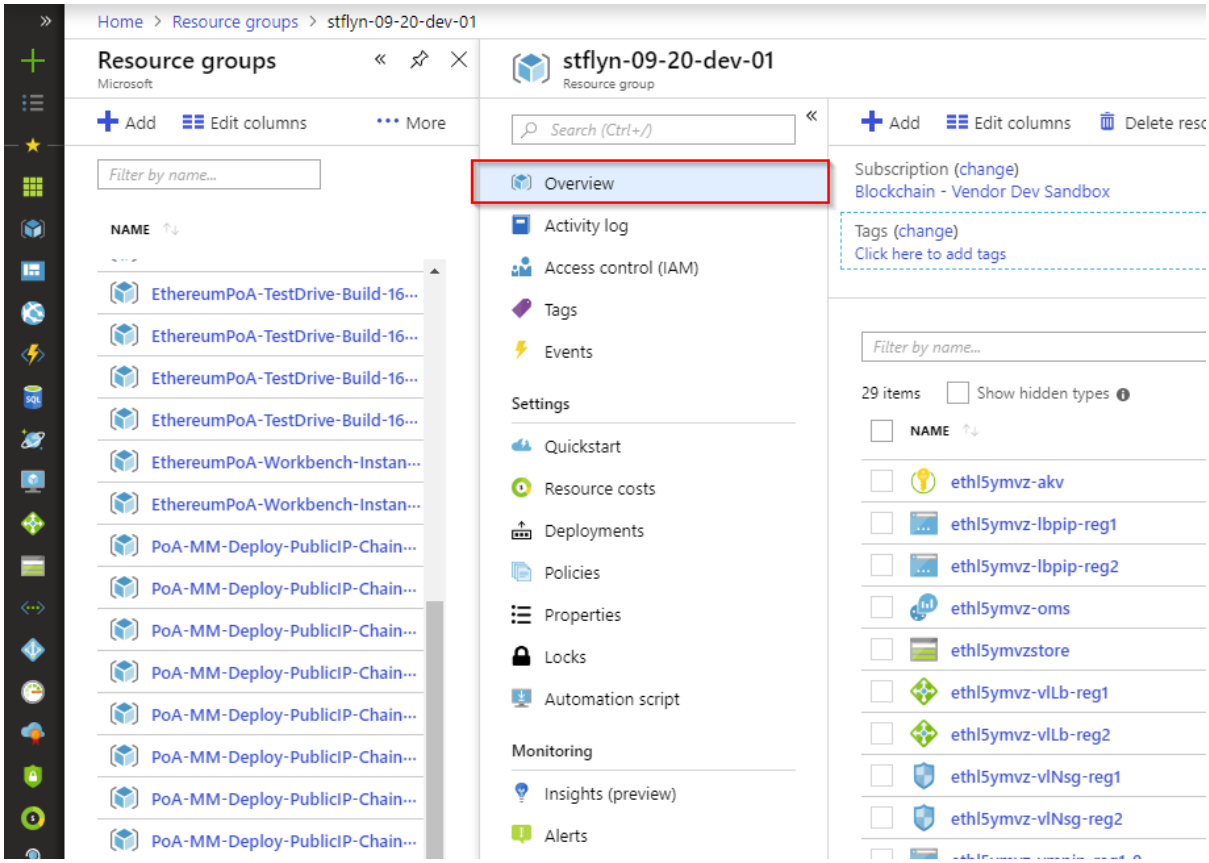
Get average peer count for a specified validator node averaged over 5 minute buckets.

```
let PeerCountRegex = @"Syncing with peers: (\d+) active, (\d+) confirmed, (\d+)";
ParityLog_CL
| where Computer == "vl-devn3lgdm-reg1000001"
| project RawData, TimeGenerated
| where RawData matches regex PeerCountRegex
| extend ActivePeers = extract(PeerCountRegex, 1, RawData, typeof(int))
| summarize avg(ActivePeers) by bin(TimeGenerated, 5m)
```

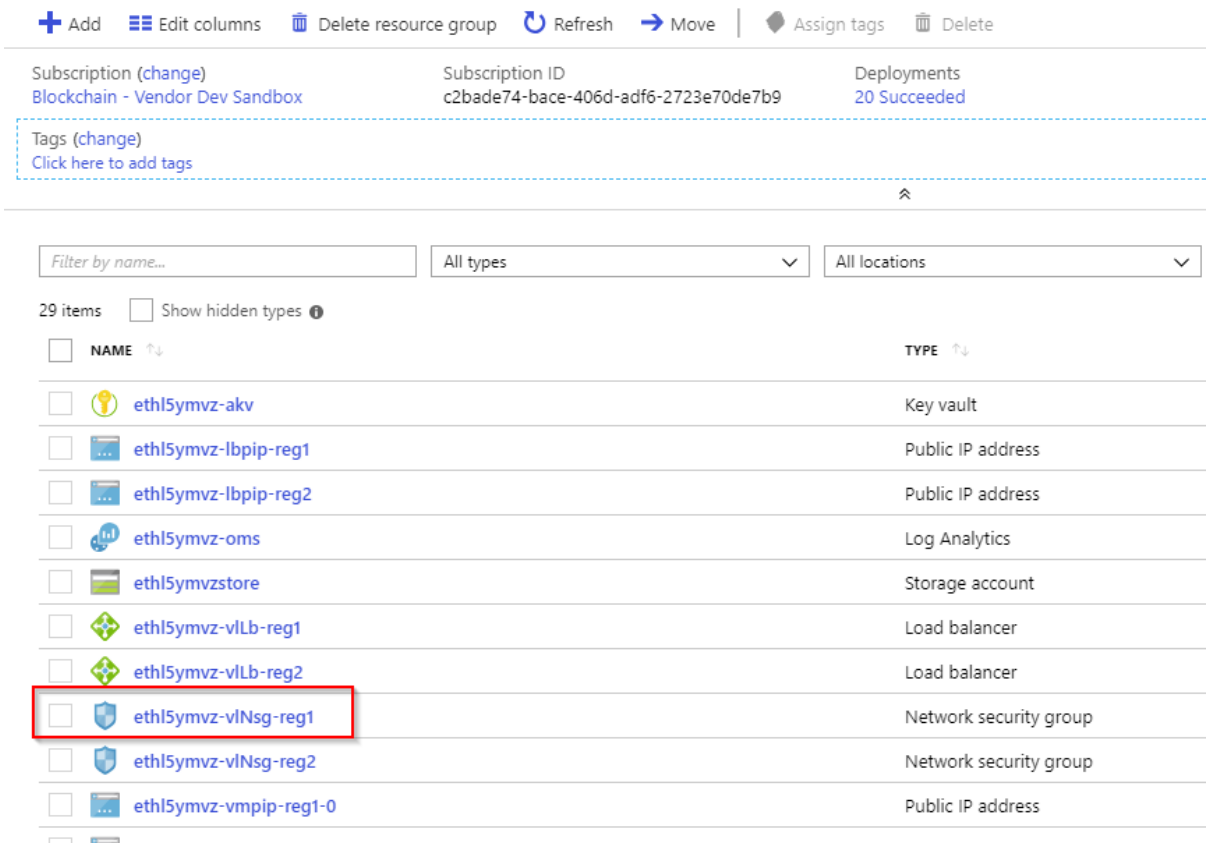
SSH access

For security reasons, the SSH port access is denied by a network group security rule by default. To access the virtual machine instances in the PoA network, you'll need to change this rule to "Allow"

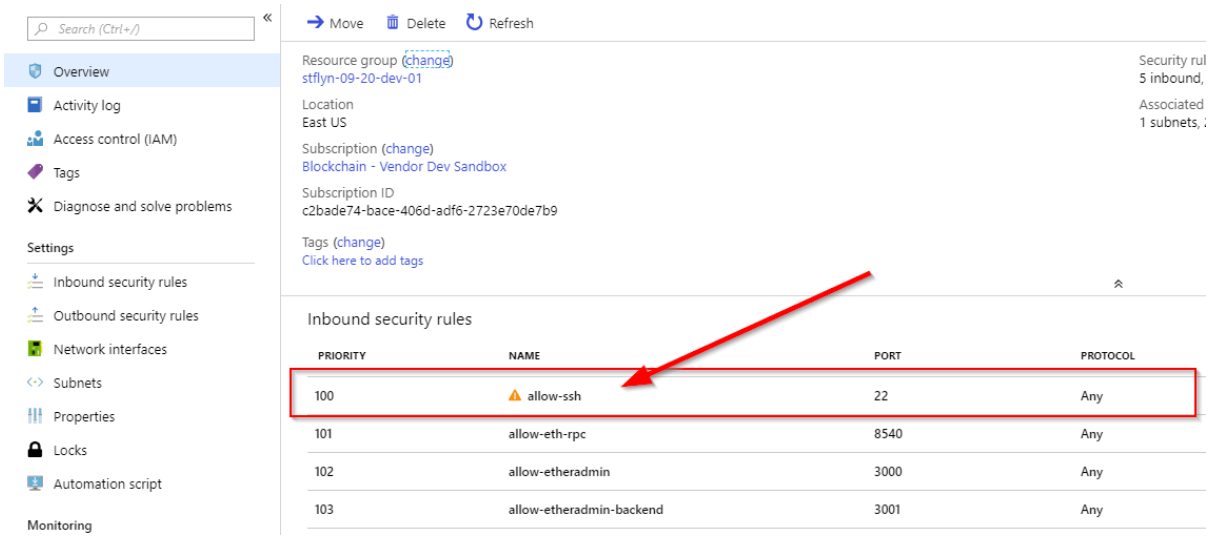
1. Start in the Overview section of the deployed resource group from Azure portal.



