Microsoft

Gaining Expertise with Azure Blockchain Workbench

A guide for developers

Version 1.0, June 2019

For the latest information on the Blockchain solutions on Azure, please see
https://azure.microsoft.com/en-us/solutions/blockchain/
and follow @MSFTBlockchain on Twitter

If you have a specific request, you can request assistance on
https://aka.ms/blockchainrequests

This page is intentionally left blank.

Table of contents

Notice

This guide for developers is intended to illustrate the Azure Blockchain Service "basics" to allow companies of all size to build upon solutions for their shared business processes. It is part of a series of guides to help developers gaining expertise on the Microsoft blockchain services available in Azure. This series can be seen as an addition to the Azure Blockchain Development Kit already available at https://azure.microsoft.com/en-us/resources/samples/blockchain-devkit/.

Publication: June 2019
Version 1.0

About this guide

Welcome to the **Gaining Expertise with Azure Blockchain Service** guide for developers.

Last May 2019, has been publicly released in preview the first version of Azure Blockchain Service[1] (ABS). This brand-new service allows users to deploy a fully managed blockchain, based on open-sourced Ethereum Quorum[2] ledger (i.e. the enterprise version of Ethereum ledger, and one of the most famous blockchain).

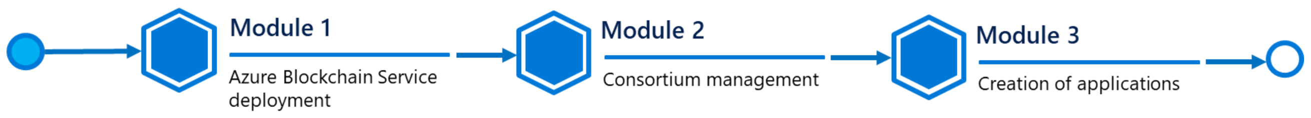
One of the best features that comes along with Azure Blockchain Service is probably the consortium management. This will allow companies of all size to work together when having to share among them workflows and businesses. Fully compatible with others Azure services, as well as natively integrated with Azure Blockchain Workbench[3], Azure Blockchain Service can be leveraged for prototyping new kind of (decentralized) applications for production-ready ones.

One should note that Microsoft also released last spring development tools on Visual Studio Code to help developers creating blockchain applications with Truffle, a full-pledged blockchain framework.

In this guide, and as its title suggest, we will cover Azure Blockchain Service key features and you will learn how to create and deploy new kind of applications on top of this services: i.e. decentralized and shared ones.

For that purposes, you're invited to follow a series of modules, each of them illustrating a specific aspect of Azure Blockchain Service.

Each module within the guide builds on the previous. You're free to stop at any module you want, but our advice is to go through all the modules.



At the end of the guide, you will be able to:

- Deploy Azure Blockchain Service and understand its key features and capabilities.
- Manage the underlying consortium of Azure Blockchain Service.
- Create new (decentralized) applications, using the Remix IDE or Visual Studio Code.
- Integrate multiple Azure services against Azure Blockchain Service (e.g. Azure Logic Apps[4], Azure App Service[5] Web app, etc.).

[1] Azure Blockchain Service: https://azure.microsoft.com/en-us/services/blockchain-service/

[2] Quorum: https://www.jpmorgan.com/Quorum

[3] Azure Blockchain Workbench: https://azure.microsoft.com/en-us/features/blockchain-workbench/

[4] Azure Logic Apps: https://azure.microsoft.com/en-us/services/logic-apps/

[5] Azure App Service: https://azure.microsoft.com/en-us/services/app-service/

Guide elements

In the guide modules you will see the following elements:

- **Step-by-step directions**. Click-through instructions - along with relevant snapshots - or links to online documentation for completing each procedure or part.
- **Important concepts**. An explanation of some of the concepts important to the procedures in the module, and what happens behind the scenes.
- **Sample applications, and files**. A downloadable version of the project containing the code that you will use in this guide, and other files you will need. **Please go to https://aka.ms/ABSDevGuideSamples to download all necessary assets.**

Guide prerequisites

To successfully leverage the provided code in this guide, you will need:

- A Microsoft account[6].
- An Azure subscription. If you don't have an Azure subscription, create a free account[7] before you begin.
- A code editor of your choice (such as Visual Studio[8] or Visual Studio Code[9]).
- Node.js[10] installed on your computer (10.15 or later).
- Any terminal, like Git[11] (2.10 or later) or PowerShell. You will see both of those terminals used in this guide.
- Python 2[12] (2.7.15 or later).

With your chosen terminal, you will also have to install additional packages:

- npm[13] (6.4.1 or later).
- Web3[14] (1.0 or later).
- Truffle[15] (5.0.0 or later).
- Ganache CLI[16] (6.0.0 or later)

[6] Microsoft Account: https://account.microsoft.com/account?lang=en-us

[7] Create your Azure free account today: https://azure.microsoft.com/en-us/free/?WT.mc_id=A261C142F

[8] Visual Studio: https://visualstudio.microsoft.com/

[9] Visual Studio Code: https://code.visualstudio.com/

[10] Node.js: https://nodejs.org/en/

[11] Git: https://git-scm.com/downloads

[12] Python: https://www.python.org/downloads/release/python-2715/

[13] npm: https://www.npmjs.com/get-npm

[14] Web3: https://github.com/ethereum/web3.js/

[15] Truffle: https://www.trufflesuite.com/truffle

[16] Ganache CLI: https://github.com/trufflesuite/ganache-cli

Module 1: Azure Blockchain Service deployment

Overview

This first module of this guide will help you understand the key features and capabilities of Azure Blockchain Service and then show you how to deploy Azure Blockchain Service in your subscription.

Overview
Important concepts
Step-by-steps directions: deployment
Module 1
Azure Blockchain Service deployment
Module 2
Consortium management
Module 3
Creation of applications

Let's with key features and capabilities before deploying Azure Blockchain Service and members. As you will see, the deployment part will be quite straightforward because you'll only have one form to fill in to get your Azure Blockchain Service instance deployed.

Important concepts

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

Indeed, thanks to a unified control for both infrastructure management as well as blockchain network governance, Azure Blockchain Service provides:

- Simple network deployment and operations.
- Built-in consortium management.

These capabilities require almost no administration and allow you to focus on app development and business logic rather than allocating time and resources to manage virtual machines and the underlying infrastructure. You can develop your smart contracts on top of it with your familiar development tools such as Visual Studio Code

If Azure Blockchain Service is designed from the ground to support multiple ledger protocols, you should note that, as of this writing, it provides support (only) for the Ethereum Quorum ledger using the IBFT[17] consensus mechanism.

The main features of Azure Blockchain Service are the followings.

[17] Quorum Consensus: https://github.com/jpmorganchase/quorum/wiki/Quorum-Consensus

Note You can get more information by reading the full Azure Blockchain Service documentation here[18].

Consortium management

When deploying your first blockchain member, you will either join or create a consortium.

As such, a consortium is a logical group used to manage the governance between blockchain members who transact in a multi-party process. Every company can join an existing consortium by having its Azure subscription invited inside it. When invited, a company just have to create an instance of Azure Blockchain Service and select the consortium to join it.

Monitoring and logging

In addition, Azure Blockchain Service provides rich metrics through Azure Monitor[19] service providing insights into nodes' CPU, memory and storage usage, as well as helpful insights into blockchain network. Metrics can be customized to provide views into the insights that are important to your blockchain application.

In addition, thresholds can be defined through alerts enabling users to trigger actions such as sending an email or text message, running an Azure Logic Apps, Azure Functions[20] or sending to a custom-defined webhook.

Security and maintenance

After provisioning your first blockchain member, you can add additional transaction nodes to your member.

By default, transaction nodes are secured through firewall rules and will need to be configured for access. Additionally, all transaction nodes encrypt data in motion via TLS. Multiple options exist for securing transaction node access, including firewall rules, basic authentication, access keys as well as Azure Active Directory[21] (Azure AD) integration.

As a managed service, Azure Blockchain Service ensures that your blockchain member's nodes are patched with the latest host operating system (OS) and ledger software stack updates, configured for high-availability (Standard tier only), eliminating much of the Dev(Sec)Ops required for traditional IaaS (Infrastructure-as-a-Service) blockchain nodes.

Develop using familiar development tools

Based on the open-sourced Quorum Ethereum ledger, you can develop decentralized applications (ĐApp) for Azure Blockchain Service the same way as you do for existing Ethereum applications.

Working with leading industry partners, the dedicated Visual Studio Code extension[22] allows developers to leverage familiar tools such as the aforementioned Truffle Suite to build smart contracts. Using the extension, developers can create, or connect to and existing consortium so that you can build and deploy your smart contracts all from one IDE.

[18] WHAT IS AZURE BLOCKCHAIN SERVICE?: https://docs.microsoft.com/en-us/azure/blockchain/service/overview

[19] Azure Monitor: https://azure.microsoft.com/en-us/services/monitor/

[20] Azure Functions: https://azure.microsoft.com/en-us/services/functions/

[21] Azure Active Directory: https://azure.microsoft.com/en-us/services/active-directory/

[22] Azure Blockchain Development Kit for Ethereum: https://marketplace.visualstudio.com/items?itemName=AzBlockchain.azure-blockchain

Step-by-step directions

This module covers the following operation: Deploying Azure Blockchain Service.

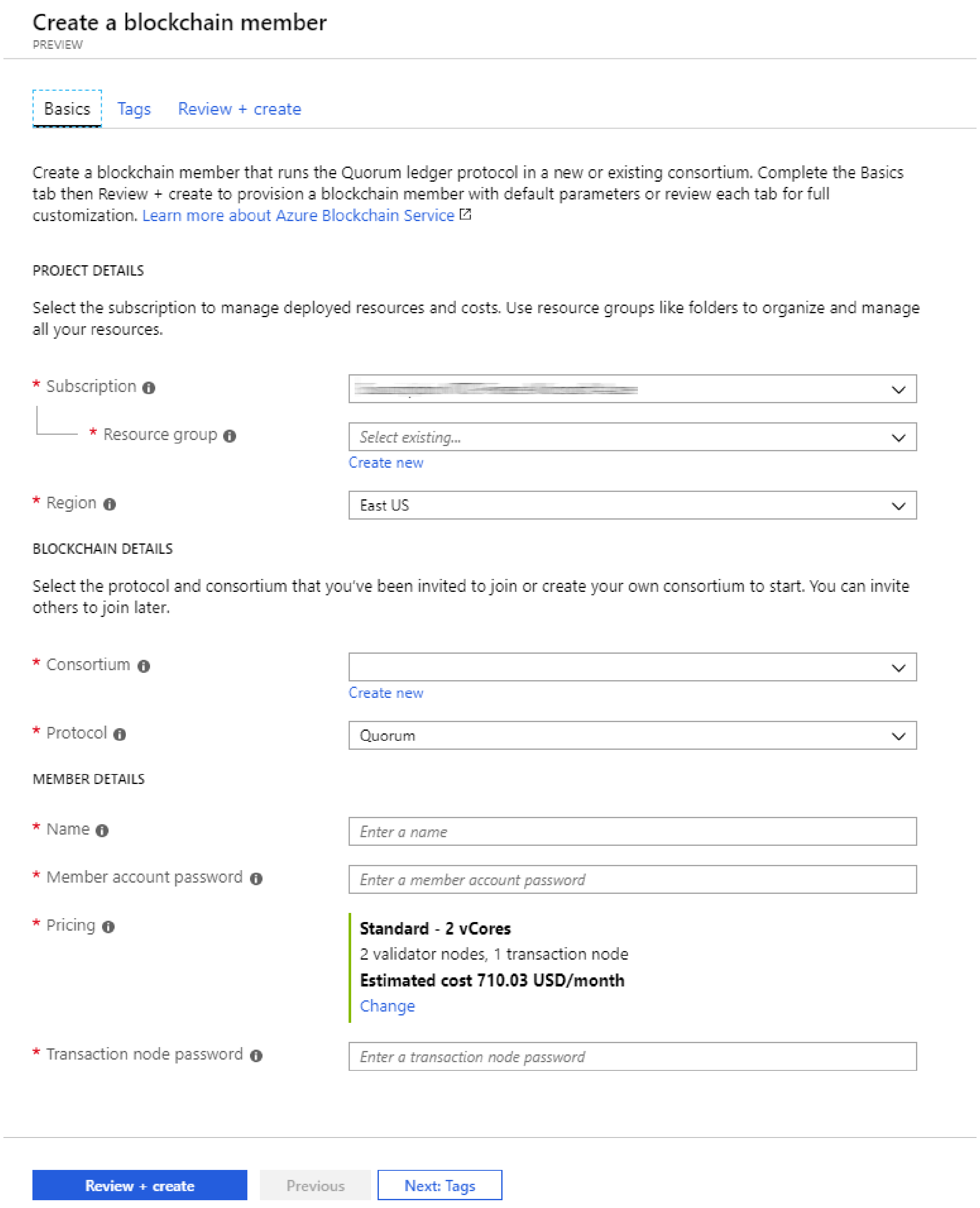
Deploying Azure Blockchain Service

The first step is to sign-in into the Azure portal with your account to create an Azure Blockchain Service instance.

Perform the following steps:

1. Sign-in to the Azure portal[23] with your (Microsoft) account.
2. Optionally switch, in the top-right corner, to the desired Azure AD tenant where you want to deploy Azure Blockchain Service.
3. In the left pane, select **Create a resource**. Search for "*Azure Blockchain Service*" in the **Search the Marketplace** search bar.
4. Select **Azure Blockchain Service**.
5. Select **Create**.
6. Complete the **Basics** settings.

[23] Azure portal: https://portal.azure.com



Create a blockchain member

PREVIEW

Basics | Tags | Review + create

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

PROJECT DETAILS

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

* Subscription	[redacted]
* Resource group	*Select existing...* Create new
* Region	East US

BLOCKCHAIN DETAILS

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

* Consortium	 Create new
* Protocol	Quorum

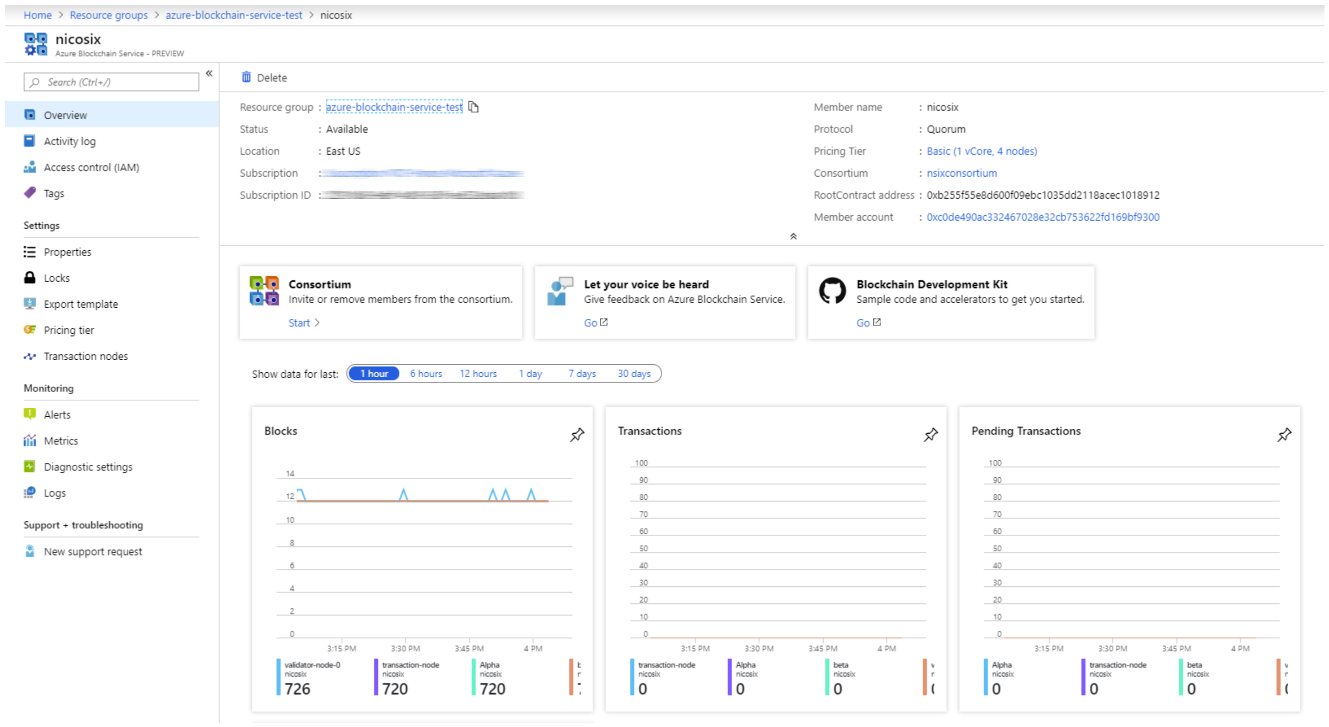
MEMBER DETAILS

* Name	*Enter a name*
* Member account password	*Enter a member account password*
* Pricing	**Standard - 2 vCores** 2 validator nodes, 1 transaction node **Estimated cost 710.03 USD/month** Change
* Transaction node password	*Enter a transaction node password*

Setting	**Description**
Subscription	Select the Azure subscription that you want to use for your instance of Azure Blockchain Service. If you have multiple subscriptions, choose the subscription in which you'd like to get billed for this resource.
Resource group	Specify a new resource group name or an existing one from your subscription.
Region	Set the location. It must be the same for all members of the consortium.
Consortium	Pick the consortium you want to join or click on **Create new** if you want to create a new one.
Protocol	Pick the blockchain protocol, **Quorum** is selected by default.
Name	Choose a unique name that identifies your Azure Blockchain Service member. The blockchain member name can only contain lowercase letters and numbers. The first character must be a letter. The value must be between 2 and 20 characters long.
Member account password	Specify the member account password. The password is used to encrypt the private key for the Ethereum account that is created for your member. You will use the member account and member account password for consortium management
Pricing	Select the node configuration for your new service. For this guide, Basic is enough.
Transaction node password	Specify the password for the member's default transaction node. Use the password for basic authentication when connecting to blockchain member's default transaction node public endpoint.

7. Click on **Review + Create**, and then wait for the deployment (it usually takes 1 hour).

Once completed, your Azure Blockchain Service is now live, and you can access to your member by clicking on it inside your newly created resource group as illustrated hereafter.


Home > Resource groups > azure-blockchain-service-test > nicosix

nicosix
Azure Blockchain Service - PREVIEW

Delete

Resource group : azure-blockchain-service-test
Status : Available
Location : East US
Subscription : [redacted]
Subscription ID : [redacted]

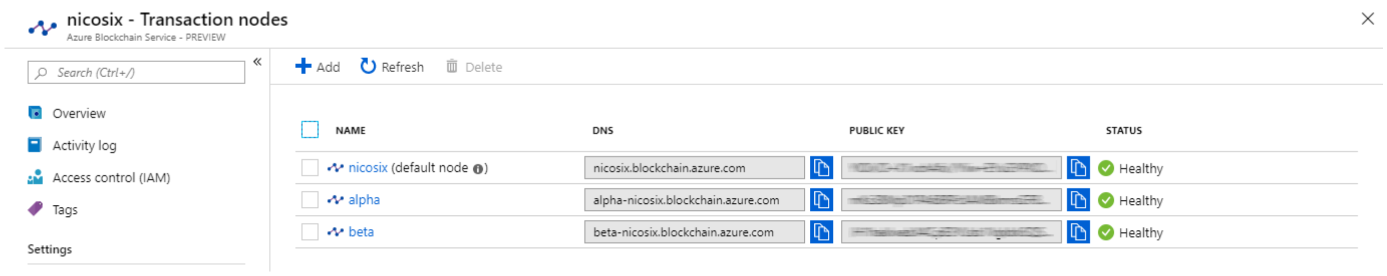
Member name : nicosix
Protocol : Quorum
Pricing Tier : Basic (1 vCore, 4 nodes)
Consortium : nsixconsortium
RootContract address : 0xb255f55e8d600f09ebc1035dd2118acec1018912
Member account : 0xc0de490ac332467028e32cb753622fd169bf9300

Search (Ctrl+/)
Overview
Activity log
Access control (IAM)
Tags
Settings
Properties
Locks
Export template
Pricing tier
Transaction nodes
Monitoring
Alerts
Metrics
Diagnostic settings
Logs
Support + troubleshooting
New support request
Consortium
Invite or remove members from the consortium.
Start
Let your voice be heard
Give feedback on Azure Blockchain Service.
Go
Blockchain Development Kit
Sample code and accelerators to get you started.
Go
Show data for last: 1 hour 6 hours 12 hours 1 day 7 days 30 days
Blocks
Transactions
Pending Transactions
3:15 PM 3:30 PM 3:45 PM 4 PM
validator-node-0 nicosix 726
transaction-node nicosix 720
Alpha nicosix 720
transaction-node nicosix 0
Alpha nicosix 0
beta nicosix 0
Alpha nicosix 0
transaction-node nicosix 0
beta nicosix 0

As a "guided tour", below are the main features of this **Overview** pane:

- **Consortium management**. This menu allows you to manage companies and people inside the consortium linked to your Azure Blockchain Service.
- **Information tile**. It contains all information about your service (RootContract address, your member account address, service name, etc.).
- **Monitoring tiles**. They allow you to track health and activity of your Azure Blockchain Service.

You can also click on the **Transaction nodes** under the **Settings** section. This is where you can administrate your transaction and validation nodes or get connection strings. It will be useful later in this guide when you will deploy new applications on Azure Blockchain Service, see MODULE 3: CREATION OF APPLICATIONS.


nicosix - Transaction nodes
Azure Blockchain Service - PREVIEW

NAME	DNS	PUBLIC KEY	STATUS
nicosix (default node)	nicosix.blockchain.azure.com	[redacted]	Healthy
alpha	alpha-nicosix.blockchain.azure.com	[redacted]	Healthy
beta	beta-nicosix.blockchain.azure.com	[redacted]	Healthy

It's now time to move to the next module, i.e. the second one, where you're going to see how the consortium works. To do so, click on **Start** in the **Consortium** tile and let's jump into the second module.

Module 2: Consortium management

Overview

Consortium is one of the top features of Azure Blockchain Service. It allows companies to work together, toward the same goal, by grouping them into a decentralized "organization", which rules blockchain transactions and memberships.

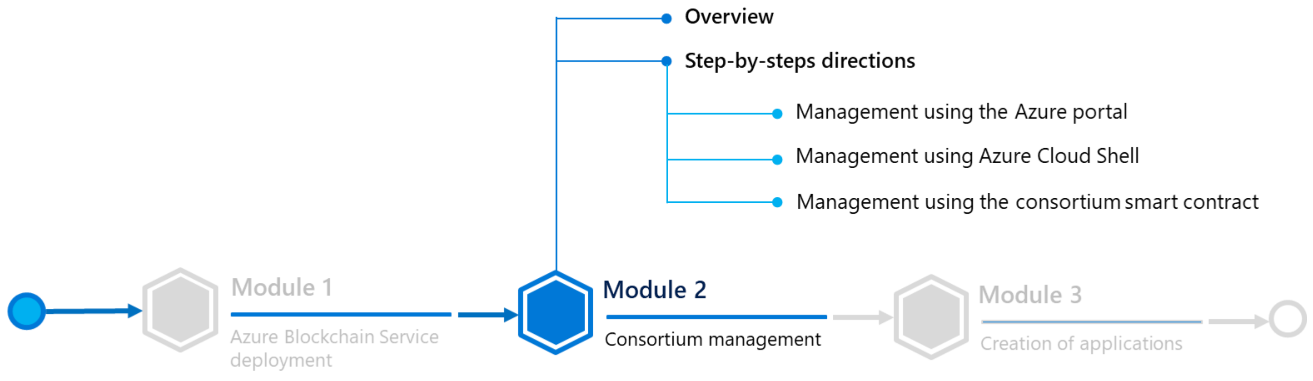
Every member of the consortium has defined permissions, which allows it to administrate the consortium/the blockchain, or just use the network depending on the permissions.

As such, the consortium can be managed in two different ways:

1. By using the **Consortium** pane.
2. By using the Command Line Interface (CLI) interface, notably available in Azure Cloud Shell.

In this guide, you will focus on the CLI interface and the management through the RootContract as those can be considered as the most efficient ways to manage a consortium, in so far as you can use external tools and applications to remotely connect to Azure Blockchain Service and execute the commands.

However, you will begin this part by having a look at the **Consortium** pane inside the Azure portal.


Overview
Step-by-steps directions
Management using the Azure portal
Management using Azure Cloud Shell
Management using the consortium smart contract
Module 1
Azure Blockchain Service deployment
Module 2
Consortium management
Module 3
Creation of applications

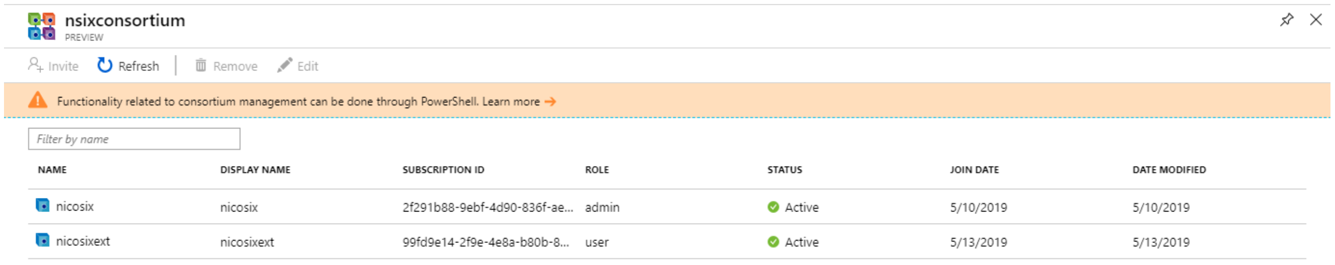
Step-by-step directions

This module covers the following activities:

1. Managing the consortium using Azure portal.
2. Managing the consortium using Azure Cloud Shell.
3. Managing the consortium using the consortium smart contract.

Each activity is described in order in the next sections.

Managing the consortium using Azure portal


nsixconsortium
PREVIEW

Invite | Refresh | Remove | Edit

Functionality related to consortium management can be done through PowerShell. Learn more →

Filter by name

NAME	DISPLAY NAME	SUBSCRIPTION ID	ROLE	STATUS	JOIN DATE	DATE MODIFIED
nicosix	nicosix	2f291b88-9ebf-4d90-836f-ae...	admin	Active	5/10/2019	5/10/2019
nicosixext	nicosixext	99fd9e14-2f9e-4e8a-b80b-8...	user	Active	5/13/2019	5/13/2019

In the **Consortium** pane, you can see the current members of the consortium. A member can be a User or an Administrator, which gives him permissions depending of his status:

Action	**User role**	**Administrator role**
Create new member	Yes	Yes
Invite new members	No	Yes
Set or change member participant role	No	Yes
Change member display name	Only for own member	Only for own member
Remove members	Only for own member	Yes
Participate in blockchain transactions	Yes	Yes

In this pane, users can be invited or removed one by one.

If you invite a new user, it will be displayed as invited in the list until this user creates an instance of Azure Blockchain Workbench[24] inside his subscription.

Let's now move to the command line.

Managing the consortium using Azure Cloud Shell

Overview and preliminary setup

In this part, you are going to consider the Consortium management through Azure Cloud Shell[25].

Azure Cloud Shell is a free CLI which can be launched right from the Azure Portal and has common Azure tools preinstalled and configured to use with your account. This is a good way to test the PowerShell commands used to manage the consortium with ease.

[24] Azure Blockchain Workbench: https://azure.microsoft.com/en-us/features/blockchain-workbench/

[25] Azure Cloud Shell: https://azure.microsoft.com/en-us/features/cloud-shell/

However, if you want to integrate this type of consortium management into another application (like a web application), you can remotely connect to the members' smart contract which is inside Azure Blockchain Service. You will see how this is possible using a code sample in the third part of this module. Stay tuned!

To launch Azure Cloud Shell, simply click the related icon in the toolbar of the Azure portal. (You can alternatively open a new tab and go on the URL https://shell.azure.com/PowerShell, sign-in with your (Microsoft) account associated with the same Azure subscription as Azure Blockchain Service.).

Regardless of the chosen approach, in Cloud Shell, select PowerShell (in lieu of Bash, which is also available).

Paste and execute those commands to install the PowerShell module required for consortium management:

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

Import-Module Microsoft.AzureBlockchainService.ConsortiumManagement.PS
```

You are now ready to execute the consortium commands.

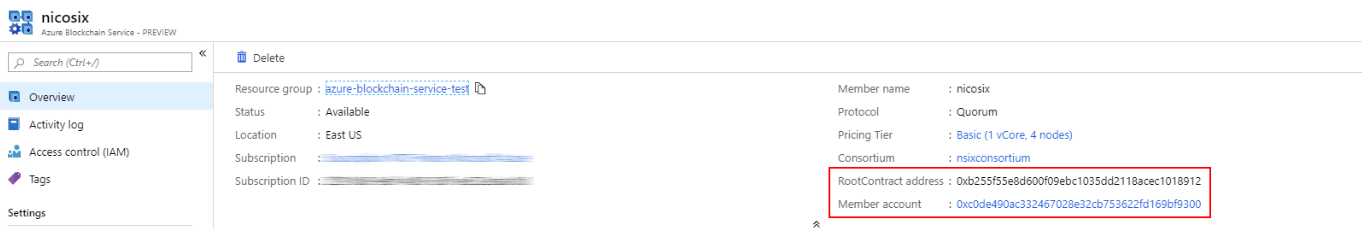
To save your time, we're going to create three PowerShell variables:

1. One containing the ledger information: `$Ledger`,
2. One containing your Ethereum account unlocked (use to authenticate you on the blockchain): `$MemberAccount`,
3. And eventually one containing contract connection information: `$ContractConnection`.

For that purpose, type and execute in order the three commands lines below where you will first update the required fields, i.e. <xxx>, with your information:

```
$Ledger = New-Web3Connection -RemoteRPCEndpoint '<Your transaction node URL containing your access
key>'

$MemberAccount = Import-Web3Account -ManagedAccountAddress '<Your Ethereum account public address>' -
ManagedAccountPassword '<Your account password>'