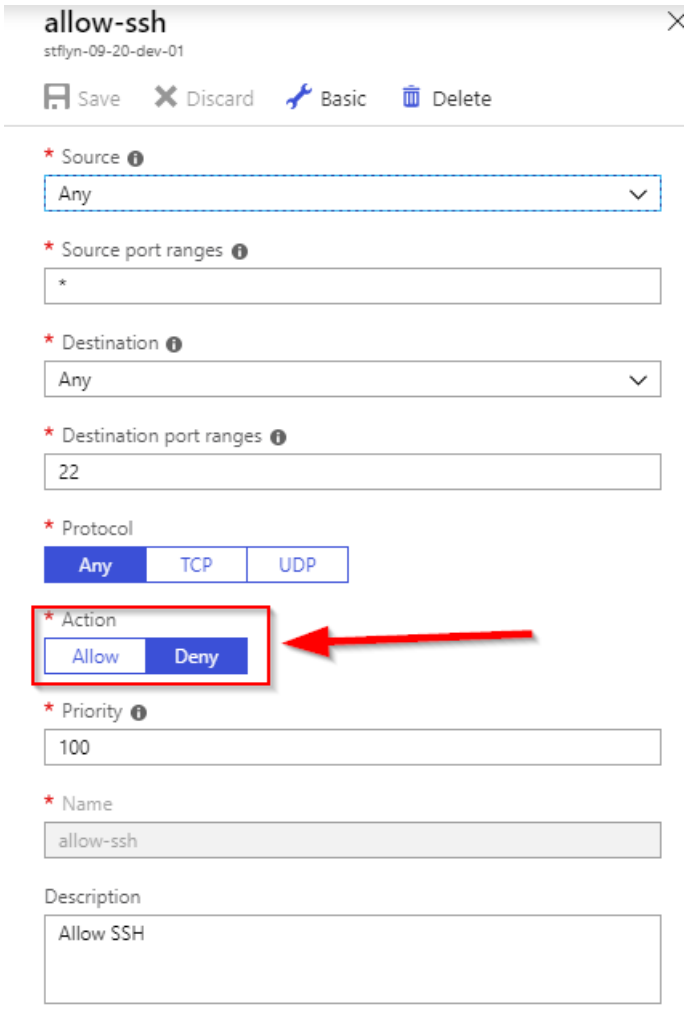
2. Select the Network Security Group for the region of the VM that you are wanting to access



3. Select the "allow-ssh" rule



4. Change "Action" to Allow



5. Click "Save" (Changes may take a few minutes to apply)

You can now remotely connect to the virtual machines for the validator nodes via SSH with your provided admin username and password/SSH key. The SSH command to run to access the first validator node is listed in the template deployment output parameter as, 'SSH_TO_FIRST_VL_NODE_REGION1' (for the sample deployment: ssh -p 4000 poaadmin@leader4vb.eastus.cloudapp.azure.com). To get to additional transaction nodes, increment the port number by one (For example, the first transaction node is on port 4000).

If you deployed to more than one region, change the above command to the DNS name or IP address of the load balancer in that region. To find the DNS name or IP address of the other regions, find the resource with the naming convention *****-lbpip-reg#, and view its DNS name and IP address properties.

Azure Traffic Manager load balancing

Azure Traffic Manager can help reduce downtime and improve responsiveness of the PoA network by routing incoming traffic across multiple deployments in different regions. Built-in health checks and automatic re-routing help ensure high availability of the RPC endpoints and the Governance DApp. This feature is useful if you have deployed to multiple regions and are production ready.

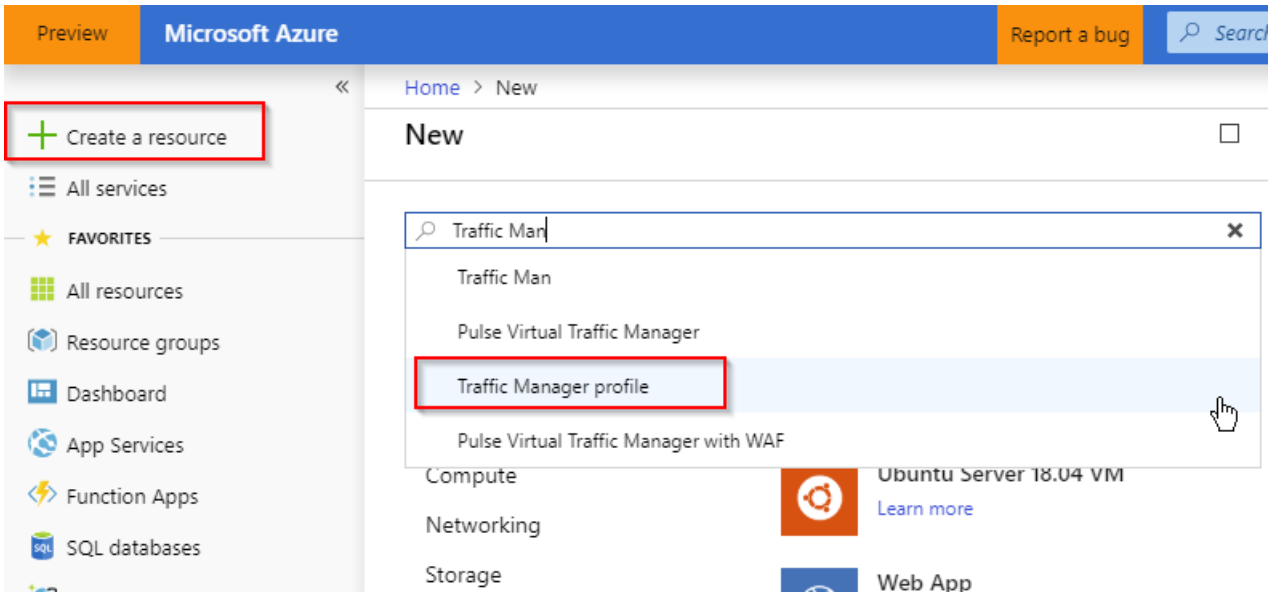
Use Traffic Manager to:

- Improve PoA network availability with automatic failover.
- Increase your networks responsiveness by routing end users to the Azure location with lowest network latency.

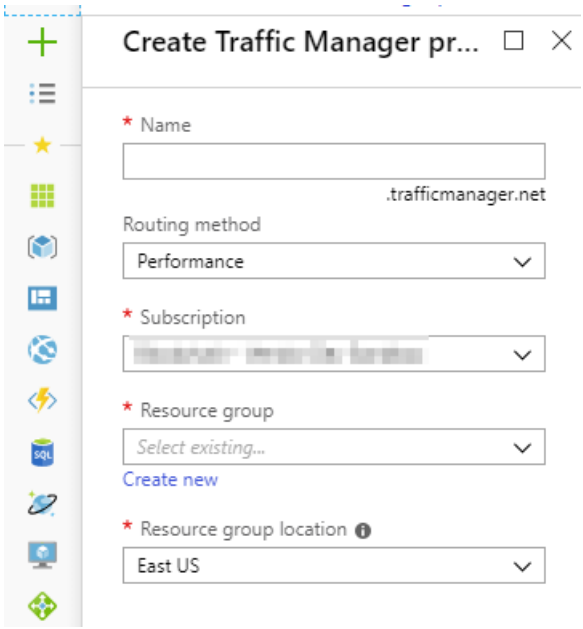
If you decide to create a Traffic Manager profile, you can use the DNS name of the profile to access your network. Once other consortium members have been added to the network, the Traffic Manager can also be used to load balance across their deployed validators.

Creating a Traffic Manager profile

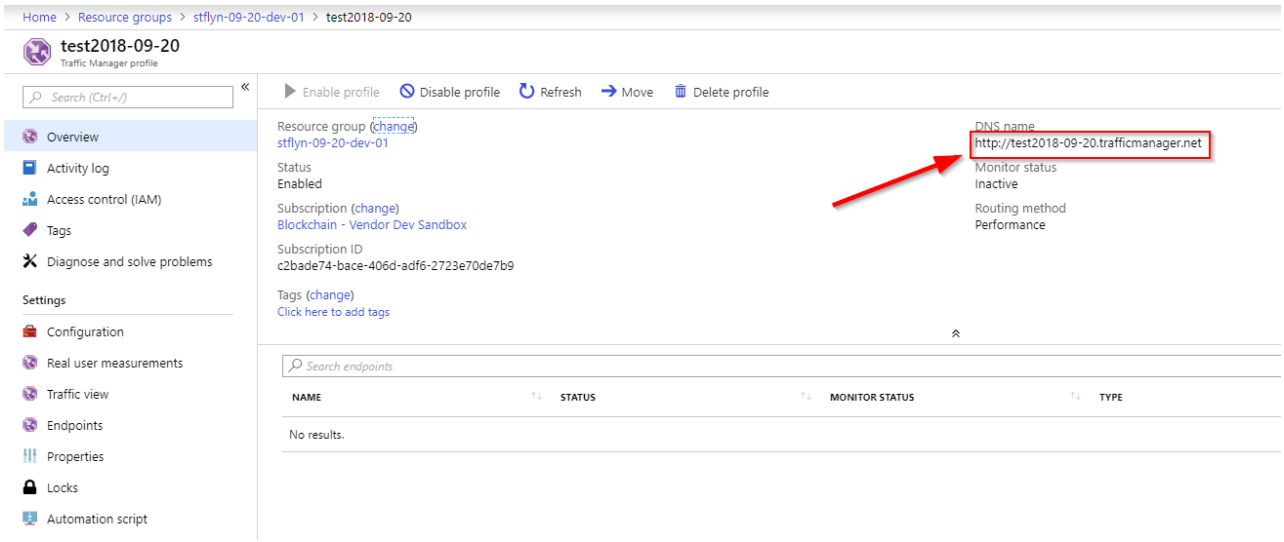
Search for and select "Traffic Manager profile" after clicking the "Create a resource" button in the Azure portal.



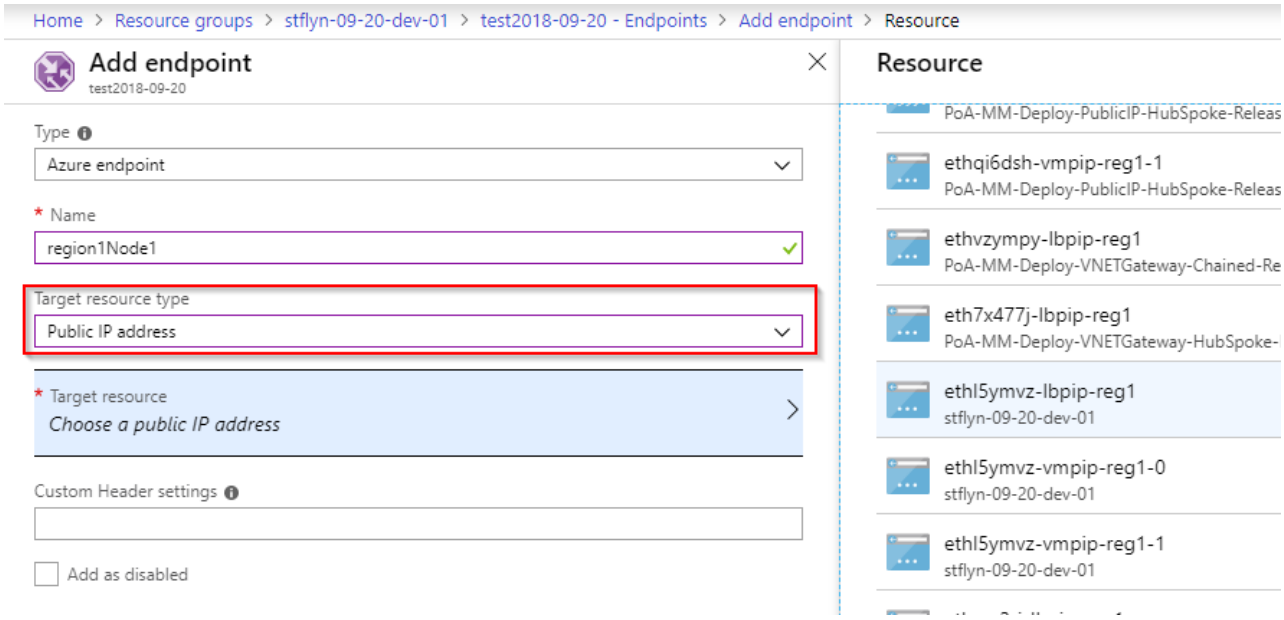
Give the profile a unique name and select the Resource Group that was created during the PoA deployment. Click the "Create" button to deploy.



Once it's deployed, then select the instance in the resource group. The DNS name to access the traffic manager can be found in the Overview tab



Select the Endpoints tab and click the Add button. Give the endpoint a unique name. Change the Target resource type to Public IP address. Then select the public IP address of the first region's load balancer.



Repeat for each region in the deployed network. Once the endpoints are in the "enabled" status, they'll be automatically load and region balanced at the DNS name of the traffic manager. You can now use this DNS name in place of the [CONSORTIUM_DATA_URL] parameter in other steps of the document.

Data API

Each consortium member hosts the necessary information for others to connect to the network. The existing member will provide the [CONSORTIUM_DATA_URL] before the member's deployment. Upon deployment, a joining member will retrieve information from the JSON interface at the following endpoint:

`<CONSORTIUM_DATA_URL>/networkinfo`

The response will contain information useful for joining members (Genesis block, Validator Set contract ABI, bootnodes) and information useful to the existing member (validator addresses). We encourage use of this standardization to extend the consortium across cloud providers. This API will return a JSON formatted response with the following structure:

```
{
  "$id": "",
  "type": "object",
  "definitions": {},
  "$schema": "https://json-schema.org/draft-07/schema#",
  "properties": {
```

```
  properties : {
    "majorVersion": {
      "$id": "/properties/majorVersion",
      "type": "integer",
      "title": "This schema’s major version",
      "default": 0,
      "examples": [
        0
      ]
    },
    "minorVersion": {
      "$id": "/properties/minorVersion",
      "type": "integer",
      "title": "This schema’s minor version",
      "default": 0,
      "examples": [
        0
      ]
    },
    "bootnodes": {
      "$id": "/properties/bootnodes",
      "type": "array",
      "items": {
        "$id": "/properties/bootnodes/items",
        "type": "string",
        "title": "This member’s bootnodes",
        "default": "",
        "examples": [

"enode://a348586f0fb0516c19de75bf54ca930a08f1594b7202020810b72c5f8d90635189d72d8b96f306f08761d576836a6bfce112cf
b6ae6a3330588260f79a3d0ecb@10.1.17.5:30300",

"enode://2d8474289af0bb38e3600a7a481734b2ab19d4eaf719f698fe885fb239f5d33faf217a860b170e2763b67c2f18d91c41272de3
7ac67386f80d1de57a3d58ddf2@10.1.17.4:30300"
        ]
      }
    },
    "valSetContract": {
      "$id": "/properties/valSetContract",
      "type": "string",
      "title": "The ValidatorSet Contract Source",
      "default": "",
      "examples": [
        "pragma solidity 0.4.21;\n\nimport \"./SafeMath.sol\";\nimport \"./Utils.sol\";\n\ncontract
ValidatorSet …"
      ]
    },
    "adminContract": {
      "$id": "/properties/adminContract",
      "type": "string",
      "title": "The AdminSet Contract Source",
      "default": "",
      "examples": [
        "pragma solidity 0.4.21;\nimport \"./SafeMath.sol\";\nimport \"./SimpleValidatorSet.sol\";\nimport
\"./Admin.sol\";\n\ncontract AdminValidatorSet is SimpleValidatorSet { …"
      ]
    },
    "adminContractABI": {
      "$id": "/properties/adminContractABI",
      "type": "string",
      "title": "The Admin Contract ABI",
      "default": "",
      "examples": [
        "[{\"constant\":false,\"inputs\":[{\"name\":\"proposedAdminAddress\",\"type\":\"address\"},…"
      ]
    },
    "paritySpec": {
      "$id": "/properties/paritySpec",
      "type": "string",
      "title": "The Parity client spec file"
```

```
      "title" : "The Parity client spec file",
      "default": "",
      "examples": [
        "\n{\n \"name\": \"PoA\",\n \"engine\": {\n \"authorityRound\": {\n \"params\": {\n \"stepDuration\":
\"2\",\n \"validators\" : {\n \"safeContract\": \"0x0000000000000000000000000000000000000006\"\n },\n
\"gasLimitBoundDivisor\": \"0x400\",\n \"maximumExtraDataSize\": \"0x2A\",\n \"minGasLimit\": \"0x2FAF080\",\n
\"networkID\" : \"0x9a2112\"\n }\n }\n },\n \"params\": {\n \"gasLimitBoundDivisor\": \"0x400\",\n
\"maximumExtraDataSize\": \"0x2A\",\n \"minGasLimit\": \"0x2FAF080\",\n \"networkID\" : \"0x9a2112\",\n
\"wasmActivationTransition\": \"0x0\"\n },\n \"genesis\": {\n \"seal\": {\n \"authorityRound\": {\n \"step\":
\"0x0\",\n \"signature\":
\"0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000\"\n }\n },\n \"difficulty\": \"0x20000\",\n \"gasLimit\": \"0x2FAF080\"\n },\n
\"accounts\": {\n \"0x0000000000000000000000000000000000000001\": { \"balance\": \"1\", \"builtin\": {
\"name\": \"ecrecover\", \"pricing\": { \"linear\": { \"base\": 3000, \"word\": 0 } } } },\n
\"0x0000000000000000000000000000000000000002\": { \"balance\": \"1\", \"builtin\": { \"name\": \"sha256\",
\"pricing\": { \"linear\": { \"base\": 60, \"word\": 12 } } } },\n
\"0x0000000000000000000000000000000000000003\": { \"balance\": \"1\", \"builtin\": { \"name\": \"ripemd160\",
\"pricing\": { \"linear\": { \"base\": 600, \"word\": 120 } } } },\n
\"0x0000000000000000000000000000000000000004\": { \"balance\": \"1\", \"builtin\": { \"name\": \"identity\",
\"pricing\": { \"linear\": { \"base\": 15, \"word\": 3 } } } },\n
\"0x0000000000000000000000000000000000000006\": { \"balance\": \"0\", \"constructor\" : \"…\" }\n }\n}"
      ]
    },
    "errorMessage": {
      "$id": "/properties/errorMessage",
      "type": "string",
      "title": "Error message",
      "default": "",
      "examples": [
        ""
      ]
    },
    "addressList": {
      "$id": "/properties/addressList",
      "type": "object",
      "properties": {
        "addresses": {
          "$id": "/properties/addressList/properties/addresses",
          "type": "array",
          "items": {
            "$id": "/properties/addressList/properties/addresses/items",
            "type": "string",
            "title": "This member's validator addresses",
            "default": "",
            "examples": [
              "0x00a3cff0dccc0ecb6ae0461045e0e467cff4805f",
              "0x009ce13a7b2532cbd89b2d28cecd75f7cc8c0727"
            ]
          }
        }
      }
    }
  }
}
```


Tutorials

Programmatically interacting with a smart contract

> **WARNING**
>
> Never send your Ethereum private key over the network! Ensure that each transaction is signed locally first and the signed transaction is sent over the network.