$ContractConnection = Import-ConsortiumManagementContracts -RootContractAddress
'<RootContractAddress>' -Web3Client $Ledger
```

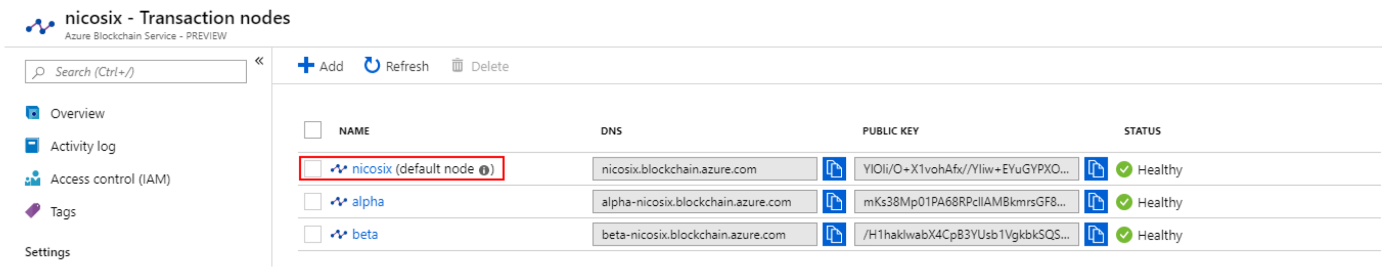
Note Your Ethereum public address and your RootContract address can be found on the **Overview** pane of your Azure Blockchain Service instance.



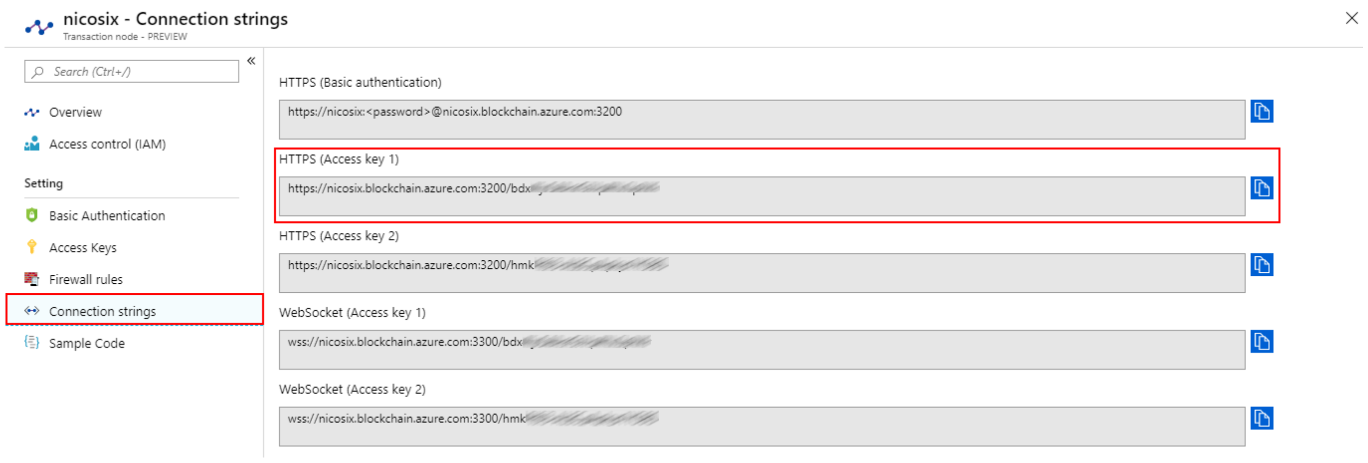
Resource group : azure-blockchain-service-test
Status : Available
Location : East US
Subscription : [redacted]
Subscription ID : [redacted]

Member name : nicosix
Protocol : Quorum
Pricing Tier : Basic (1 vCore, 4 nodes)
Consortium : nsixconsortium
RootContract address : 0xb255f55e8d600f09ebc1035dd2118acec1018912
Member account : 0xc0de490ac332467028e32cb753622fd169bf9300

Your transaction node URL can be found into the **Transaction nodes** pane. Click on your default node name, then on the connection string, and eventually copy the HTTPS connection string containing the access key as illustrated hereafter.


nicosix - Transaction nodes
Azure Blockchain Service - PREVIEW

NAME	DNS	PUBLIC KEY	STATUS
nicosix (default node)	nicosix.blockchain.azure.com	YIOli/O+X1vohAfx//Yliw+EYuGYPXO...	Healthy
alpha	alpha-nicosix.blockchain.azure.com	mKs38Mp01PA68RPclIAMBkmrsGF8...	Healthy
beta	beta-nicosix.blockchain.azure.com	/H1haklwabX4CpB3YUsb1VgkbkSQS...	Healthy


nicosix - Connection strings
Transaction node - PREVIEW

HTTPS (Basic authentication)
https://nicosix:<password>@nicosix.blockchain.azure.com:3200

HTTPS (Access key 1)
https://nicosix.blockchain.azure.com:3200/bdx[redacted]

HTTPS (Access key 2)
https://nicosix.blockchain.azure.com:3200/hmk[redacted]

WebSocket (Access key 1)
wss://nicosix.blockchain.azure.com:3300/bdx[redacted]

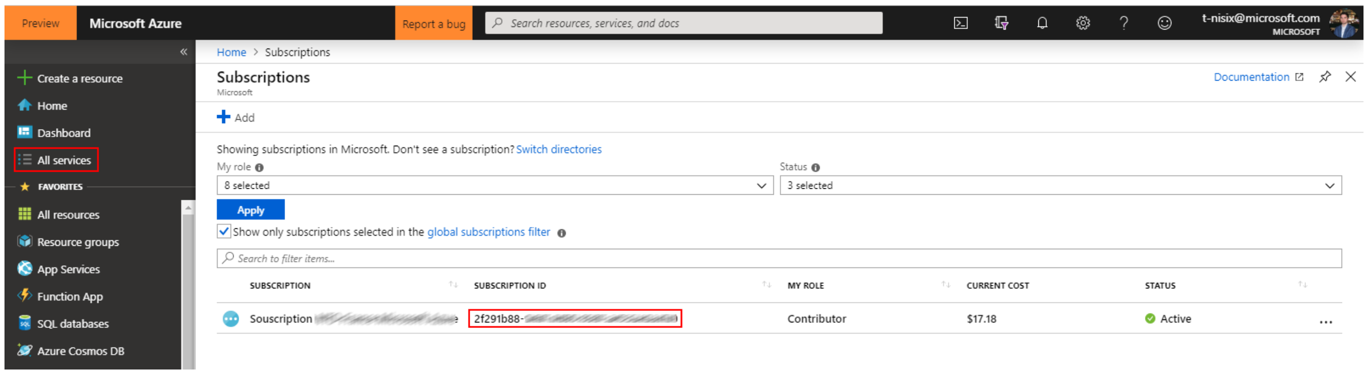
WebSocket (Access key 2)
wss://nicosix.blockchain.azure.com:3300/hmk[redacted]

It's time to consider the available commands.

Leveraging available commands

Inviting a new member (admin)

For the sake of this guide, you will invite our own subscription to try this feature. You can get your subscription ID by going in your Azure portal, then click on **All services** and **Subscriptions**.



Subscriptions
Microsoft

Showing subscriptions in Microsoft. Don't see a subscription? Switch directories

My role: 8 selected
Status: 3 selected

Show only subscriptions selected in the global subscriptions filter

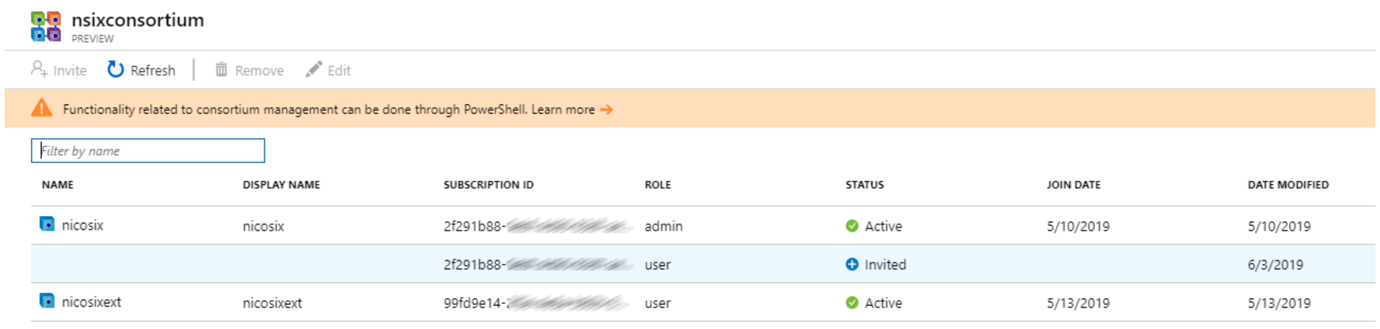
Search to filter items...

SUBSCRIPTION	SUBSCRIPTION ID	MY ROLE	CURRENT COST	STATUS
Souscription [redacted]	2f291b88-[redacted]	Contributor	$17.18	Active

When found, paste the following line into your Azure Cloud Shell by replacing the `SubscriptionId` parameter value by your own Subscription ID:

```
$ContractConnection | New-BlockchainMemberInvitation -SubscriptionId <Newly invited subscription ID>
-Role USER -Web3Account $MemberAccount
```

Now, if you go back into your **Consortium** pane, you will see that a new member was invited to the consortium.

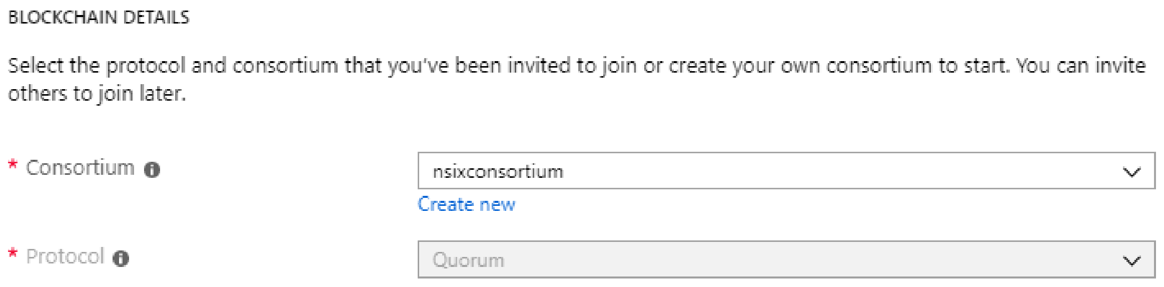

nsixconsortium
PREVIEW

Functionality related to consortium management can be done through PowerShell. Learn more →

Filter by name

NAME	DISPLAY NAME	SUBSCRIPTION ID	ROLE	STATUS	JOIN DATE	DATE MODIFIED
nicosix	nicosix	2f291b88-[redacted]	admin	● Active	5/10/2019	5/10/2019
		2f291b88-[redacted]	user	● Invited		6/3/2019
nicosixext	nicosixext	99fd9e14-[redacted]	user	● Active	5/13/2019	5/13/2019

If you try to create another Azure Blockchain Service, you will have the possibility to join this consortium instead of creating a new one.


BLOCKCHAIN DETAILS

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

* Consortium	nsixconsortium
	Create new
* Protocol	Quorum

Note You can try for instance another Azure Blockchain Service to create a second member in your consortium for test reasons, but this is not mandatory for this guide.

Canceling an invitation (admin)

To do so, just execute the following command:

```
Remove-BlockchainMemberInvitation -Members $ContractConnection.Members -SubscriptionId <Subscription
ID to delete> -Web3Account $MemberAccount -Web3Client $Ledger
```

Wait a few minutes to complete the command and go back in your **Consortium** pane. You should see that your invitation disappeared from the consortium.

Removing a member (admin)

If needed over the time, you can remove a member (whatever his role is) by typing the following command:

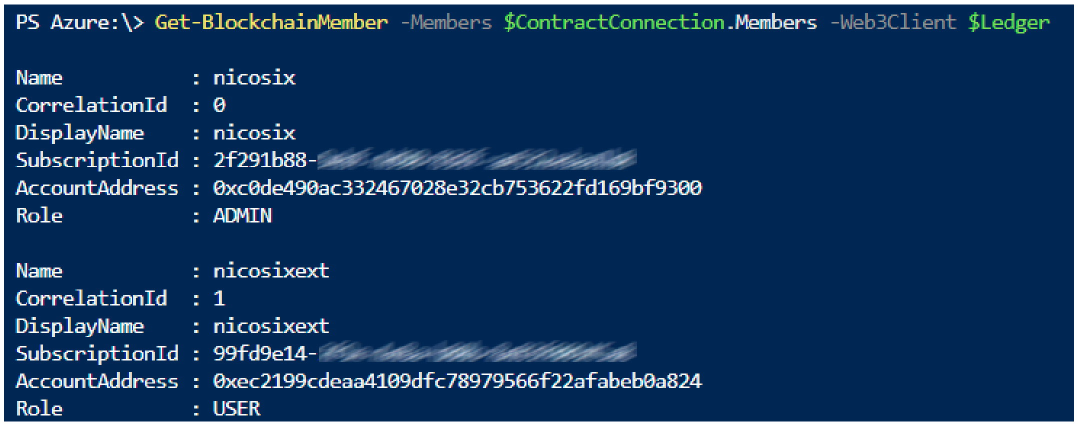
```
Remove-BlockchainMember -Name “<Members’ name>” -Members $ContractConnection.Members -Web3Account
$MemberAccount -Web3Client $Ledger
```

Getting members' profiles (all)

To get the members of the consortium, simply type the following command:

```
Get-BlockchainMember -Members $ContractConnection.Members -Web3Client $Ledger
```

You will get the following output from Azure Cloud Shell:



```
PS Azure:\> Get-BlockchainMember -Members $ContractConnection.Members -Web3Client $Ledger

Name           : nicosix
CorrelationId  : 0
DisplayName    : nicosix
SubscriptionId : 2f291b88-[redacted]
AccountAddress : 0xc0de490ac332467028e32cb753622fd169bf9300
Role           : ADMIN

Name           : nicosixext
CorrelationId  : 1
DisplayName    : nicosixext
SubscriptionId : 99fd9e14-[redacted]
AccountAddress : 0xec2199cdeaa4109dfc78979566f22afabeb0a824
Role           : USER
```

You can also add the `-Name "<searched name>"` argument into your command line if you want to get profile information about a specific member of the consortium. You will get in return a profile information like the ones outputted by the list command, or an error if the profile does not exist.

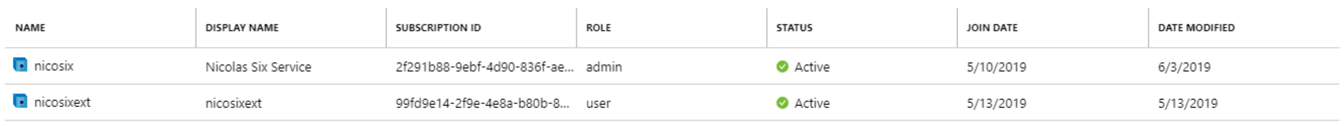
Modifying members' attributes (all)

The `Set-BlockchainMember` cmdlet allows you to modify the role of a member (User or Administrator), as well as his displayed name. Note that a User can only modify his displayed name.

To do so, type the following command by replacing corresponding fields with their actual values:

```
Set-BlockchainMember -Name "<Member service name>" -DisplayName "<New displayed name>" -Members
$ContractConnection.Members -Web3Account $MemberAccount -Web3Client $Ledger
```

You can now go back into your **Consortium** pane to check effective changes:



NAME	DISPLAY NAME	SUBSCRIPTION ID	ROLE	STATUS	JOIN DATE	DATE MODIFIED
nicosix	Nicolas Six Service	2f291b88-9ebf-4d90-836f-ae...	admin	✓ Active	5/10/2019	6/3/2019
nicosixext	nicosixext	99fd9e14-2f9e-4e8a-b80b-8...	user	✓ Active	5/13/2019	5/13/2019

Now, if you want to change a member's role, you just have to type the following command:

```
Set-BlockchainMember -Name "<Member service name>" -Role "<user or admin>" -Members
$ContractConnection.Members -Web3Account $MemberAccount -Web3Client $Ledger
```

Make sure that you correctly typed user or admin, in lowercase or uppercase, because otherwise, you will get an error…

You will see in your **Consortium** pane that the selected user has their role changed.

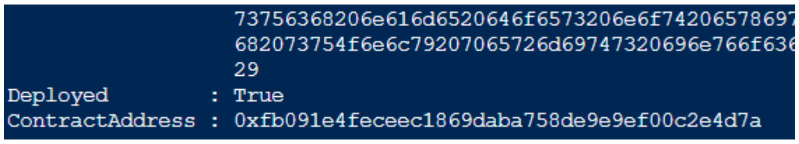
Getting consortium smart contract address (all)

For the next part of this module, you will interact directly with the consortium smart contract hosted by Azure Blockchain Service.

To do so, you will need the smart contract address, which can be obtained by typing:

```
$ContractConnection.Members
```

You will get some information about the smart contract, just copy and save the address for the next part.



```
                   73756368206e616d6520646f6573206e6f74206578697
                   682073754f6e6c79207065726d69747320696e766f636
                   29
Deployed         : True
ContractAddress  : 0xfb091e4feceec1869daba758de9e9ef00c2e4d7a
```

Managing the consortium using the consortium smart contract

Using Azure Cloud Shell to manage the consortium is fine when you want to do small operations or just try the feature, but when you want to integrate consortium management inside an application, a better approach may consist in directly interacting with the smart contract which rules the consortium located inside the Azure Blockchain Service blockchain.

To do so, you will need to retrieve first the contract's Ethereum address as outlined above. Then, you will be able use it in a web app with:

1. Web3, an Ethereum JavaScript API which connects to the Generic JSON RPC[26] specification.
2. Truffle, an Ethereum framework to interact with the Members contract by providing his address and other information about Azure Blockchain Service.

To get the address, if you don't have it yet, please follow the directions of **OVERVIEW AND PRELIMINARY SETUP** then **LEVERAGING AVAILABLE COMMANDS > GETTING CONSORTIUM SMART CONTRACT ADDRESS (ALL)** from section § **MANAGING THE CONSORTIUM USING AZURE CLOUD SHELL**.

Then, download or clone the code samples' repo here[27] on GitHub if you haven't already done that before.

Leveraging the Truffle code sample

One of the first samples that you will use is based on Truffle. This is a framework for Ethereum that helps developers to create smart contracts and decentralized applications, from development to deployment on the live network.

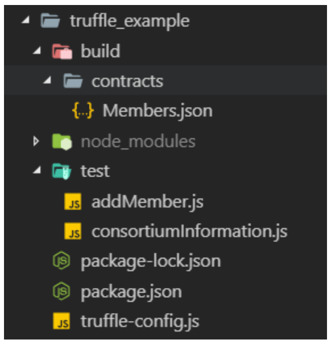
Note In this part, you're only going to briefly run the sample, but stay tuned! One of the following parts will indeed cover Truffle with greater details!s"

[26] JSON RPC: https://github.com/ethereum/wiki/wiki/JSON-RPC

[27] Guide Samples' projet : https://aka.ms/ABSDevGuideSamples

Inside the code samples' root directory, under *consortium-management-samples*, you will find a directory named *truffle_example*. Open it.

Following is the content of this directory:

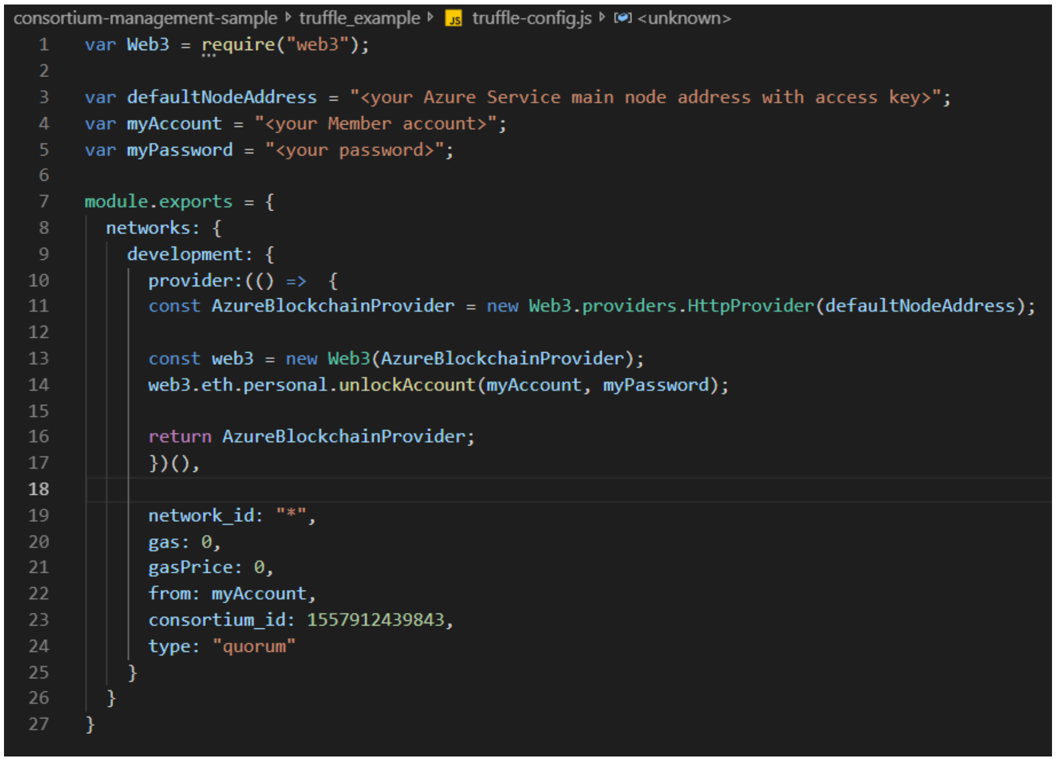