In the following example, we use *ethereumjs-wallet* to generate an Ethereum address, *ethereumjs-tx* to sign locally, and *web3* to send the raw transaction to the Ethereum RPC endpoint.

We'll use this simple Hello-World smart contract for this example:

```
pragma solidity ^0.4.11;
contract postBox {
    string message;
    function postMsg(string text) public {
        message = text;
    }
    function getMsg() public view returns (string) {
        return message;
    }
}
```

This example assumes the contract is already deployed. You can use *solc* and *web3* for deploying a contract programmatically. First install the following node modules:

```
sudo npm install web3@0.20.2
sudo npm install ethereumjs-tx@1.3.6
sudo npm install ethereumjs-wallet@0.6.1
```

This nodeJS script will perform the following:

- Construct a raw transaction: postMsg
- Sign the transaction using the generated private key
- Submit the signed transaction to the Ethereum network

```
var ethereumjs = require('ethereumjs-tx')
var wallet = require('ethereumjs-wallet')
var Web3 = require('web3')

// TODO Replace with your contract address
var address = "0xfe53559f5f7a77125039a993e8d5d9c2901edc58";
var abi = [{"constant": false,"inputs": [{"name": "text","type": "string"}],"name": "postMsg","outputs":
[],"payable": false,"stateMutability": "nonpayable","type": "function"},{"constant": true,"inputs": [],"name":
"getMsg","outputs": [{"name": "","type": "string"}],"payable": false,"stateMutability": "view","type":
"function"}];

// Generate a new Ethereum account
var account = wallet.generate();
var accountAddress = account.getAddressString()
var privateKey = account.getPrivateKey();

// TODO Replace with your RPC endpoint
var web3 = new Web3(new Web3.providers.HttpProvider(
    "http://testzvdky-dns-reg1.eastus.cloudapp.azure.com:8545"));

// Get the current nonce of the account
web3.eth.getTransactionCount(accountAddress, function (err, nonce) {
   var data = web3.eth.contract(abi).at(address).postMsg.getData("Hello World");
   var rawTx = {
     nonce: nonce,
     gasPrice: '0x00',
     gasLimit: '0x2FAF080',
     to: address,
     value: '0x00',
     data: data
   }
   var tx = new ethereumjs(rawTx);

   tx.sign(privateKey);

   var raw = '0x' + tx.serialize().toString('hex');
   web3.eth.sendRawTransaction(raw, function (txErr, transactionHash) {
     console.log("TX Hash: " + transactionHash);
     console.log("Error: " + txErr);
   });
 });
```

Deploy smart contract with Truffle

- Install necessary libraries

```
npm init

npm install truffle-hdwallet-provider --save
```

- In truffle.js, add following code to unlock your MetaMask account and configure the PoA node as entry point by providing the mnemonic phrase (MetaMask / Settings / Reveal Seed Words)

```
var HDWalletProvider = require("truffle-hdwallet-provider");

var rpc_endpoint = "XXXXXX";
var mnemonic = "twelve words you can find in metamask/settings/reveal seed words";

module.exports = {
  networks: {
    development: {
      host: "localhost",
      port: 8545,
      network_id: "*" // Match any network id
    },
    poa: {
      provider: new HDWalletProvider(mnemonic, rpc_endpoint),
      network_id: 3,
      gasPrice : 0
    }
  }
};
```

- Deploy to PoA network

```
$ truffle migrate --network poa
```

Debug smart contract with Truffle

Truffle has a local develop network that is available for debugging smart contract. You can find the full tutorial here.

WebAssembly (WASM) support

WebAssembly support is already enabled for you on newly deployed PoA networks. It allows for smart-contract development in any language that transpiles to Web-Assembly (Rust, C, C++). See the links below for additional information

- Parity Overview of WebAssembly - https://wiki.parity.io/WebAssembly-Home
- Tutorial from Parity Tech - https://github.com/paritytech/pwasm-tutorial

Reference

FAQ

I notice there are many transactions on the network that I didn't send. Where are these coming from?

It is insecure to unlock the personal API. Bots listen for unlocked Ethereum accounts and attempt to drain the funds. The bot assumes these accounts contain real-ether and attempt to be the first to siphon the balance. Do not enable the personal API on the network. Instead pre-sign the transactions either manually using a wallet like MetaMask or programmatically as outlined in the section Programmatically Interacting with a Smart Contract.

How to SSH onto a VM?

The SSH port is not exposed for security reasons. Follow this guide to enable the SSH port.

How do I set up an audit member or transaction nodes?

Transaction nodes are a set of Parity clients that are peered with the network but are not participating in consensus. These nodes can still be used to submit Ethereum transactions and read the smart contract state. This works well as a mechanism for providing auditability to non-authority consortium members on the network. To achieve this simply follow Step 2 from Growing the Consortium.

Why are MetaMask transactions taking a long time?

To ensure transactions are received in the correct order, each Ethereum transaction comes with an incrementing nonce. If you've used an account in MetaMask on a different network, you'll need to reset the nonce value. Click on

the settings icon (3-bars), Settings, Reset Account. The transaction history will be cleared and now you can resubmit the transaction.

Do I need to specify gas fee in MetaMask?

Ether doesn't serve a purpose in proof-of-authority consortium. Hence there is no need to specify gas fee when submitting transactions in MetaMask.

What should I do if my deployment fails due to failure to provision Azure OMS?

Monitoring is an optional feature. In some rare cases where your deployment fails because of inability to successfully provision Azure Monitor resource you can redeploy without Azure Monitor.

Are public IP deployments compatible with private network deployments?

No, peering requires two-way communication so the entire network must either be public or private.

What is the expected transaction throughput of Proof-of-Authority?

The transaction throughput will be highly dependent upon the types of transactions and the network topology. Using simple transactions, we've benchmarked an average of 400 transactions per second with a network deployed across multiple regions.

How do I subscribe to smart contract events?

Ethereum Proof-of-Authority now supports web-sockets. Check your deployment email or deployment output to locate the web-socket URL and port.

Next steps

Get started by using the Ethereum Proof-of-Authority Consortium solution.

Hyperledger Fabric consortium network

5/9/2019 • 6 minutes to read • Edit Online

You can use the Hyperledger Fabric consortium solution template to deploy and configure a Hyperledger Fabric consortium network on Azure.

After reading this article, you will:

- Obtain working knowledge of blockchain, Hyperledger Fabric, and more complicated consortium network architectures
- Learn how to deploy and configure a Hyperledger Fabric consortium network from within the Azure portal

About blockchain

If you are new to the blockchain community, this solution template is a great opportunity to learn about the technology in an easy and configurable manner on Azure. Blockchain is the underlying technology behind Bitcoin; however, it is much more than just an enabler for a virtual currency. It is a composite of existing database, distributed system, and cryptographic technologies that enables secure multi-party computation with guarantees around immutability, verifiability, auditability, and resiliency to attack. Different protocols employ different mechanisms to provide these attributes. Hyperledger Fabric is one such protocol.

Consortium architecture on Azure

To enable Hyperledger Fabric in Azure, there are two primary deployment types that are supported. These deployments are designed to accommodate different topologies, based on desired target.