```
truffle_example
  build
    contracts
      Members.json
  node_modules
  test
    addMember.js
    consortiumInformation.js
  package-lock.json
  package.json
  truffle-config.js
```

- *build*. This directory usually contains smart contract artifacts. For this sake of this guide, you will only find here the contract Application Binary Interface (ABI) - an ABI contains the list of every callable function of the smart contract, their parameters and returned values -.
- *test*. This directory contains two scripts samples, one to invite a new member and another one to get members list.
- *truffle-config.js*. This configuration file contains information about how Truffle will establish a connection with the blockchain.

To start, open the *truffle-config.js* file. You will see inside of it three variables, which respectively represent your node address (with access key) and your consortium account credentials.

Note We cannot stress enough how writing credentials inside the code is a bad practice, and this is only used here to easily test the sample. You must remove this information after the test being done.


consortium-management-sample › truffle_example › truffle-config.js › <unknown>

```javascript
var Web3 = require("web3");

var defaultNodeAddress = "<your Azure Service main node address with access key>";
var myAccount = "<your Member account>";
var myPassword = "<your password>";

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

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

      return AzureBlockchainProvider;
      })(),

      network_id: "*",
      gas: 0,
      gasPrice: 0,
      from: myAccount,
      consortium_id: 1557912439843,
      type: "quorum"
    }
  }
}
```

Replace all the variables placeholders by your own values:

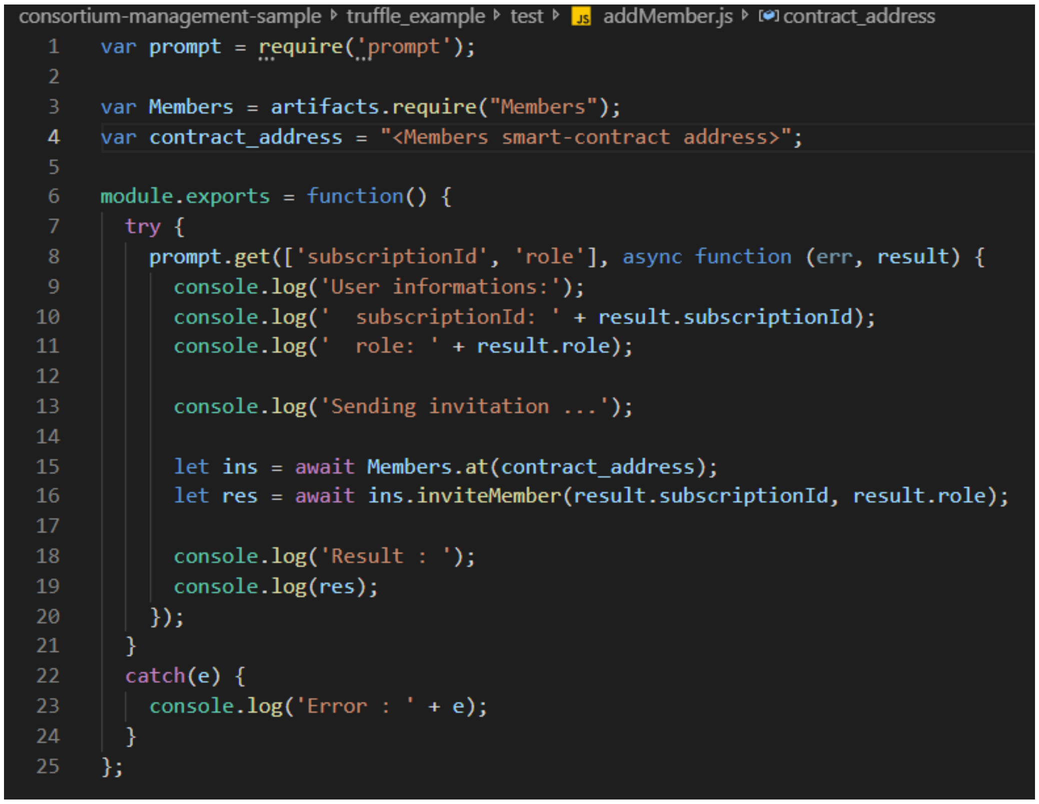
Variable	How to get it?
defaultNodeAddress	This variable is your transaction node URL, displayed on your transaction node pane > connection strings. You can follow the directions of the part **MANAGING THE CONSORTIUM USING AZURE CLOUD SHELL** > **OVERVIEW AND PRELIMINARY SETUP** to see how to obtain it.
myAccount	This variable is your Ethereum public address, displayed on your Azure Blockchain Service pane, in the Azure Portal. You can also refer to the part **MANAGING THE CONSORTIUM USING AZURE CLOUD SHELL** > **OVERVIEW AND PRELIMINARY SETUP** to understand where you can obtain it.
myPassword	This variable is your account password, defined when you created Azure Blockchain Service.

Once done, you will need to specify the members contract address in the following two JavaScript files:

1. *addMember.js*,
2. *consortiumInformation.js*.

, where the address is the one retrieved in the part **MANAGING THE CONSORTIUM USING AZURE CLOUD SHELL > LEVERAGING AVAILABLE COMMANDS > GETTING CONSORTIUM SMART CONTRACT ADDRESS (ALL)**.

Open each of the two above JavaScript files, and replace the address placeholder with your own address:



consortium-management-sample › truffle_example › test › addMember.js › contract_address

```
var prompt = require('prompt');

var Members = artifacts.require("Members");
var contract_address = "<Members smart-contract address>";

module.exports = function() {
  try {
    prompt.get(['subscriptionId', 'role'], async function (err, result) {
      console.log('User informations:');
      console.log('  subscriptionId: ' + result.subscriptionId);
      console.log('  role: ' + result.role);

      console.log('Sending invitation ...');

      let ins = await Members.at(contract_address);
      let res = await ins.inviteMember(result.subscriptionId, result.role);

      console.log('Result : ');
      console.log(res);
    });
  }
  catch(e) {
    console.log('Error : ' + e);
  }
};
```

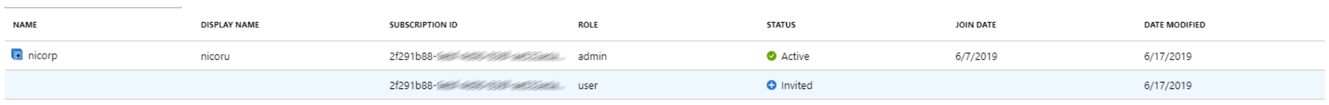
You're now ready to test the sample. Open a terminal console and, from the command prompt, go inside the *truffle_example* directory.

Once inside, type those commands:

```
npm install
truffle exec test/addMember.js -network development
```

The console will prompt you for a subscription ID. You can type in your own to try.

After that, press ENTER and wait for confirmation. You should now see that your invitation is pending in your consortium panel.

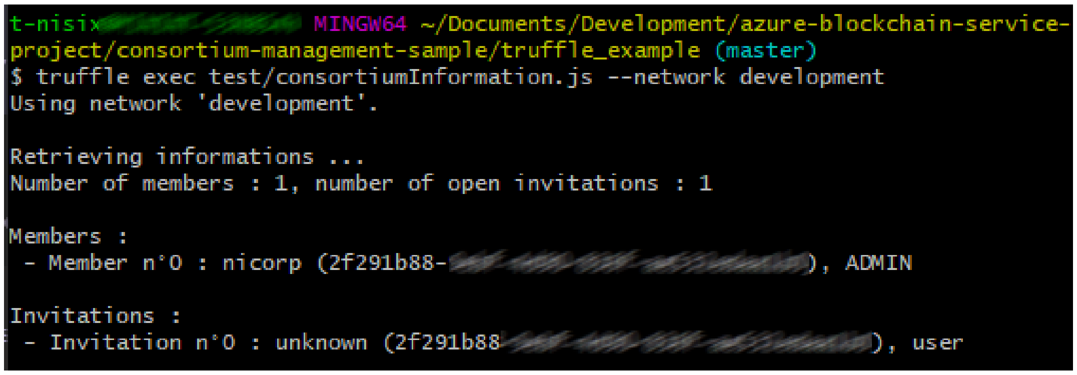


NAME	DISPLAY NAME	SUBSCRIPTION ID	ROLE	STATUS	JOIN DATE	DATE MODIFIED
nicorp	nicoru	2f291b88-[redacted]	admin	✓ Active	6/7/2019	6/17/2019
		2f291b88-[redacted]	user	⊕ Invited		6/17/2019

You can also execute the following additional command, which list members and invitations in your consortium. Simply type the following command and press ENTER:

```
truffle exec test/consortiumInformation.js -network development
```

You will see the consortium details appear right after that:



```
t-nisix[redacted] MINGW64 ~/Documents/Development/azure-blockchain-service-
project/consortium-management-sample/truffle_example (master)
$ truffle exec test/consortiumInformation.js --network development
Using network 'development'.

Retrieving informations ...
Number of members : 1, number of open invitations : 1

Members :
 - Member n°0 : nicorp (2f291b88-[redacted]), ADMIN

Invitations :
 - Invitation n°0 : unknown (2f291b88[redacted]), user
```

When you type "*truffle exec* …" in your console, Truffle uses the file *truffle-config.js* to connect and transact with your contract using your code in the called file.

The illustration we provide here is simply an example, but generally speaking, and as you will discover over the time, Truffle is really useful when testing newly created contracts.

To sum-up a little bit where you are right now, you've seen so far how to communicate with the consortium contract using Truffle with console commands.

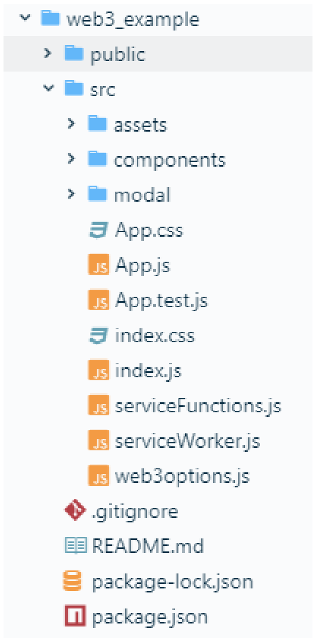
In the next section, you will see how to do the same thing using Web3, a more generic JavaScript API.

Leveraging the Web3 code sample

In this part, we're going to use Web3, which is a collection of libraries which allow you to interact with an Ethereum node (and therefore Azure Blockchain Service) to build an application capable of managing the consortium. (If needed, you can refer to the Web3 documentation here[28])

[28] web3.js - Ethereum JavaScript API : https://web3js.readthedocs.io/en/1.0/

To do so, we will run a React[29] web application, structured as follows:



```
web3_example
  public
  src
    assets
    components
    modal
    App.css
    App.js
    App.test.js
    index.css
    index.js
    serviceFunctions.js
    serviceWorker.js
    web3options.js
  .gitignore
  README.md
  package-lock.json
  package.json
```

- *public*. This directory contains the *html* content for the app, driven by React JavaScript library.
- *src*. This directory contains the source code:
    - *assets*. This directory is where images and contract ABIs are stored.
    - *components*. This directory is where the app components are defined, including the navbar, the members table, etc.
    - *modal*. This directory is where form contents for modal windows are stored.
    - *App.js*: This JavaScript file is the starting component of the application.
    - *Index.js*. This JavaScript file is the starting point of the application, display the starting component.
    - *web3options.js*. This JavaScript file contains a Web3 instance constructor, called by functions to request something on the blockchain.
    - *serviceFunctions.js*: This JavaScript file contains functions called by components to operate the contract.

To start, open the file web3options.js and change variables placeholders by your own values, like what you've done in the previous part, as the variables are the same.

[29] React JavaScript library: https://reactjs.org/

As a kind reminder, here are the variables to replace:

Variable	How to get it?
defaultNodeAddress	This variable is your transaction node URL, displayed on your transaction node pane > connection strings. You can follow the directions of the part **MANAGING THE CONSORTIUM USING AZURE CLOUD SHELL** > **OVERVIEW AND PRELIMINARY SETUP** to see how to obtain it.
serviceAddress	This variable is your Ethereum public address, displayed on your Azure Blockchain Service pane, on Azure Portal. You can also refer to the part **MANAGING THE CONSORTIUM USING AZURE CLOUD SHELL** > **OVERVIEW AND PRELIMINARY SETUP** to understand where you can obtain it.
servicePassword	This is your account password, defined when you created Azure Blockchain Service.
contractAddress	This is the address retrieved in the part **MANAGING THE CONSORTIUM USING AZURE CLOUD SHELL > LEVERAGING AVAILABLE COMMANDS > GETTING CONSORTIUM SMART CONTRACT ADDRESS (ALL)**.

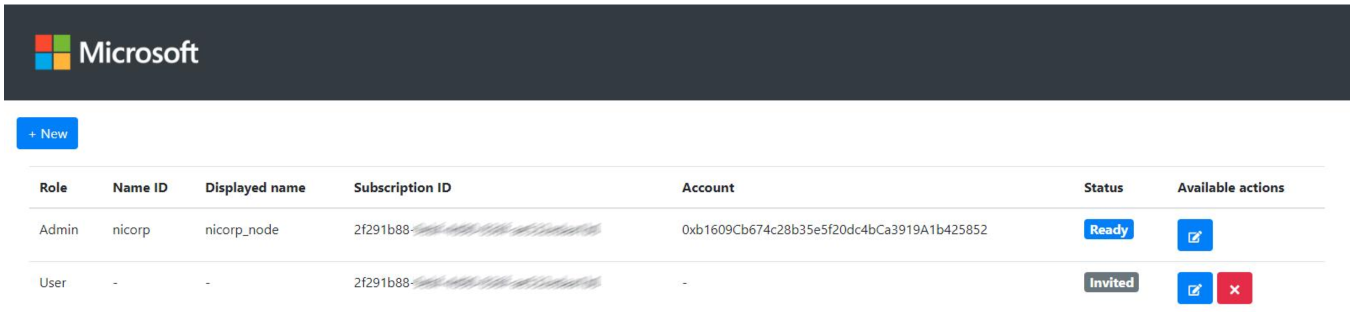
Note Writing credentials inside the code is a bad practice, and only used here to test the sample.

Now we're ready to start the application.

Open a terminal console, go inside the above directory, and type the following command:

```
npm run start
```

A web server will start, and the application will be displayed in your main browser. You will see your consortium, and some buttons to interact with it:



Role	Name ID	Displayed name	Subscription ID	Account	Status	Available actions
Admin	nicorp	nicorp_node	2f291b88-[redacted]	0xb1609Cb674c28b35e5f20dc4bCa3919A1b425852	Ready	
User	-	-	2f291b88-[redacted]	-	Invited	

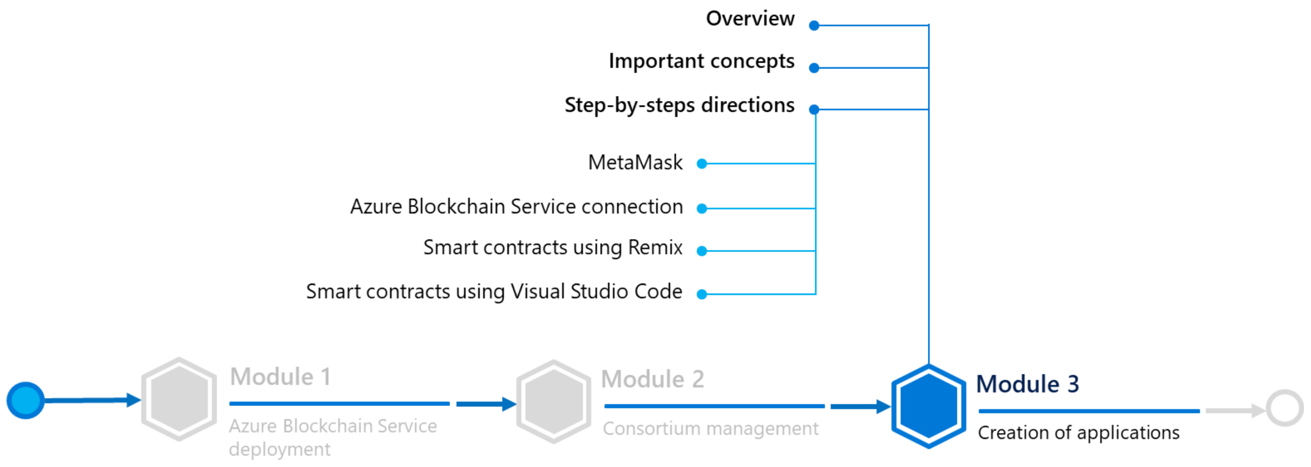
If you want to, you can have a look to the code, especially in the file *serviceFunctions.js* where the blockchain stuff is done and the contract is being called.

In this part, we briefly saw how to interact with smart contracts nested in the blockchain, the next part will be about creating and testing your own smart contracts.

Module 3: Creation of applications

Overview

In this last module, you will spend some time on decentralized application (ĐApp) development.


Overview
Important concepts
Step-by-steps directions
MetaMask
Azure Blockchain Service connection
Smart contracts using Remix
Smart contracts using Visual Studio Code
Module 1
Azure Blockchain Service deployment
Module 2
Consortium management
Module 3
Creation of applications

Before doing that, let’s consider some important concepts.

Important concepts

Smart contract

In some blockchain ledger technologies (as Ethereum Quorum in Azure Blockchain Service blockchain), smart contracts are supported.

A smart contract is a chunk of code which is deployed inside the blockchain and can be accessed via transactions. Each contract has an address which identifies it on it and an ABI, a file obtained after contract compilation which describes what can be done with the contract (functions with parameters and returned values, variable reading, etc.).

Anybody on a permissionless public blockchain can call a contract and read its values, and on a private blockchain it depends of the blockchain type (for example, Quorum allows users to do private transactions and therefore, contract values can only be read by transaction parties).

A contract is, by extension, a sum of blockchain transactions sent to him. Therefore, every function call which changed contract data cannot be reversed (can only be counterbalanced by the opposite transaction for example).

Developing decentralized applications (ĐApps) via smart contracts is different from classic developments in so far as all those specificities should be taken in account. In this part, we will see how to develop, compile and test them, on multiple code editors or frameworks.

Ethereum Remix IDE

Remix[30] is an IDE developed by the Ethereum Foundation, an open-source organization in charge of a number of projects around the Ethereum blockchain.

Accessible through a browser, Remix is a great tool to start experimenting with an Ethereum blockchain, and therefore Azure Blockchain Service.

Step-by-step directions

This module covers the following activities:

1. Installing MetaMask.
2. Setting up the Azure Blockchain Service connection.
3. Developing, building and testing smart contracts with Remix.
4. Developing, building and testing contracts with Visual Studio Code.

Each activity is described in order in the next sections.

Installing MetaMask

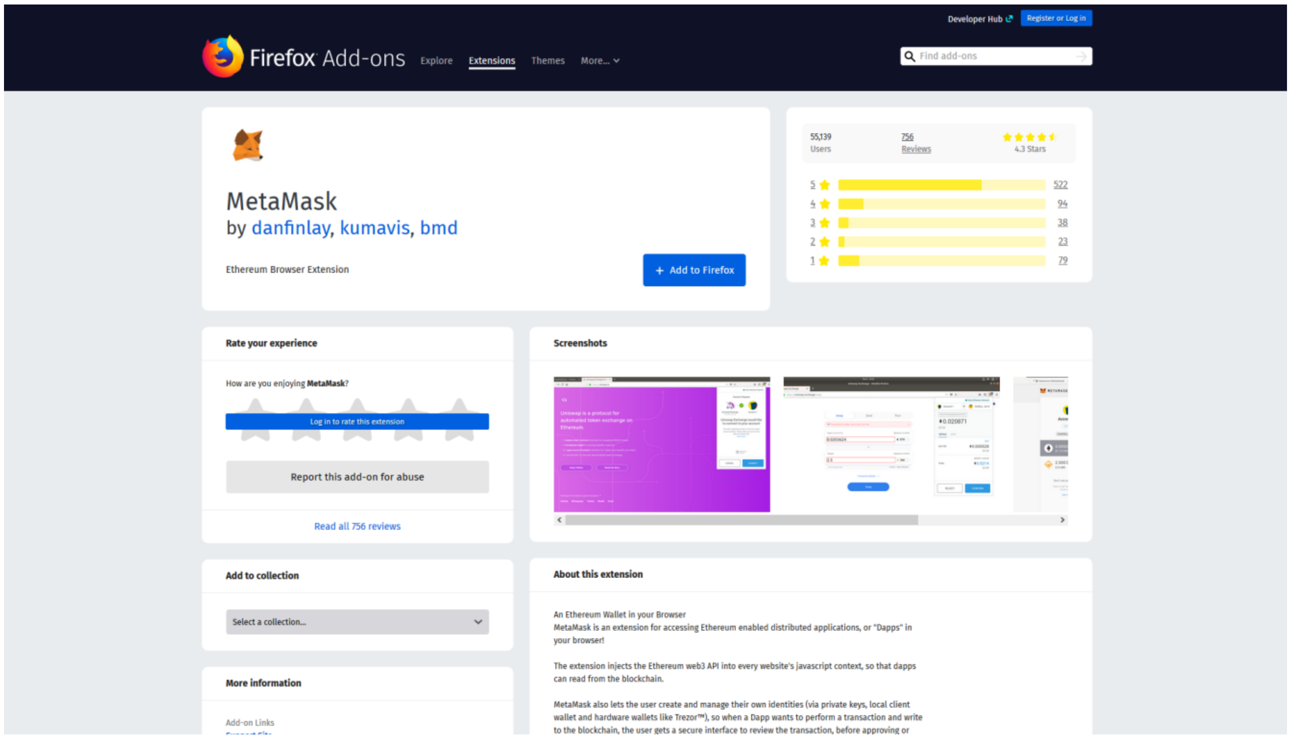
To use it, we will have to install MetaMask first. MetaMask is a browser extension that allows a user to send some transactions to blockchain networks. You will use it to sign-in and send transactions through Remix for testing purposes.

MetaMask is compatible with the following browsers: Edge (chromium), Chrome, Firefox, Opera and Brave. For the sake of this guide, we will install it on Firefox[31], but the extension can work on any browser previously mentioned.

Open a new tab if you are on a compatible browser or choose another one if not.

Go on the MetaMask website[32], and click on **Get Firefox Addon**. A window will open with the **Firefox MetaMask extension** page, click on **Add to Firefox** and when It will ask you for permissions, click on **Add**.

[30] Remix - Ethereum IDE: https://remix.ethereum.org/
[31] Mozilla Firefox: https://www.mozilla.org/en/firefox/
[32] MetaMask: https://metamask.io/



MetaMask
by danfinlay, kumavis, bmd

Ethereum Browser Extension

55,139 Users
756 Reviews
4.3 Stars

5 ★	522
4 ★	94
3 ★	38
2 ★	23
1 ★	79

Rate your experience

How are you enjoying **MetaMask**?

Screenshots

Add to collection

More information

Add-on Links

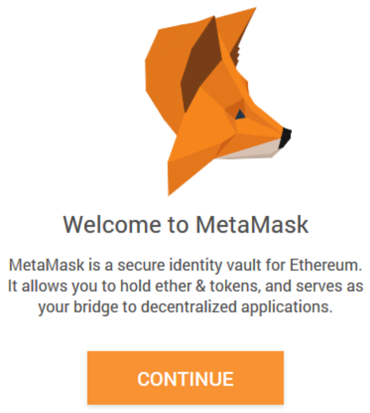
About this extension

An Ethereum Wallet in your Browser
MetaMask is an extension for accessing Ethereum enabled distributed applications, or "Dapps" in your browser!

The extension injects the Ethereum web3 API into every website's javascript context, so that dapps can read from the blockchain.

MetaMask also lets the user create and manage their own identities (via private keys, local client wallet and hardware wallets like Trezor™), so when a Dapp wants to perform a transaction and write to the blockchain, the user gets a secure interface to review the transaction, before approving or

After that, the extension will open itself, and prompt you to begin the installation. Click on **Continue.**



Welcome to MetaMask

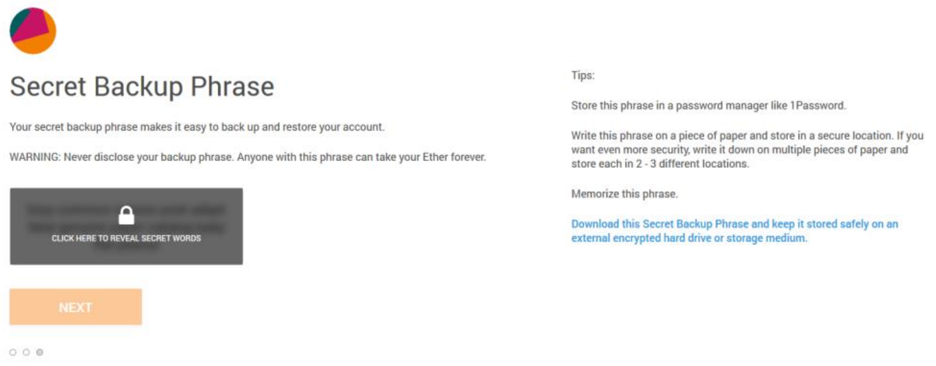
MetaMask is a secure identity vault for Ethereum. It allows you to hold ether & tokens, and serves as your bridge to decentralized applications.

It will ask you to create a password. Type one and click on **Create,** then **Next** and accept the **Terms of Use, Privacy Notice & Phishing Warning**.

You will arrive on the Secret Backup **Phrase** page. Click on the lock to reveal the secret words.

This phrase is REALLY important, because it represents your Ethereum account. Indeed, thanks to a cryptographic algorithm, you can generate your Ethereum Wallet from this phrase.

Note If you want to use this wallet in the future and have the possibility to retrieve your account in any case, **please keep this phrase SECRET** as anybody who has this phrase is able to connect to your account**, and save it somewhere** like on a paper or in a password manager.



Secret Backup Phrase

Your secret backup phrase makes it easy to back up and restore your account.

WARNING: Never disclose your backup phrase. Anyone with this phrase can take your Ether forever.

Tips:

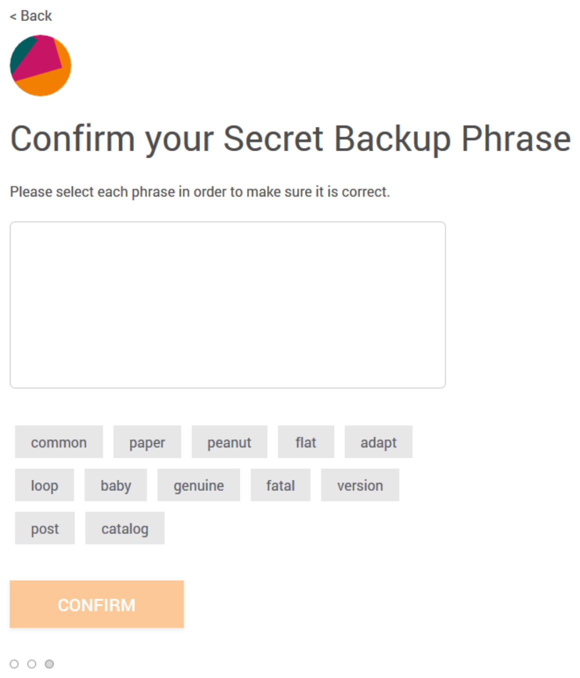
Store this phrase in a password manager like 1Password.

Write this phrase on a piece of paper and store in a secure location. If you want even more security, write it down on multiple pieces of paper and store each in 2 - 3 different locations.

Memorize this phrase.

Download this Secret Backup Phrase and keep it stored safely on an external encrypted hard drive or storage medium.

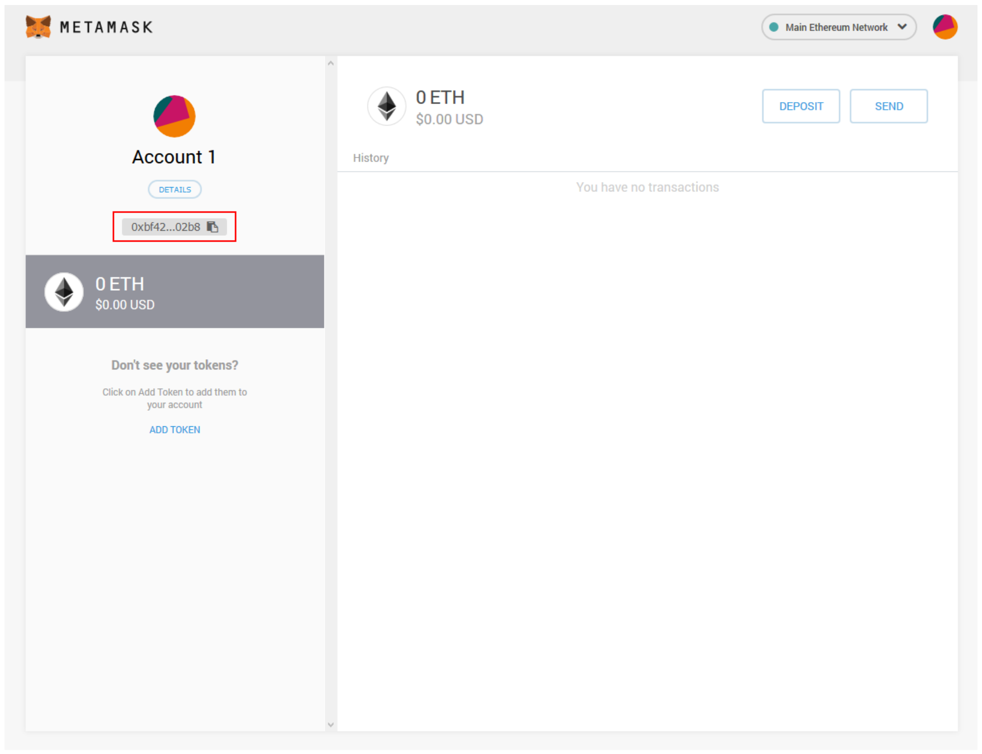
After saving it and clicking **Next**, it will ask you to select each word to make sure you wrote your phrase somewhere. Do it by clicking on every word in order and click **Confirm**.



Confirm your Secret Backup Phrase

Please select each phrase in order to make sure it is correct.

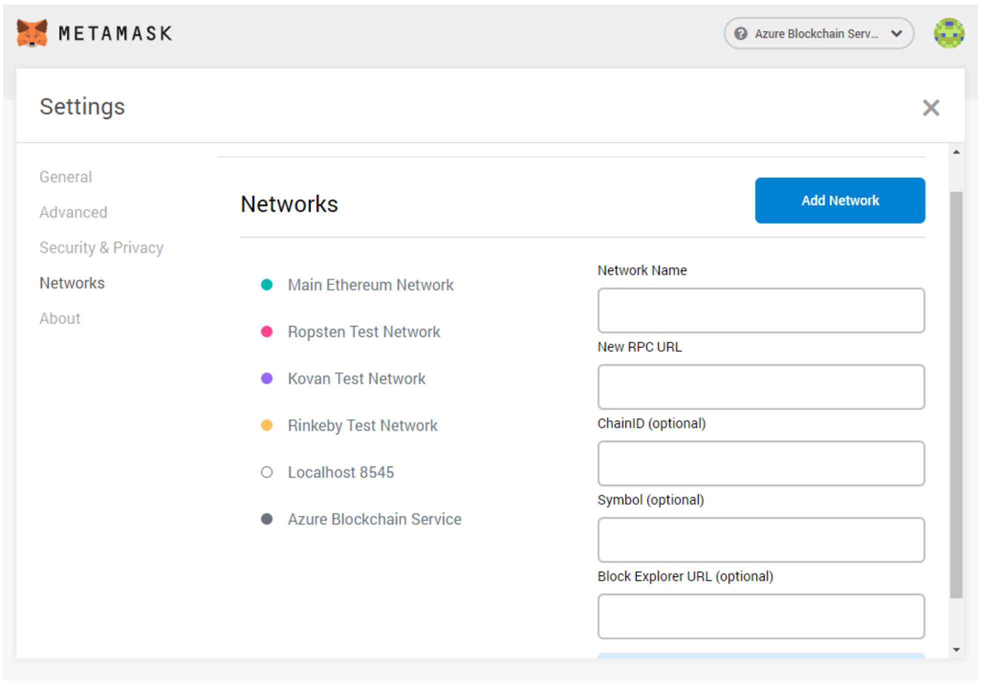
Now that the installation is finished, you can see the main menu, which displays your account address and your transaction history (here, for the main Ethereum blockchain network).



Setting up the Azure Blockchain Service connection

Now that you have MetaMask installed, you need to connect it to your Azure Blockchain Service. On MetaMask, click on **Settings**, **Network** then **Add network**.

Complete the form with your Azure Blockchain Service settings:



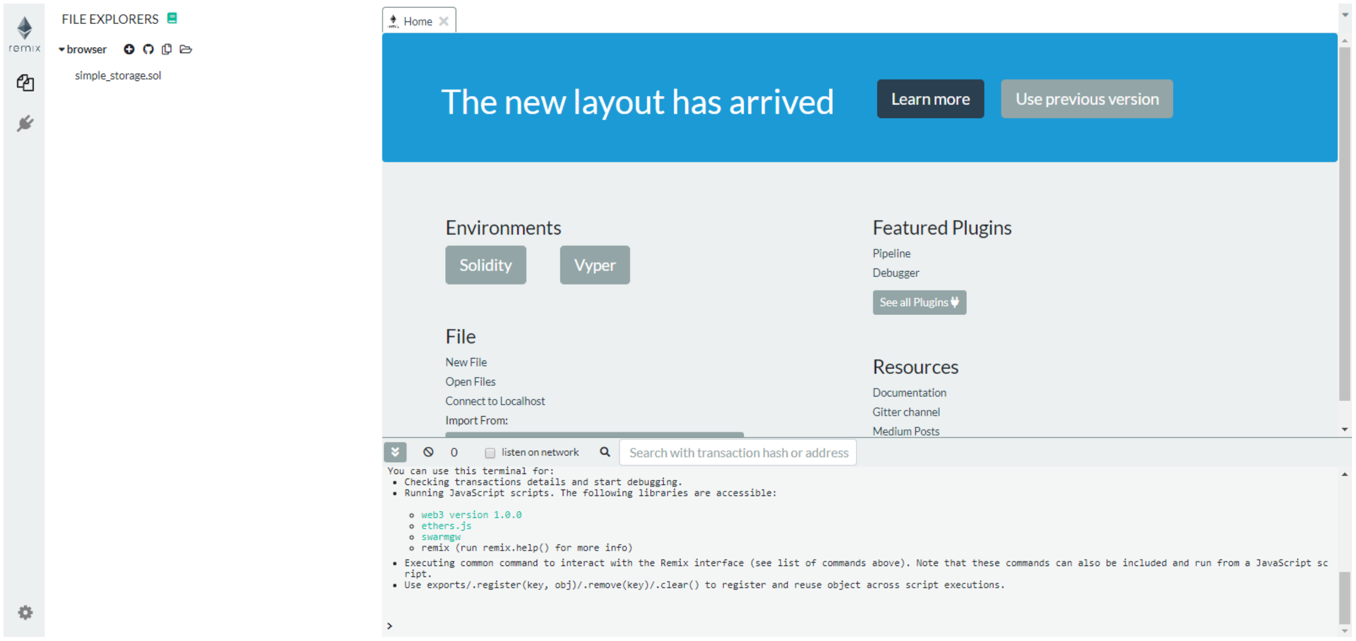
Field	Description
Network Name	This is the network name which will be displayed in MetaMask. You can type anything you want, for example **Azure Blockchain Service**.
New RPC URL	This variable is your transaction node URL, displayed on your transaction node pane > connection strings. You can follow the directions of the part **MANAGING THE CONSORTIUM USING AZURE CLOUD SHELL** > **OVERVIEW AND PRELIMINARY SETUP** to see how to get it.
ChainID (optional)	Leave it empty.
Symbol (optional)	Type **ETH**, as the network "currency" is Ethereum (even if transactions are free).
Block Explorer URL (optional)	Leave it empty, not needed in this example.

Click on **Save**, and *MetaMask* will be connected to your network. You're now able to interact with Azure Blockchain Service on Remix.

Developing, building and testing smart contracts with Remix

You're finally ready to start creating smart contracts and decentralized applications!

Open https://remix.ethereum.org in the same browser than MetaMask. You will arrive in your IDE:



```
You can use this terminal for:
 • Checking transactions details and start debugging.
 • Running JavaScript scripts. The following libraries are accessible:

    o web3 version 1.0.0
    o ethers.js
    o swarmgw
    o remix (run remix.help() for more info)
 • Executing common command to interact with the Remix interface (see list of commands above). Note that these commands can also be included and run from a JavaScript sc
   ript.
 • Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across script executions.

>
```

Here, you're being asked to choose an environment. Select **Solidity** as you will use it to create your contracts. After choosing it, you will see multiple menus appear on the left. Those are:

- A file explorer for all your contracts in Remix.
- The Solidity compiler, to compile your contract and therefore get the ABI and the binary code of your contract.

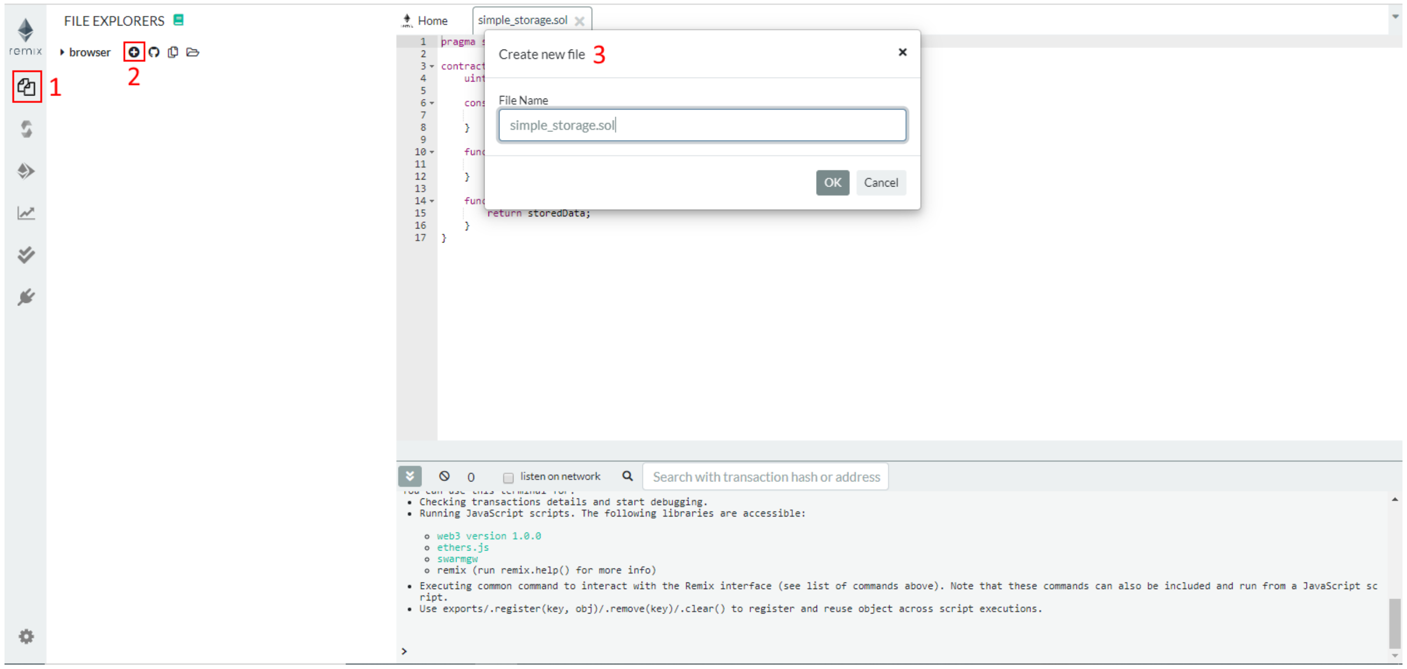
- The deployment menu, where you can deploy contracts and interact with them on multiple networks (including our Azure Blockchain Service network in this case).
- A unit test section, where you can generate contracts dedicated to test your main contracts, using functions like *Assert*.
- A solidity static analysis menu, where the IDE will run tests on your contract to ensure that security and performance standards are met.
- A plugin manager to configure your IDE.

As you can see, Remix is quite complete as an IDE.

To start and test how everything inside it works, you will use a really simple contract named **Simple Storage**.

Simple Storage is simply an integer stored in the contract, and it can be changed or retrieved at any moment by anybody. You will find the code inside the GitHub repo[33] for this guide, under *solidity-contract-sample* > *simple_storage.sol*.

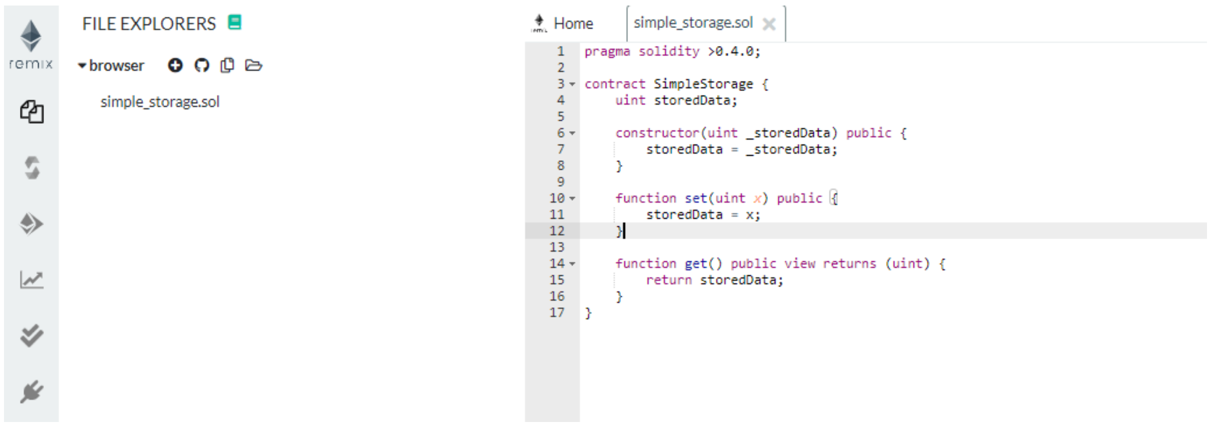
Create a file inside Remix and paste the content of *simple_storage.sol* inside it.



```
pragma
contract
    uint
    cons
    }
    func
    }
    func
        return storedData;
    }
}
```

```
• Checking transactions details and start debugging.
• Running JavaScript scripts. The following libraries are accessible:

    o web3 version 1.0.0
    o ethers.js
    o swarmgw
    o remix (run remix.help() for more info)

• Executing common command to interact with the Remix interface (see list of commands above). Note that these commands can also be included and run from a JavaScript sc
  ript.
• Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across script executions.

>
```

[33] Guide Samples' projet : https://aka.ms/ABSDevGuideSamples



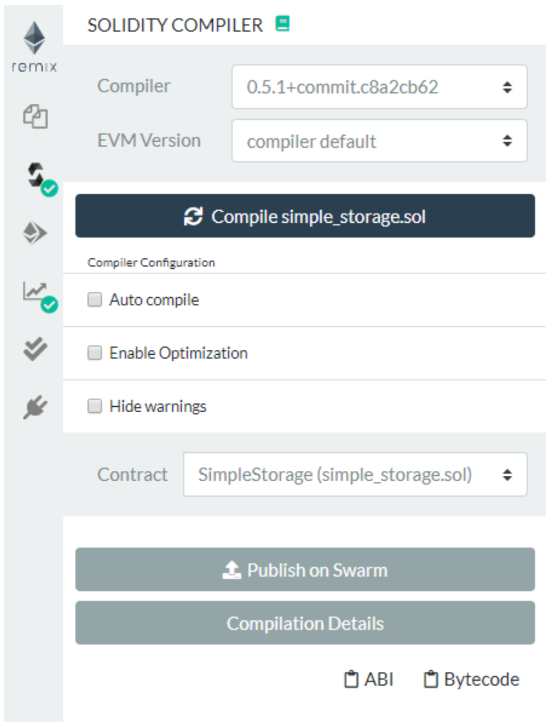
```
pragma solidity >0.4.0;

contract SimpleStorage {
    uint storedData;

    constructor(uint _storedData) public {
        storedData = _storedData;
    }

    function set(uint x) public {
        storedData = x;
    }

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

As you can see, the contract is constituted of an integer variable, a constructor and two functions:

- uint *storedData*: is the integer being stored.
- *constructor*: is called only once (when you initially deploy the contract). It initializes the variable value with the integer passed as an argument.
- function *set*: is used to change the value of the integer.
- function *get*: is used to retrieve the value of the integer - in Solidity, you can also directly get the value of a contract property by calling it instead of a function, so this function is not really useful… -.

To ensure that everything is in order, you will compile the above simple contract : click on *simple_storage.sol* in your file explorer, then click on the solidity compiler icon in the menu bar (just under the file explorer one) and click on **Compile simple_storage.sol.**

If everything is right, you should see two little green validated icons appear on the solidity compiler and the solidity static analysis icons. This indicates that your contract has been compiled and no warnings were found.



Compiler: 0.5.1+commit.c8a2cb62

EVM Version: compiler default

- [ ] Auto compile
- [ ] Enable Optimization
- [ ] Hide warnings

Contract: SimpleStorage (simple_storage.sol)

When a smart contract is compiled, it generates two files: an Application Binary Interface (ABI) file and a Bytecode file.

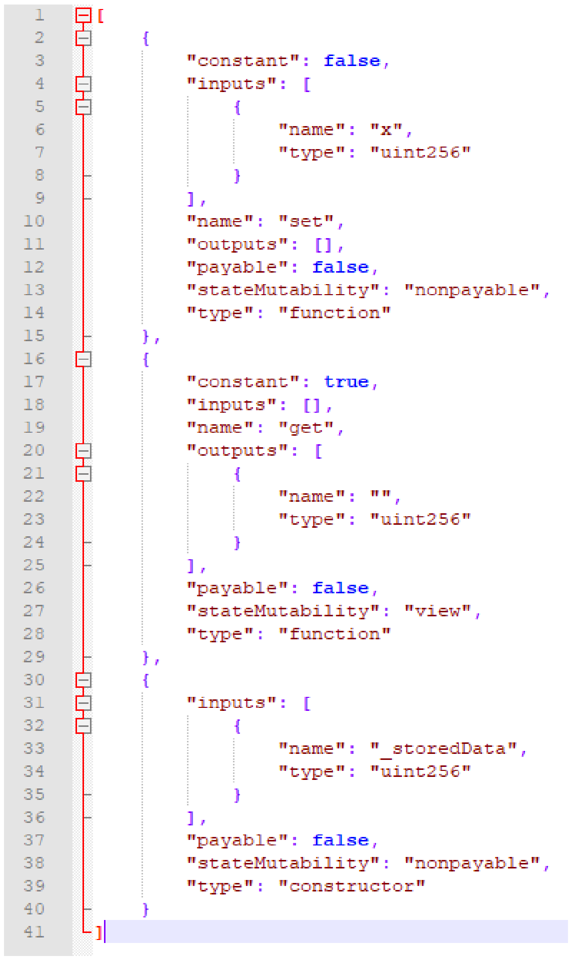
Opening the bytecode is quite useless as it is only constituted with numbers or assembly code used by machines, but if you want to, you can click on ABI, which will copy the ABI content into your clipboard and you can then paste it in a code editor or notepad to see what it looks like.

An ABI file is a file that describes the contract with JSON metadata.

You will see that, in the ABI, every contract function is defined by their name, inputs, outputs and other function parameters.

The ABI file is used when you want to interact with the associated contract, already deployed somewhere. Indeed, Ethereum libraries and frameworks (such as Web3 and Truffle) use the ABI to send transactions to contracts.

Hereafter is the Simple Storage ABI:



```
[
    {
        "constant": false,
        "inputs": [
            {
                "name": "x",
                "type": "uint256"
            }
        ],
        "name": "set",
        "outputs": [],
        "payable": false,
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "constant": true,
        "inputs": [],
        "name": "get",
        "outputs": [
            {
                "name": "",
                "type": "uint256"
            }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "name": "_storedData",
                "type": "uint256"
            }
        ],
        "payable": false,
        "stateMutability": "nonpayable",
        "type": "constructor"
    }
]
```

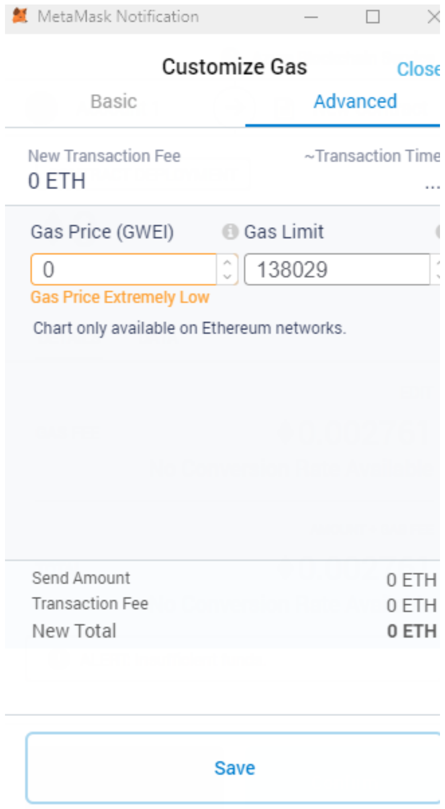
You will not have to use it here, because Remix handles the ABI for you. But this is interesting to see as you will use it later in this guide.

Now that you are sure that your contract is well-compiled, you can deploy it. To do so, go inside the **Deploy & Run transactions** menu. Click on the **Environment** dropdown and select **Injected Web3**. This corresponds to the default MetaMask network, so by extension Azure Blockchain Service.

Once selected, leave other fields by default, type an integer value for example "*12*" in the input next to the **Deploy** button and click on **Deploy**. A MetaMask popup will open, asking you to confirm the transaction.

It will display you an error, as MetaMask is asking for funds to execute the transaction and you don't have any, because you are on a blockchain where there isn't any currency.

To remove it, click on **Edit** then on the **Advanced section** of the menu and change Gas Price (GWEI) field to "**0**", then click on **Save**.



New Transaction Fee 0 ETH ~Transaction Time ...

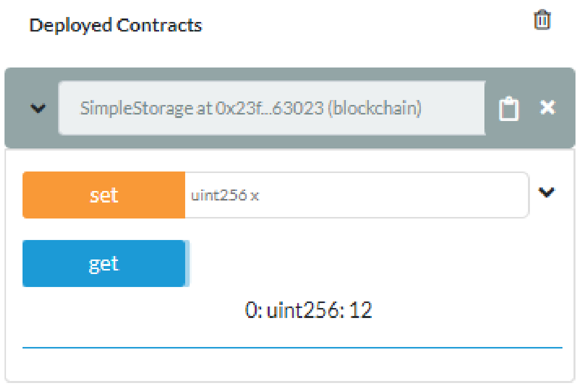
Gas Price (GWEI): 0 — Gas Price Extremely Low
Gas Limit: 138029

Chart only available on Ethereum networks.

Send Amount 0 ETH
Transaction Fee 0 ETH
New Total **0 ETH**

You're now able to send the transaction, on the main notification menu click on **Confirm**. Wait a little bit to get your transaction executed, and you will get a notification which says that everything went well.