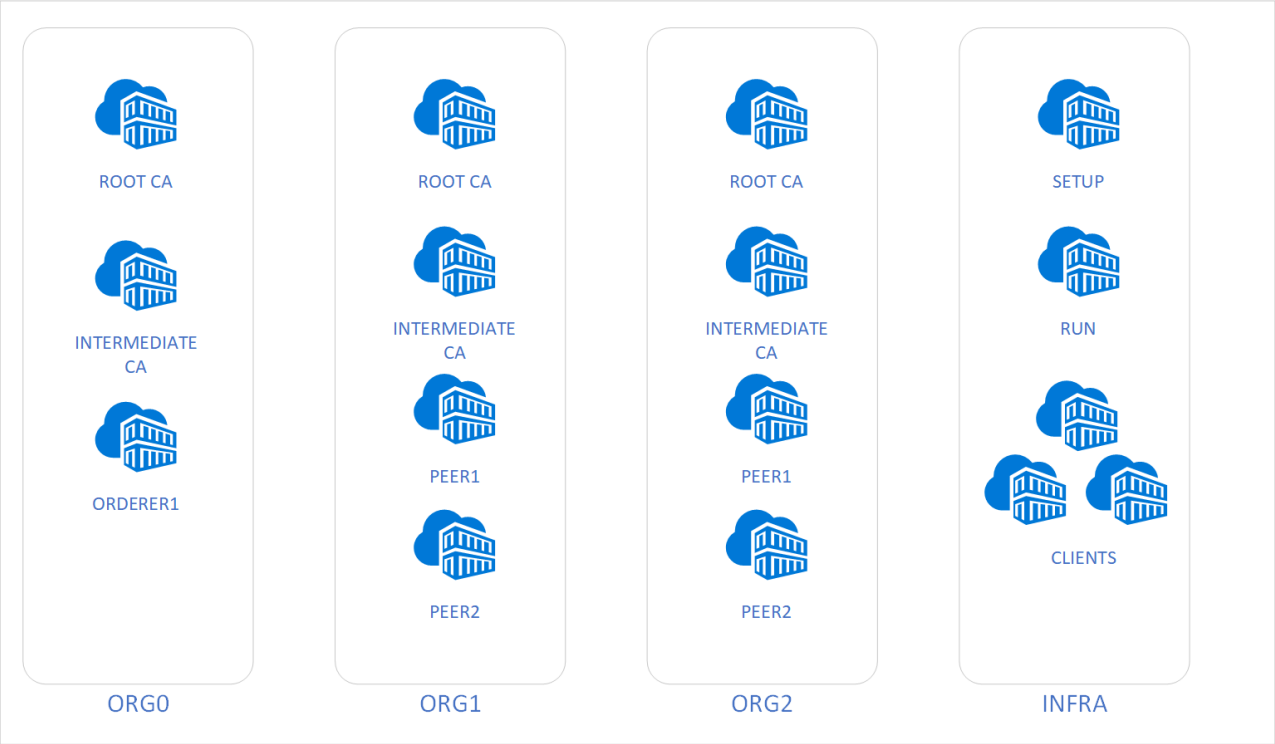
- **Single virtual machine, developer server** - This deployment type is designed as a development environment used to build and test solutions built on Hyperledger Fabric.
- **Multiple virtual machines, scale out deployment** - This deployment type is designed for environments that model a consortium of different participants leveraging a shared environment.

In either deployment, the building blocks that are make the core of Hyperledger Fabric are the same. The differences in the deployments are how these components are scaled out.

- **CA nodes**: A node running Certificate Authority that is used to generate certificates that are used for identities in the network.
- **Orderer nodes**: A node running the communication service implementing a delivery guarantee, such as total order broadcast or atomic transactions.
- **Peer nodes**: A node that commits transactions and maintains the state and a copy of the distributed ledger.
- **CouchDB nodes**: A node that can run the CouchDB service that can hold the state database and provide rich querying of chaincode data, expanding from simple key/value to JSON type storage.

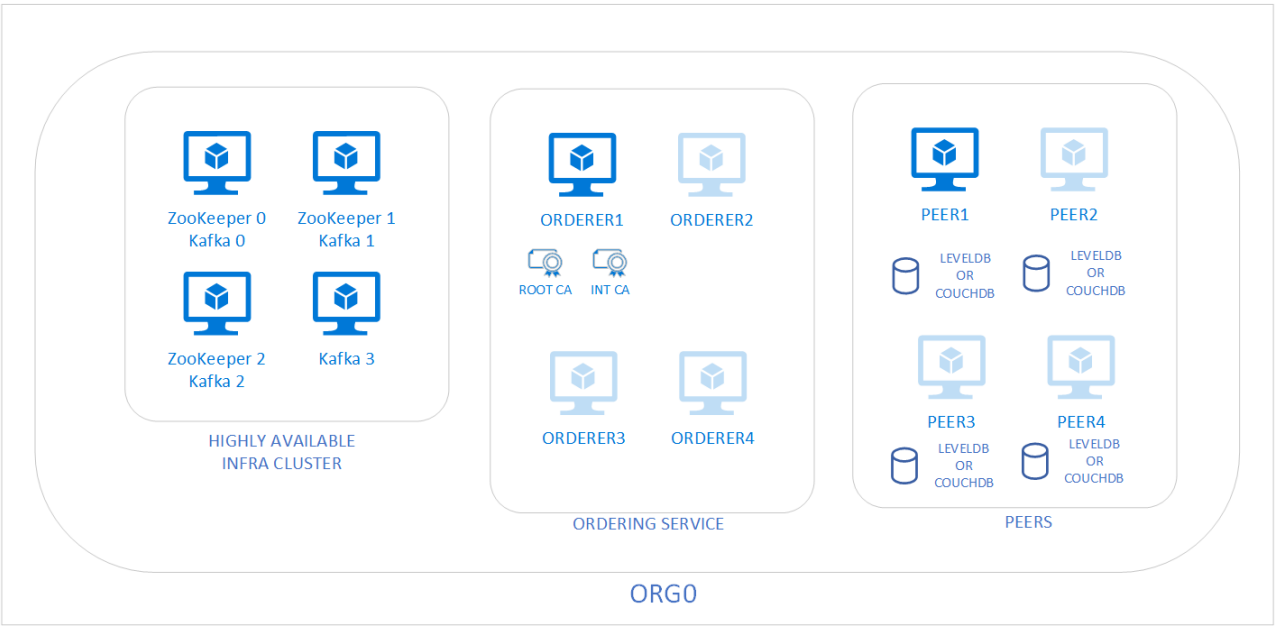
Single virtual machine architecture

As mentioned previously the single virtual machine architecture is built for developers to have a low footprint server that is used to develop applications. All containers shown are running in a single virtual machine. The ordering service is using SOLO for this configuration. This configuration is *not* a fault tolerant ordering service, but is designed to be lightweight for development purposes.



Multiple virtual machine architecture

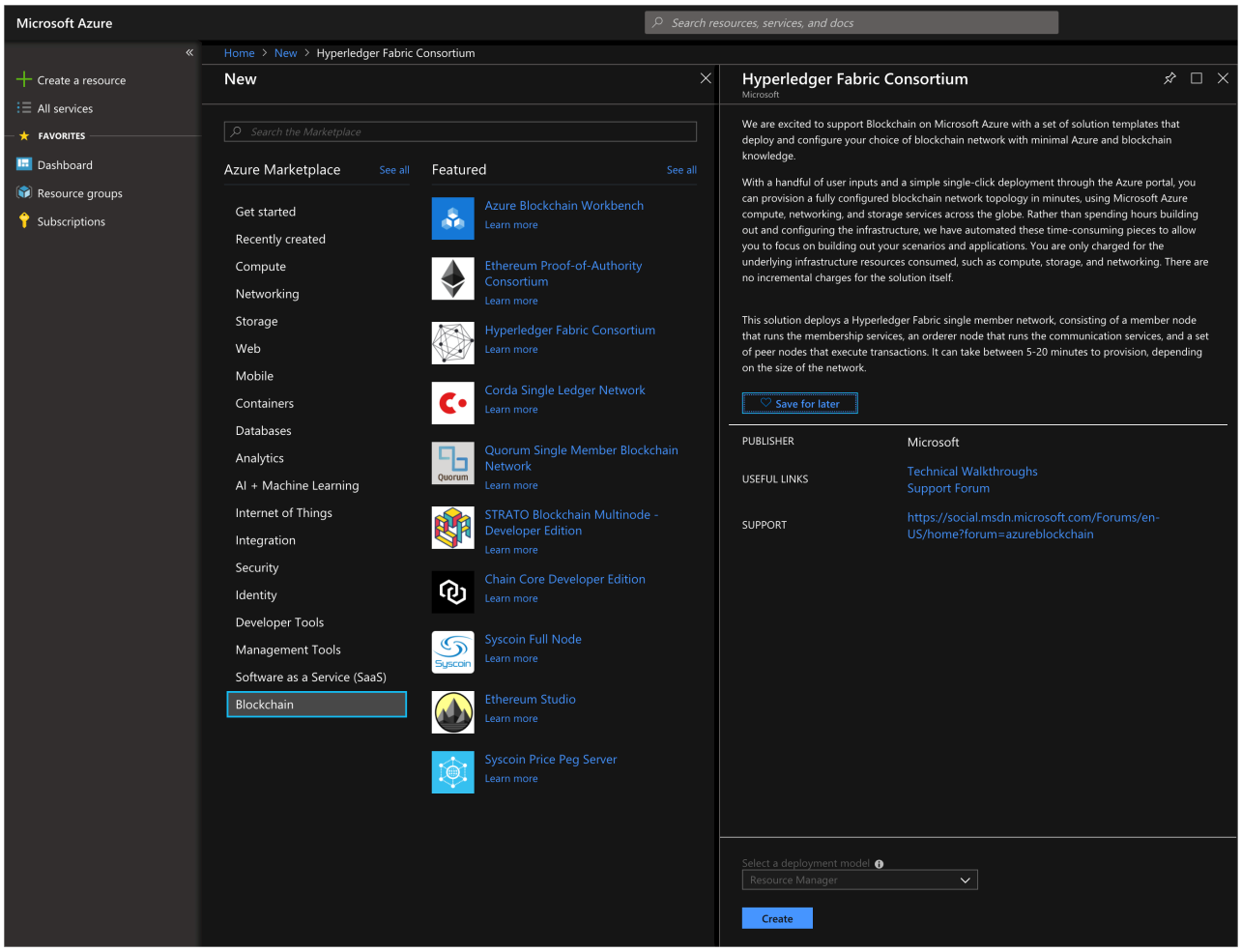
The multiple virtual machine, scale-out architecture, is built with high availability and scaling of each component at the core. This architecture is much more suitable for production grade deployments.



Getting started

To begin, you need an Azure subscription that can support deploying several virtual machines and standard storage accounts. If you do not have an Azure subscription, you can create a free Azure account.

Once you have a subscription, go to the Azure portal. Select **Create a resource > Blockchain > Hyperledger Fabric Consortium**.



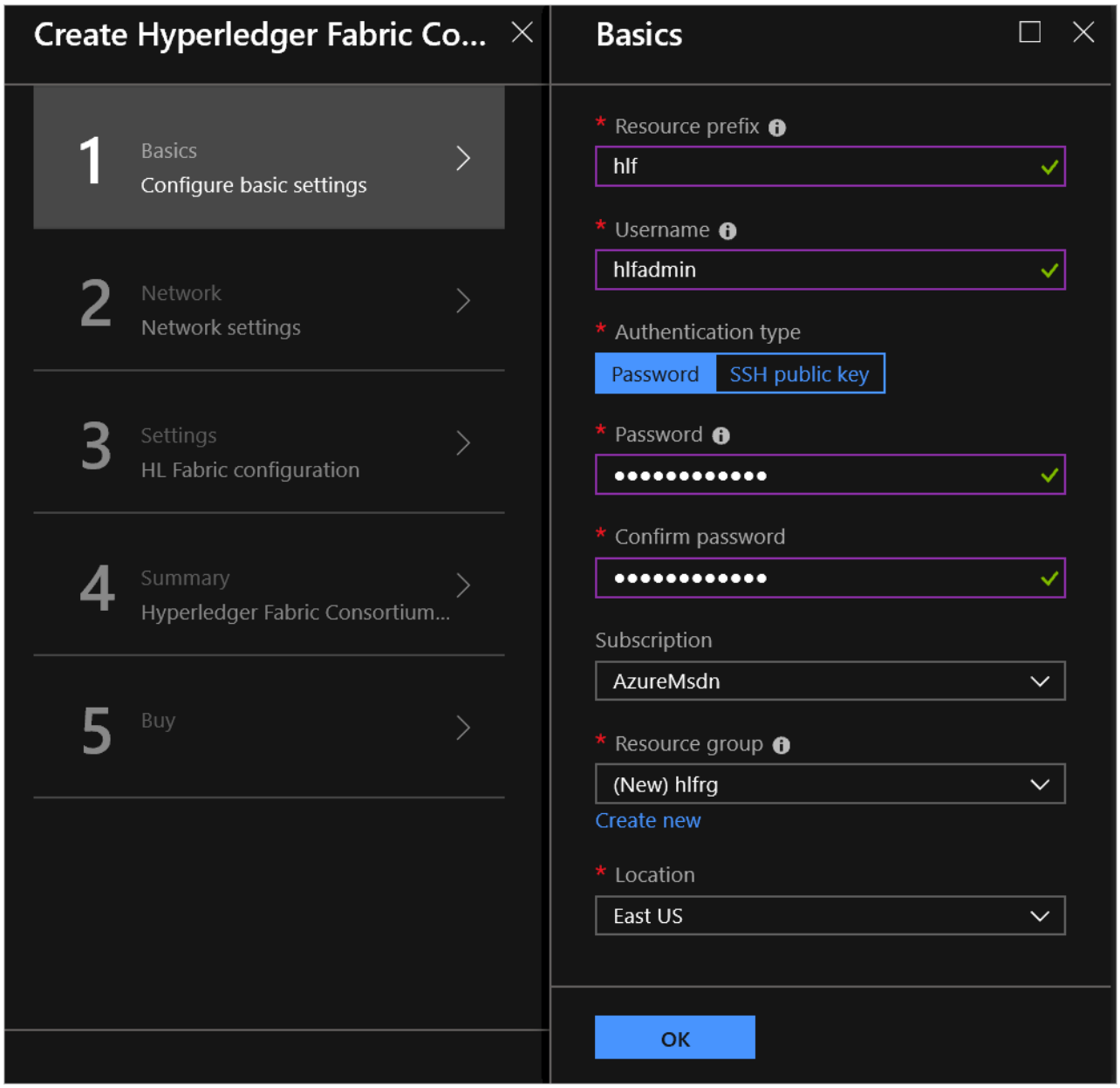
Deployment

In the **Hyperledger Fabric Consortium** template, select **Create**.

The template deployment will walk you through configuring the multi-node Hyperledger 1.3 network. The deployment flow is divided into four steps: Basics, Consortium Network Settings, Fabric configuration, and Optional components.

Basics

In **Basics**, specify values for standard parameters for any deployment. Such as, subscription, resource group, and basic virtual machine properties.



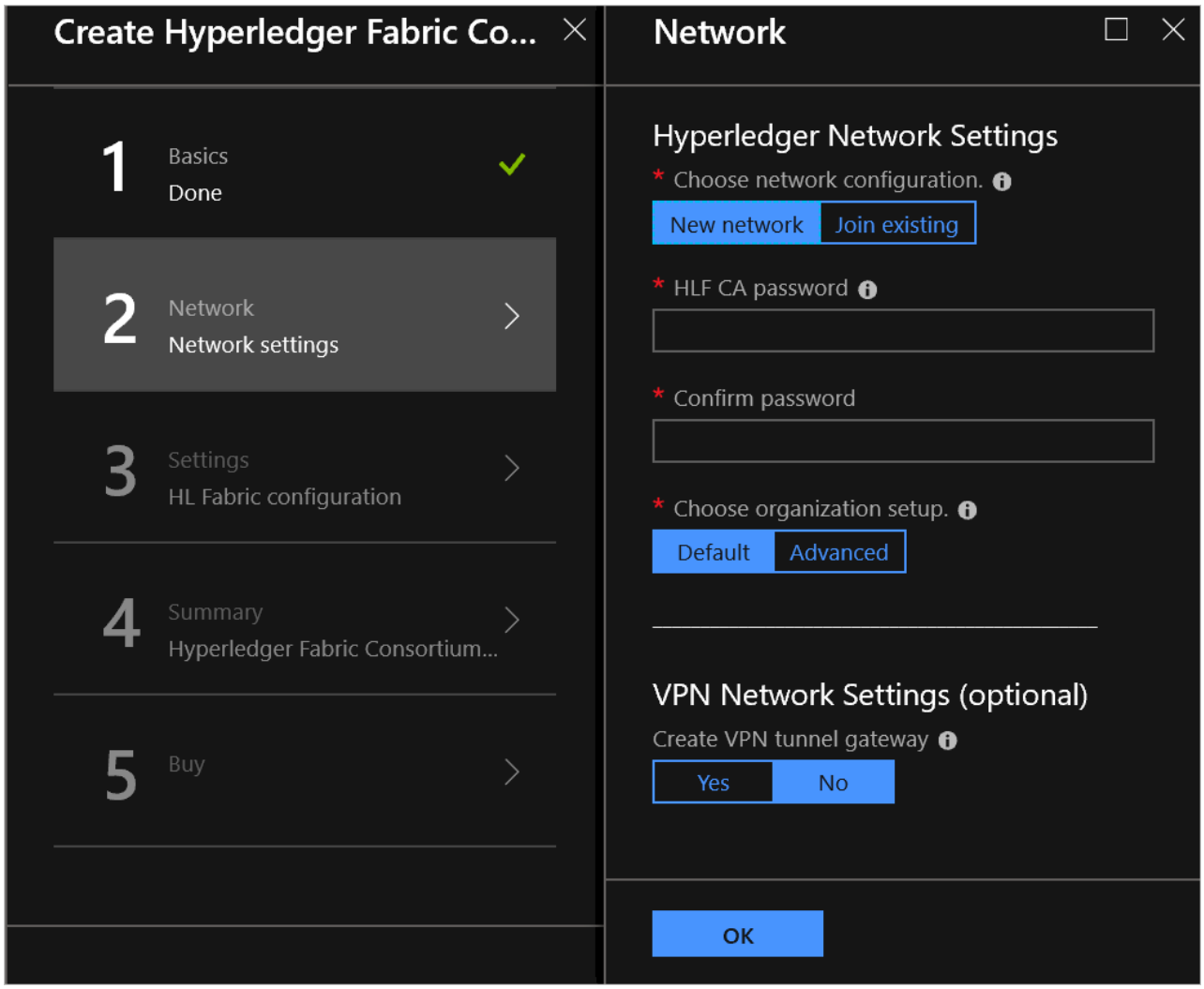
PARAMETER NAME	DESCRIPTION	ALLOWED VALUES
Resource prefix	Name prefix for resources provisioned as part of the deployment	6 characters or less
Username	The user name of the administrator for each of the virtual machines deployed for this member	1 - 64 characters
Authentication type	The method to authenticate to the virtual machine	Password or SSH public key

PARAMETER NAME	DESCRIPTION	ALLOWED VALUES
Password (Authentication type = Password)	The password for the administrator account for each of the virtual machines deployed. The password must contain three of the following character types: 1 upper case character, 1 lower case character, 1 number, and 1 special character While all VMs initially have the same password, you can change the password after provisioning	12 - 72 characters
SSH key (Authentication type = SSH public key)	The secure shell key used for remote login	
Subscription	The subscription to which to deploy	
Resource group	The resource group to which to deploy the consortium network	
Location	The Azure region to which to deploy the first member in	

Select **OK**.