Let's interact with the newly created contract: click on the **SimpleStorage** contract under **Deployed Contract**, and 2 rows will appear: one for the *get* function and one for the *set* function.



SimpleStorage at 0x23f...63023 (blockchain)

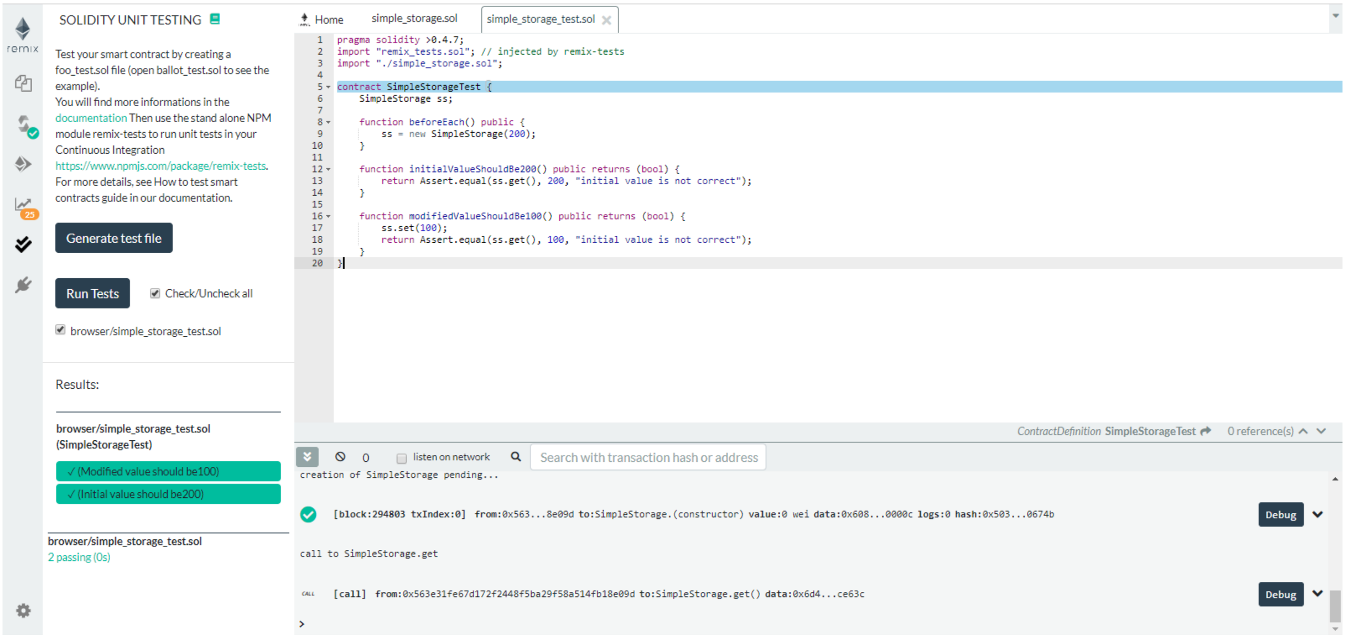
0: uint256: 12

You can try to click on **get**, and you will get in return the value entered when you created your contract (here, 12 for our illustration). You can also send a new value to the contract by typing it into set and clicking on the function button. You will have to do the same things with MetaMask as when you've deployed the contract - change gas price and confirm -, and after a while, your contract value will change.

Testing contracts with this interface is nice, but for production contracts you will need to automate testing. Interestingly enough, Remix is equipped with a unit test library for smart contracts to do so - you can get more information here[34] about the library being used -.

To start:

1. Create another file in the File Explorer and name it *simple_storage_test.sol*.
2. Fill it with the content of the *simple_storage_test.sol* file in our GitHub code sample which can be found at the same emplacement as *simple_storage.sol*.
3. Select it in Remix and click on the **Solidity** unit testing icon.



Test your smart contract by creating a foo_test.sol file (open ballot_test.sol to see the example).
You will find more informations in the documentation Then use the stand alone NPM module remix-tests to run unit tests in your Continuous Integration https://www.npmjs.com/package/remix-tests.
For more details, see How to test smart contracts guide in our documentation.

browser/simple_storage_test.sol

Results:

browser/simple_storage_test.sol (SimpleStorageTest)
✓ (Modified value should be100)
✓ (Initial value should be200)

browser/simple_storage_test.sol
2 passing (0s)

```
pragma solidity >0.4.7;
import "remix_tests.sol"; // injected by remix-tests
import "./simple_storage.sol";

contract SimpleStorageTest {
    SimpleStorage ss;

    function beforeEach() public {
        ss = new SimpleStorage(200);
    }

    function initialValueShouldBe200() public returns (bool) {
        return Assert.equal(ss.get(), 200, "initial value is not correct");
    }

    function modifiedValueShouldBe100() public returns (bool) {
        ss.set(100);
        return Assert.equal(ss.get(), 100, "initial value is not correct");
    }
}
```

ContractDefinition SimpleStorageTest 0 reference(s)

0 listen on network Search with transaction hash or address

```
creation of SimpleStorage pending...

[block:294803 txIndex:0] from:0x563...8e09d to:SimpleStorage.(constructor) value:0 wei data:0x608...0000c logs:0 hash:0x503...0674b

call to SimpleStorage.get

CALL [call] from:0x563e31fe67d172f2448f5ba29f58a514fb18e09d to:SimpleStorage.get() data:0x6d4...ce63c

>
```

You can see that your test contract has been detected by your test unit menu, because its name ends by _*test*.

Try to click on **Run**: the contract will execute each function of itself and return a pass or a fail label for them.

Here, there are 2 straightforward test functions:

1. One that checks if the constructor value is correctly assigned to the integer variable,
2. Another one that checks if the set function works.

Those are really basic ones as there are intended for illustration purposes. In a more ambitious project, you're free to add as many tests as you want.

This part is now complete. You allowed you to discover some of the Remix features.

[34] Remix-Tests: https://github.com/ethereum/remix/tree/master/remix-tests

In the next and final part, you are going to see another popular code editor: Visual Studio Code. This editor has other interesting features in the context of this guide. Let's consider them.

Developing, building and testing contracts with Visual Studio Code

Developing applications on Remix is a good way to start while experimenting, but for production-grade application, opting for Visual Studio Code could be a better pick for all its features.

In this part, you will build step-by-step a smart contract-based application, using two extensions which enhances Visual Studio Code:

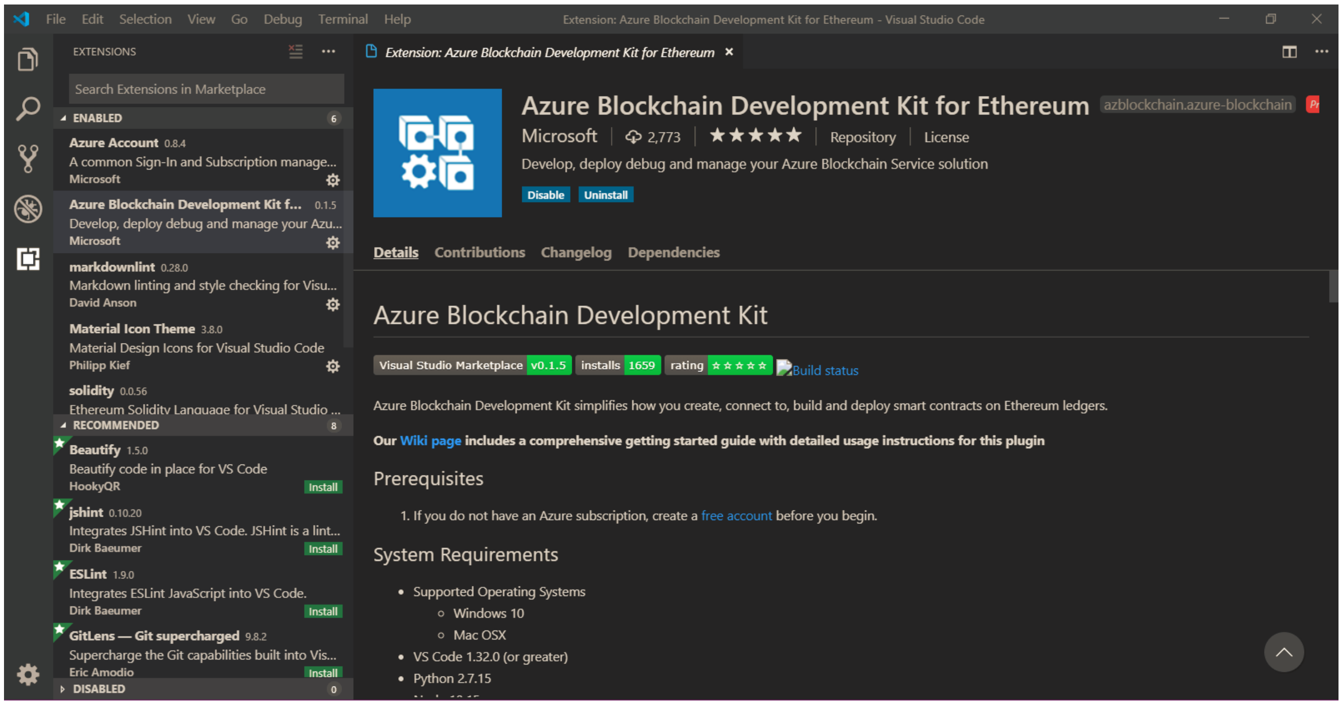
1. Azure Blockchain Development Kit for Ethereum[35].
2. Solidity[36].

The former adds a framework to smart contract development to Visual Studio Code, using Truffle and the Command Palette from the IDE. The latter is a linter and a compiler for Solidity contracts.

Installing required extensions

To install extensions in Visual Studio Code, go into **View** > **Extensions**. Then, search for "*Visual Studio Code: Azure Blockchain Development Kit for Ethereum & Solidity*" and once found, click on **Install**.

When correctly installed, you will see them in the **Enabled** column of extensions.



File Edit Selection View Go Debug Terminal Help
Extension: Azure Blockchain Development Kit for Ethereum - Visual Studio Code
EXTENSIONS
Search Extensions in Marketplace
ENABLED 6
Azure Account 0.8.4
A common Sign-In and Subscription manage...
Microsoft
Azure Blockchain Development Kit f... 0.1.5
Develop, deploy debug and manage your Azu...
Microsoft
markdownlint 0.28.0
Markdown linting and style checking for Visu...
David Anson
Material Icon Theme 3.8.0
Material Design Icons for Visual Studio Code
Philipp Kief
solidity 0.0.56
Ethereum Solidity Language for Visual Studio ...
RECOMMENDED 8
Beautify 1.5.0
Beautify code in place for VS Code
HookyQR
Install
jshint 0.10.20
Integrates JSHint into VS Code. JSHint is a lint...
Dirk Baeumer
Install
ESLint 1.9.0
Integrates ESLint JavaScript into VS Code.
Dirk Baeumer
Install
GitLens — Git supercharged 9.8.2
Supercharge the Git capabilities built into Vis...
Eric Amodio
Install
DISABLED 0
Extension: Azure Blockchain Development Kit for Ethereum
Azure Blockchain Development Kit for Ethereum
azblockchain.azure-blockchain
Microsoft
2,773
★★★★★
Repository
License
Disable
Uninstall
Details
Contributions
Changelog
Dependencies
Visual Studio Marketplace v0.1.5
installs 1659
rating ★★★★★
Build status

Develop, deploy debug and manage your Azure Blockchain Service solution

Azure Blockchain Development Kit

Azure Blockchain Development Kit simplifies how you create, connect to, build and deploy smart contracts on Ethereum ledgers.

Our Wiki page includes a comprehensive getting started guide with detailed usage instructions for this plugin

Prerequisites

1. If you do not have an Azure subscription, create a free account before you begin.

System Requirements

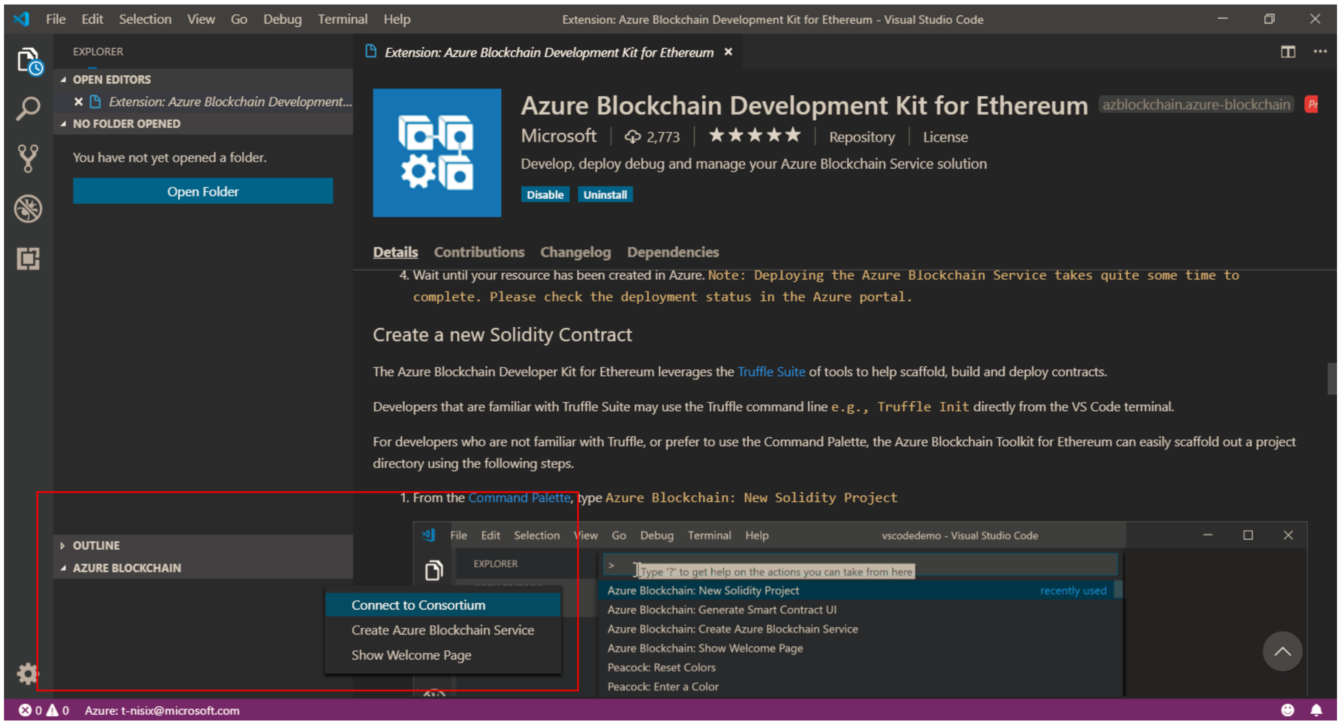
- Supported Operating Systems
  - Windows 10
  - Mac OSX
- VS Code 1.32.0 (or greater)
- Python 2.7.15

[35] Azure Blockchain Development Kit for Ethereum: https://marketplace.visualstudio.com/items?itemName=AzBlockchain.azure-blockchain

[36] Solidity: https://marketplace.visualstudio.com/items?itemName=JuanBlanco.solidity

And, after installing the two above extensions, you need to do one more thing: connect the first extension to Azure Blockchain Service consortium.

To do so, click on the **file icon** on the left, then in the Azure Blockchain pane click on the 3 dots and click on **Connect to Consortium**.


File Edit Selection View Go Debug Terminal Help
Extension: Azure Blockchain Development Kit for Ethereum - Visual Studio Code
EXPLORER
OPEN EDITORS
Extension: Azure Blockchain Development...
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
OUTLINE
AZURE BLOCKCHAIN
Connect to Consortium
Create Azure Blockchain Service
Show Welcome Page
Azure Blockchain Development Kit for Ethereum
azblockchain.azure-blockchain
Microsoft
2,773
Repository
License
Disable
Uninstall
Details
Contributions
Changelog
Dependencies
Type '?' to get help on the actions you can take from here
Azure Blockchain: New Solidity Project
recently used
Azure Blockchain: Generate Smart Contract UI
Azure Blockchain: Create Azure Blockchain Service
Azure Blockchain: Show Welcome Page
Peacock: Reset Colors
Peacock: Enter a Color
vscodedemo - Visual Studio Code
Azure: t-nisix@microsoft.com

Develop, deploy debug and manage your Azure Blockchain Service solution

4. Wait until your resource has been created in Azure. `Note: Deploying the Azure Blockchain Service takes quite some time to complete. Please check the deployment status in the Azure portal.`

Create a new Solidity Contract

The Azure Blockchain Developer Kit for Ethereum leverages the Truffle Suite of tools to help scaffold, build and deploy contracts.

Developers that are familiar with Truffle Suite may use the Truffle command line `e.g., Truffle Init` directly from the VS Code terminal.

For developers who are not familiar with Truffle, or prefer to use the Command Palette, the Azure Blockchain Toolkit for Ethereum can easily scaffold out a project directory using the following steps.

1. From the Command Palette, type `Azure Blockchain: New Solidity Project`

You will be prompted to choose between: connecting to a Local Ganache network (a sandbox blockchain provided by Truffle framework), Azure Blockchain Service, Test Ethereum network or Public Ethereum network.