Consortium Network Settings

In **Network settings**, specify inputs for creating or joining an existing consortium network and configure your organization settings.

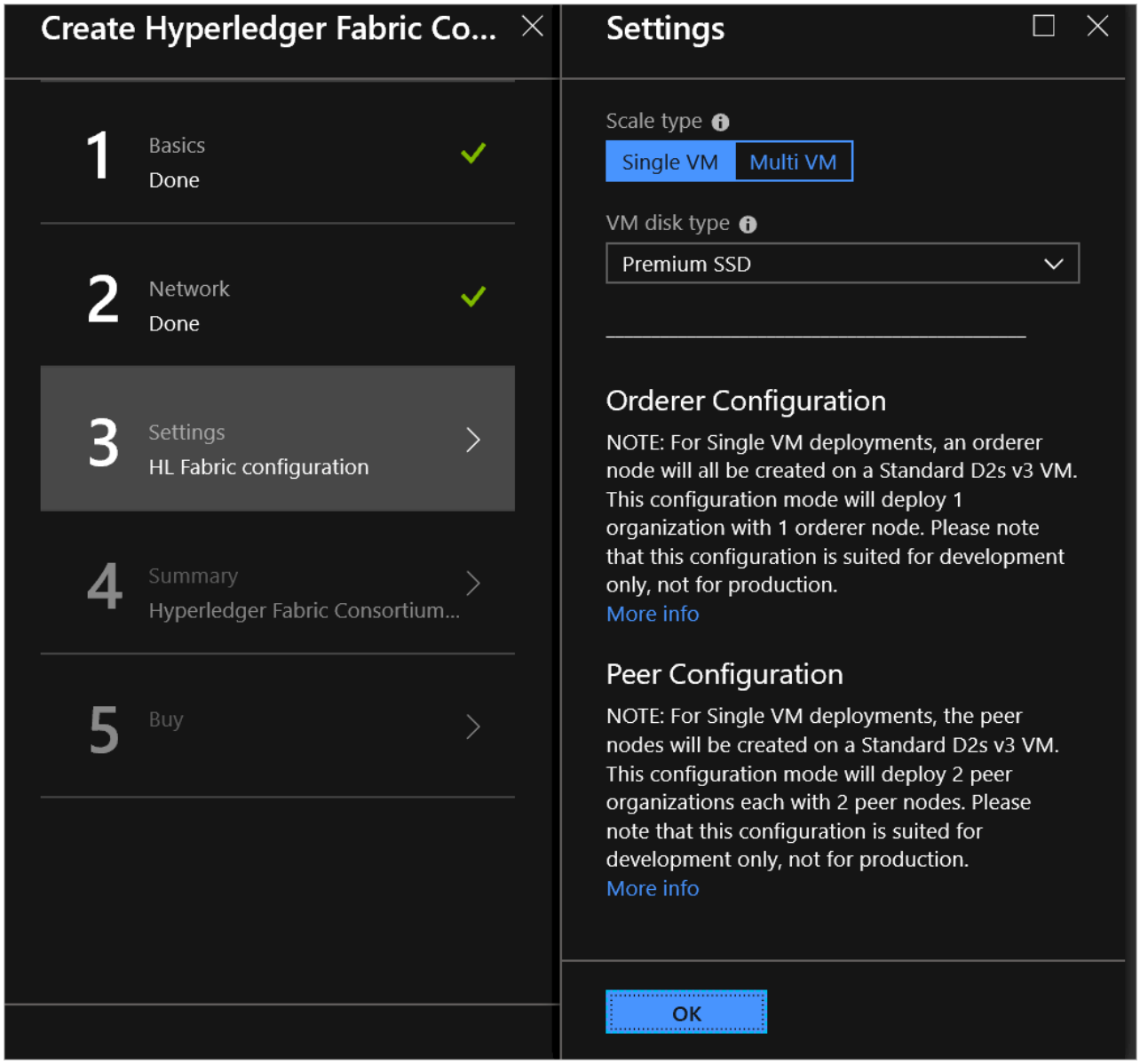


PARAMETER NAME	DESCRIPTION	ALLOWED VALUES
Network configuration	You can choose to create a new network or join an existing one. If you choose *Join existing*, you need to provide additional values.	New network Join existing
HLF CA password	A password used for the certificates generated by the certificate authorities that are created as part of the deployment. The password must contain three of the following character types: 1 upper case character, 1 lower case character, 1 number, and 1 special character. While all virtual machines initially have the same password, you can change the password after provisioning.	1 - 25 characters
Organization setup	You can customize your Organization's name and certificate or have default values to be used.	Default Advanced
VPN network settings	Provision a VPN tunnel gateway for accessing the VMs	Yes No

Select **OK**.

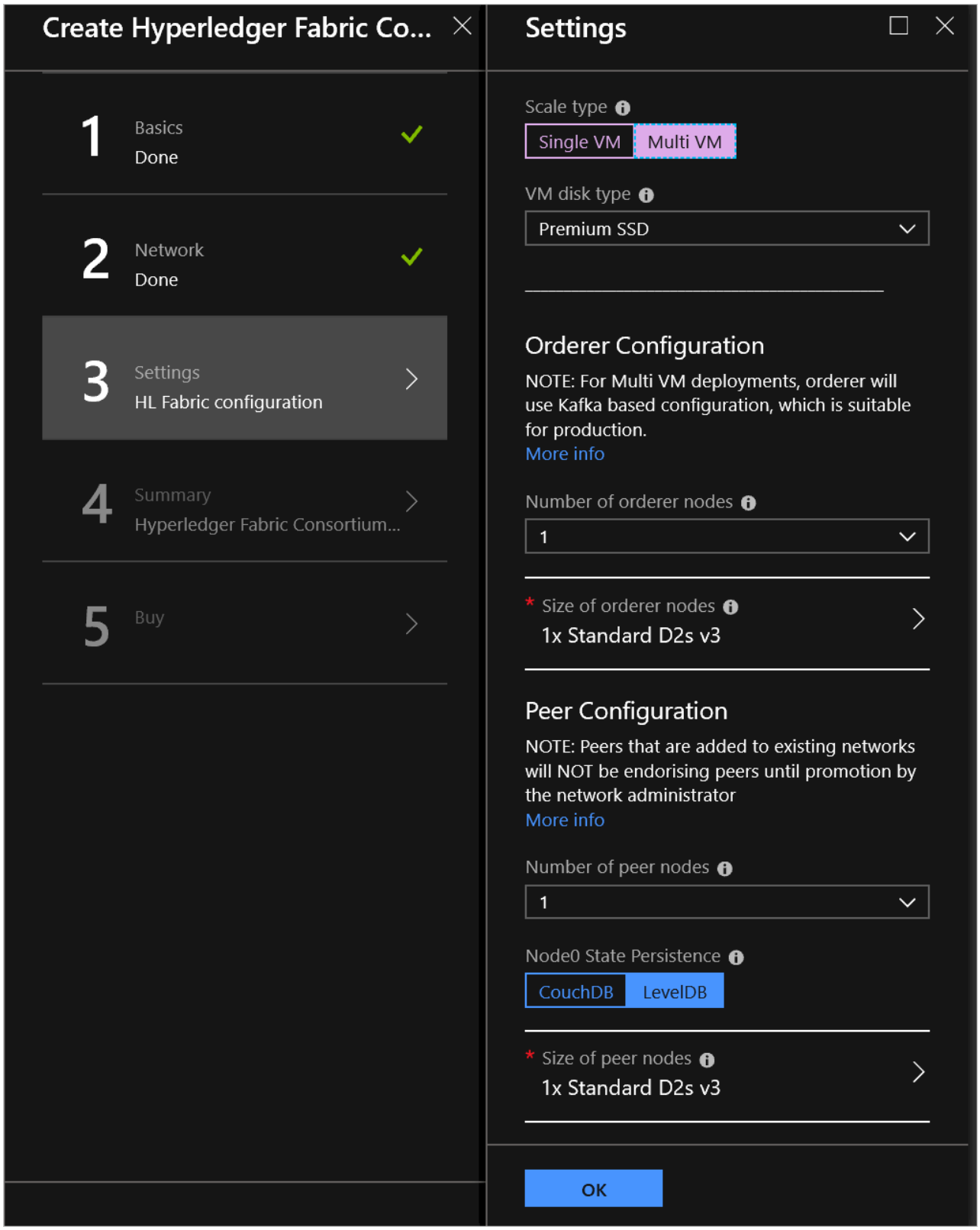
Fabric-specific settings

In **Fabric configuration**, you configure network size and performance, and specify inputs for the availability of the network. Such as, number orderer and peer nodes, persistence engine used by each node, and the VM size.



PARAMETER NAME	DESCRIPTION	ALLOWED VALUES
Scale type	The deployment type of either a single virtual machine with multiple containers or multiple virtual machines in a scale-out model.	Single VM or Multi VM
VM Disk type	The type of storage backing each of the deployed nodes. To learn more about the available disk types, visit select a disk type.	Standard SSD Premium SSD

Multiple VM deployment (additional settings)

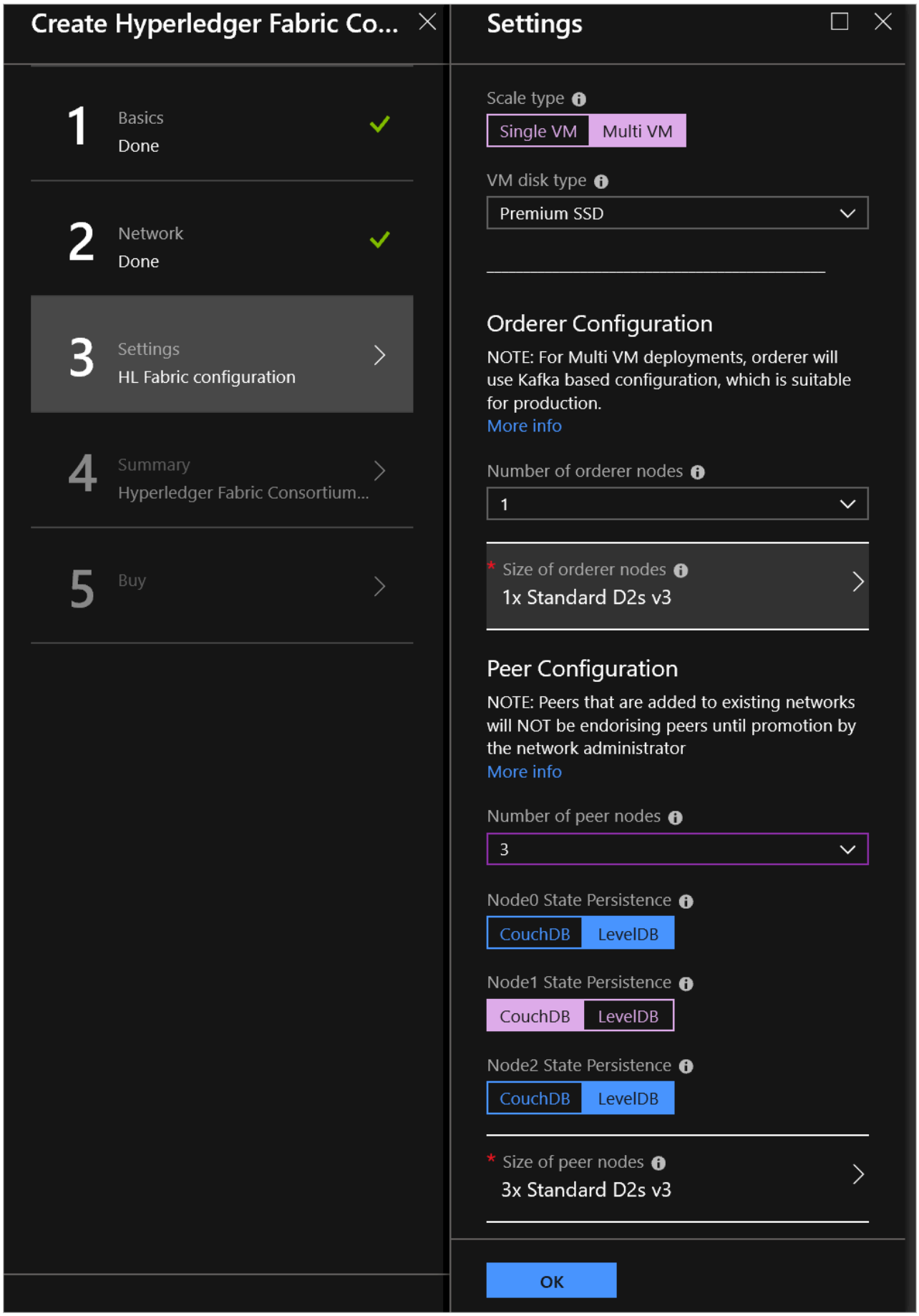


PARAMETER NAME	DESCRIPTION	ALLOWED VALUES
Number of orderer nodes	The number of nodes that order (organize) transactions into a block. For additional details on the ordering service, visit the Hyperledger documentation	1-4
Orderer node virtual machine size	The virtual machine size used for orderer nodes in the network	Standard Bs, Standard Ds, Standard FS

PARAMETER NAME	DESCRIPTION	ALLOWED VALUES
Number of peer nodes	Nodes that are owned by consortium members that execute transactions and maintain the state and a copy of the ledger. For additional details on the ordering service, visit the Hyperledger documentation.	1-4
Node state persistence	The persistence engine used by the peer nodes. You can configure this engine per peer node. See details below for multiple peer nodes.	CouchDB LevelDB
Peer node virtual machine size	The virtual machine size used for all nodes in the network	Standard Bs, Standard Ds, Standard FS

Multiple peer node configuration

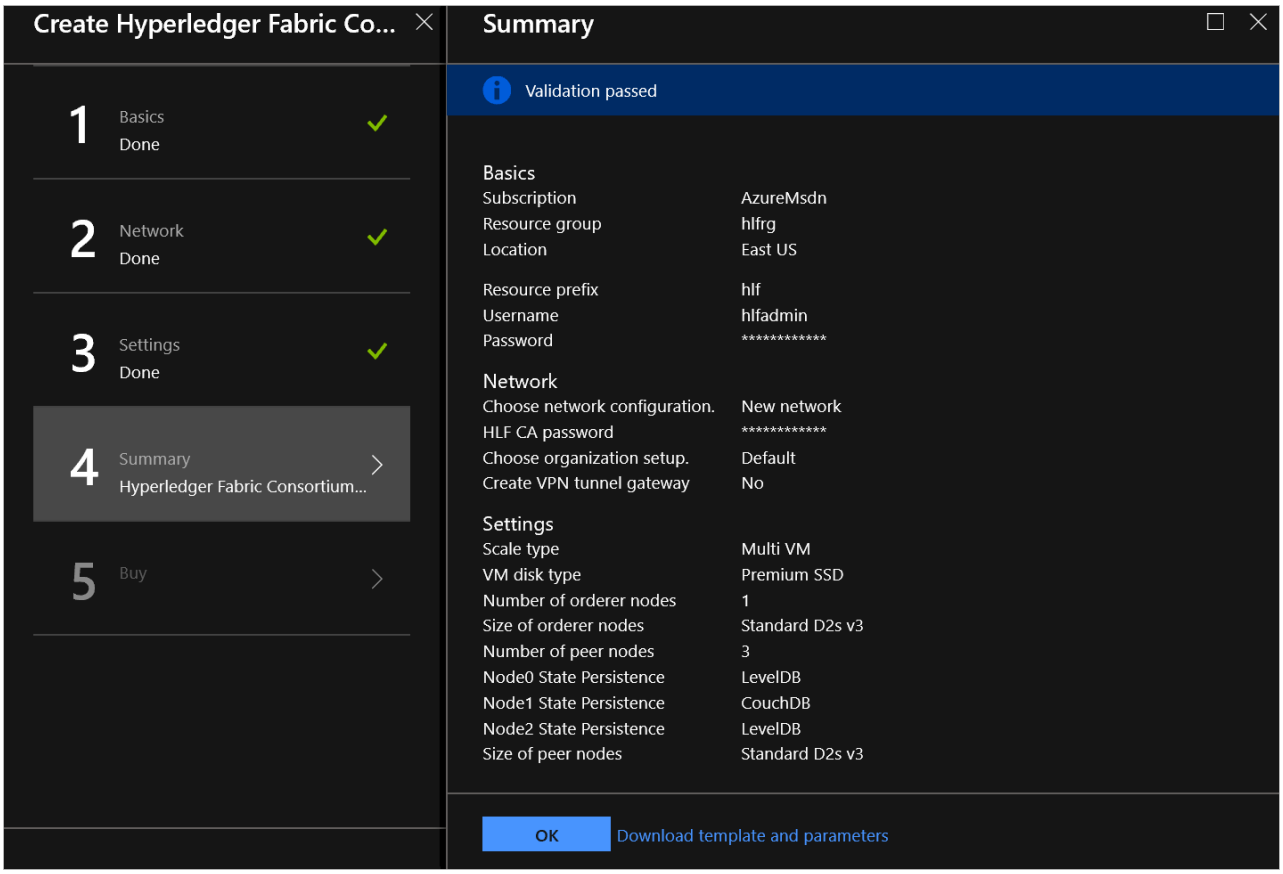
This template allows you to pick your persistence engine per peer node. For example, if you have three peer nodes you can use CouchDB on one and LevelDB on the other two.



Select **OK**.

Deploy

In **Summary**, review the inputs specified and to run basic pre-deployment validation.



Review legal and privacy terms and select **Purchase** to deploy. Depending on the number of VMs being provisioned, deployment time can vary from a few minutes to tens of minutes.

Next steps

You are now ready to focus on application and chaincode development against your Hyperledger consortium blockchain network.