Choose **Azure Blockchain Service**, and then, choose your subscription, the resource group where you created your service, and the corresponding service. After that, you will be connected to it.

Creating and configuring a project

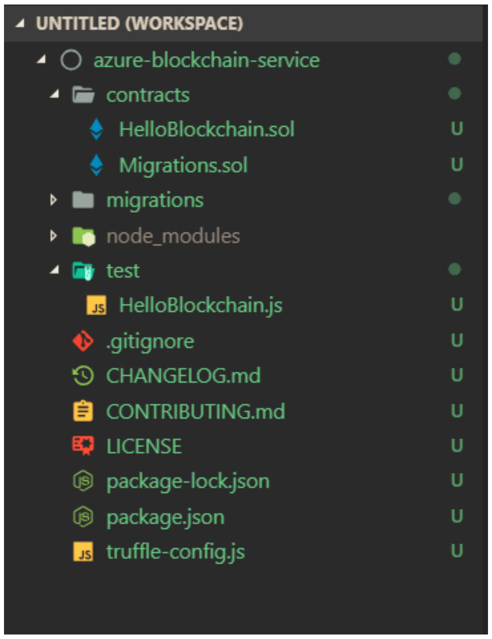
It's now time to create a new project. Fortunately, Azure Blockchain Development Kit[37] comes with a feature that creates a blank project.

To do so:

1. Go into **View** > **Command Palette** and type Azure Blockchain, then pick **New solidity project**.
2. Click on **Create basic project**.
3. Choose the repository you want to put your code and the installation will begin.
4. Make sure that you installed all needed packages to do so.

[37] Azure Blockchain Development Kit: https://azure.microsoft.com/en-us/resources/samples/blockchain-devkit/

This will be the result of the installation:



```
UNTITLED (WORKSPACE)
  azure-blockchain-service
    contracts
      HelloBlockchain.sol          U
      Migrations.sol               U
    migrations
    node_modules
    test
      HelloBlockchain.js           U
    .gitignore                     U
    CHANGELOG.md                   U
    CONTRIBUTING.md                U
    LICENSE                        U
    package-lock.json              U
    package.json                   U
    truffle-config.js              U
```

Following is the skeleton of a smart contract project, constituted with:

- *contracts*: where smart contracts are developed and stored.
    - *HelloBlockchain.sol* is the test contract you're going to use inside this project.
    - *Migrations.sol* is a contract generated by Truffle which tracks contracts deployment and updates.
- *migrations*: used to deploy contracts on the blockchain and track them. With those scripts, Truffle is aware of deployed contracts from this project and can manage contract updates.
- *test*: this is the directory where unit tests are stored.
- *truffle-config.js*: contains information to establish a connection with the blockchain.

As you want to interact with Azure Blockchain Service, you need to update the file *truffle-config.js* to take this in account.

In the guide's GitHub repo[38] code sample > **visual-studio-code-samples**, there is a file named *truffle-config.js*. Copy its content and paste it into the file *truffle-config.js* auto-generated by Visual Studio Code.

Replace placeholders' variables by your own values, where *myAccount* and *myPassword* are your Ethereum account credentials defined at Azure Blockchain Service creation and *defaultNode* is your node URL with its access key.

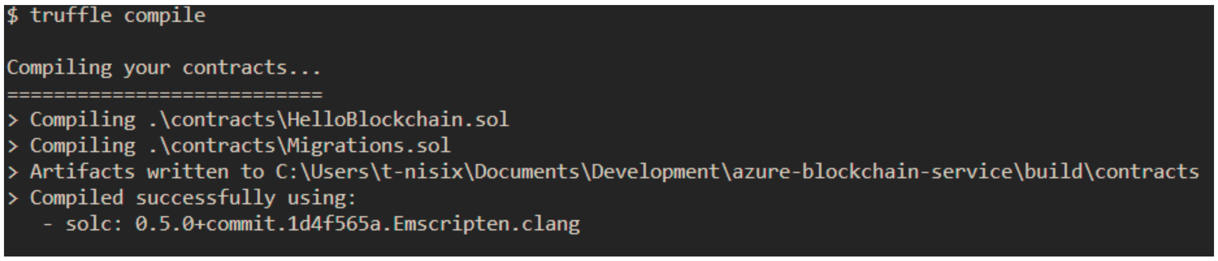
To build this project, you will use the smart contract generated with the project, and not develop a new one. But if you want to learn more about Solidity and smart contract programming, you can read Solidity documentation here[39], where the language is fully described. If you have some JavaScript or C++ background, you should easily learn it.

[38] Guide Samples' projet : https://aka.ms/ABSDevGuideSamples

[39] Solidity: https://solidity.readthedocs.io/en/v0.5.8/

Open a terminal console by going into **Terminal** > **New terminal**.

To start, let's type `truffle compile`. This will compile our contracts and return an artifact, containing an ABI to interact with the contract later and binary code.



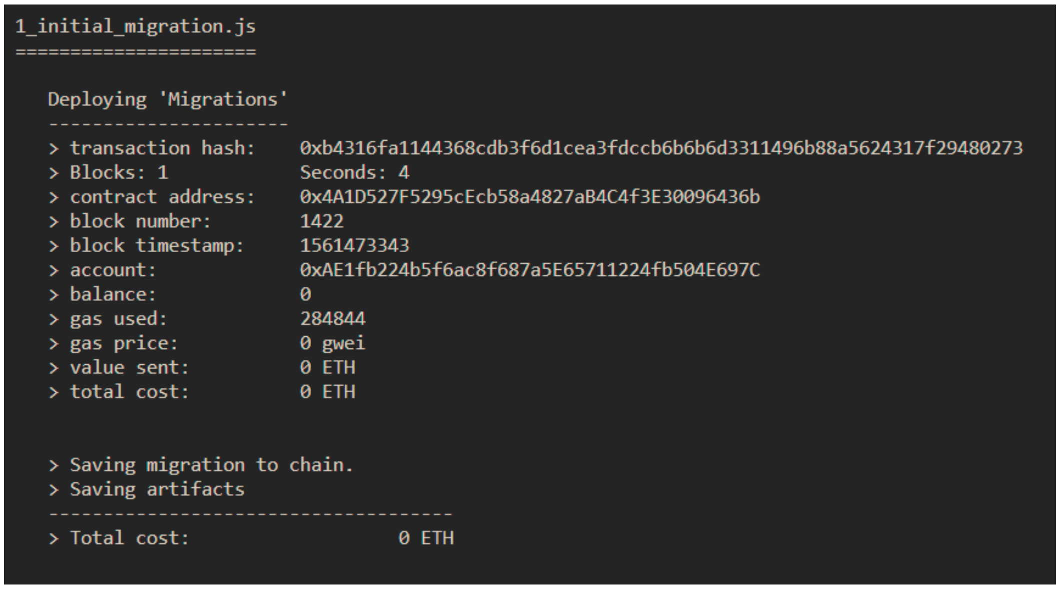
```
$ truffle compile

Compiling your contracts...
===========================
> Compiling .\contracts\HelloBlockchain.sol
> Compiling .\contracts\Migrations.sol
> Artifacts written to C:\Users\t-nisix\Documents\Development\azure-blockchain-service\build\contracts
> Compiled successfully using:
   - solc: 0.5.0+commit.1d4f565a.Emscripten.clang
```

If you want to see what it looks like, you can open artifacts inside the directory **build** > **contracts**.

Now, you're going to migrate (aka deploy) our contracts to Azure Blockchain Service. Type the following command (to indicate that we are using Azure Blockchain Service node for deployment):

```
truffle migrate –network defaultnode
```

Wait for completion.

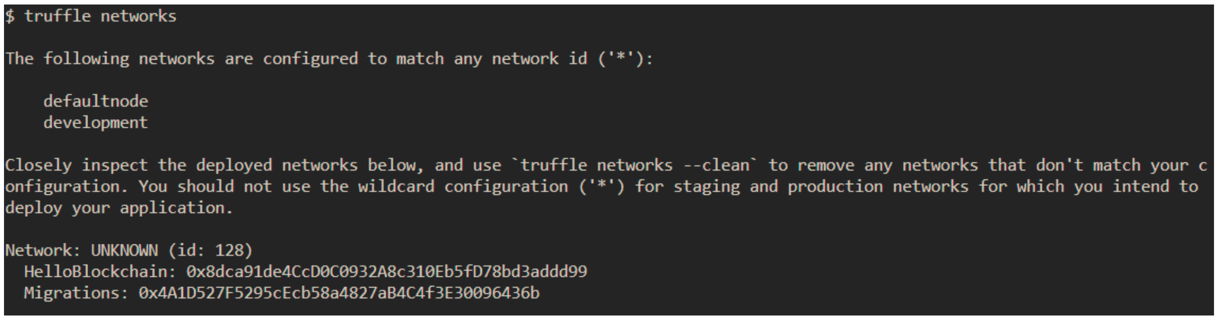


```
1_initial_migration.js
======================

   Deploying 'Migrations'
   ----------------------
   > transaction hash:    0xb4316fa1144368cdb3f6d1cea3fdccb6b6b6d3311496b88a5624317f29480273
   > Blocks: 1            Seconds: 4
   > contract address:    0x4A1D527F5295cEcb58a4827aB4C4f3E30096436b
   > block number:        1422
   > block timestamp:     1561473343
   > account:             0xAE1fb224b5f6ac8f687a5E65711224fb504E697C
   > balance:             0
   > gas used:            284844
   > gas price:           0 gwei
   > value sent:          0 ETH
   > total cost:          0 ETH


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

After that, you can also verify contract deployment and get deployed contracts list by typing:

```
truffle networks
```

Every contract deployed inside this project will be shown.

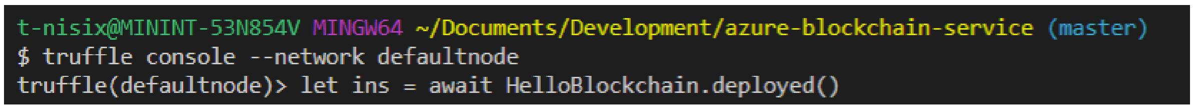

```
$ truffle networks

The following networks are configured to match any network id ('*'):

    defaultnode
    development

Closely inspect the deployed networks below, and use `truffle networks --clean` to remove any networks that don't match your c
onfiguration. You should not use the wildcard configuration ('*') for staging and production networks for which you intend to
deploy your application.

Network: UNKNOWN (id: 128)
  HelloBlockchain: 0x8dca91de4CcD0C0932A8c310Eb5fD78bd3addd99
  Migrations: 0x4A1D527F5295cEcb58a4827aB4C4f3E30096436b
```

Now that your contracts are deployed, it's time to test their functionalities. To open a console which can interact with Azure Blockchain Service, type:

```
truffle console –network defaultnode
```

And then to create an instance of the deployed contract:

```
let ins = await HelloBlockchain.deployed()
```


```
t-nisix@MININT-53N854V MINGW64 ~/Documents/Development/azure-blockchain-service (master)
$ truffle console --network defaultnode
truffle(defaultnode)> let ins = await HelloBlockchain.deployed()
```

This `ins` variable allows you to interact with the contract, as it implicitly contains the ABI and contract address. If you are curious, you can type `ins` in your console to get all variable content. Here, you're only going to use functions contained inside the instance.

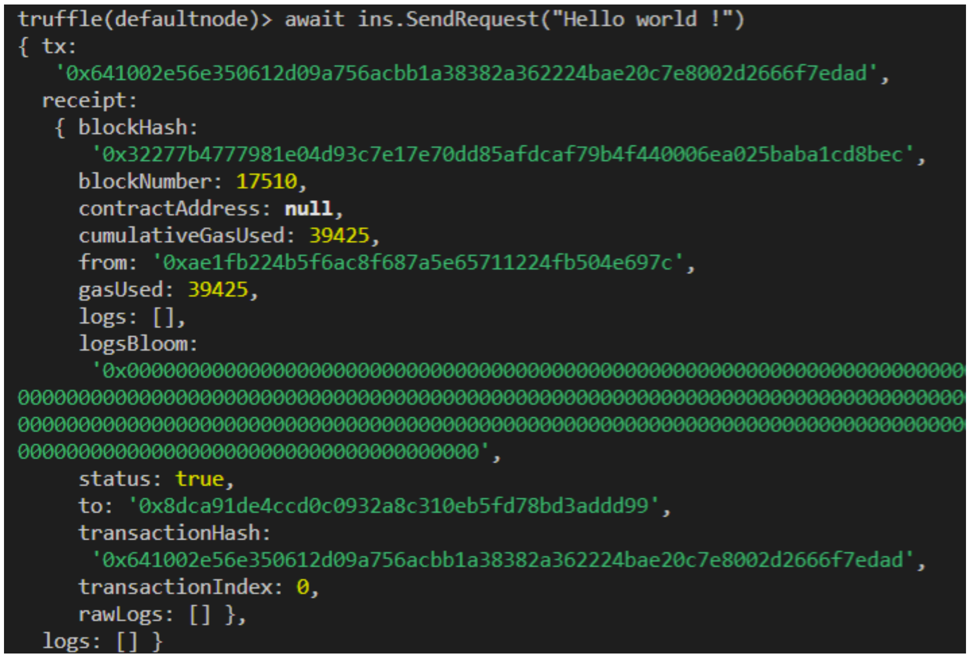
The contract is really a simple one to handle: when somebody (named as a requestor) deploys it, he/she sends with its transaction a message (named a *requestMessage* here) to store in the contract. He/She also can modify the message with the function SendRequest.

After that, someone else can send a response message to the contract, and he/she will be marked as the responder.

This is a really dummy contract, but, once again, this is just for testing purposes. To call the first function *SendRequest* and therefore send a message to the contract, type:

```
await ins.SendRequest("Hello world !")
```

If the transaction has been correctly submitted, you should get a transaction hash and other elements as a response, it means that your transaction is stored inside it.

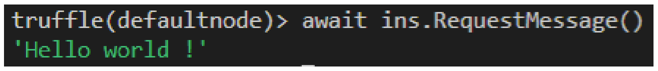


```
truffle(defaultnode)> await ins.SendRequest("Hello world !")
{ tx:
   '0x641002e56e350612d09a756acbb1a38382a362224bae20c7e8002d2666f7edad',
  receipt:
   { blockHash:
      '0x32277b4777981e04d93c7e17e70dd85afdcaf79b4f440006ea025baba1cd8bec',
     blockNumber: 17510,
     contractAddress: null,
     cumulativeGasUsed: 39425,
     from: '0xae1fb224b5f6ac8f687a5e65711224fb504e697c',
     gasUsed: 39425,
     logs: [],
     logsBloom:
      '0x000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000',
     status: true,
     to: '0x8dca91de4ccd0c0932a8c310eb5fd78bd3addd99',
     transactionHash:
      '0x641002e56e350612d09a756acbb1a38382a362224bae20c7e8002d2666f7edad',
     transactionIndex: 0,
     rawLogs: [] },
  logs: [] }
```

And therefore, you can check the *RequestMessage* variable state, by typing:

```
await ins.RequestMessage()
```

(The variable is read as you're calling a function).



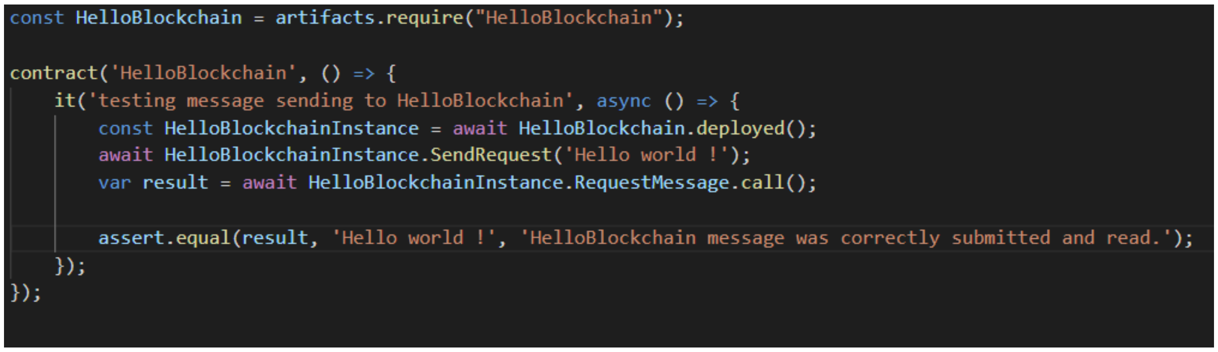
```
truffle(defaultnode)> await ins.RequestMessage()
'Hello world !'
```

Another method to test contract is with unit tests. A unit test is a method that instantiates a small portion of our application and verifies its behavior independently from other parts.

Truffle provides a set of functions to do so in JavaScript. The following command allows Truffle to test contracts by executing all units' tests present in the test directory:

```
truffle test
```

For this part, you will try to run one test which test message sending to the blockchain. The test will call the function *SendRequest*, then retrieve the variable *RequestMessage* and finally compare *RequestMessage* with a predefined string.



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

contract('HelloBlockchain', () => {
    it('testing message sending to HelloBlockchain', async () => {
        const HelloBlockchainInstance = await HelloBlockchain.deployed();
        await HelloBlockchainInstance.SendRequest('Hello world !');
        var result = await HelloBlockchainInstance.RequestMessage.call();

        assert.equal(result, 'Hello world !', 'HelloBlockchain message was correctly submitted and read.');
    });
});
```

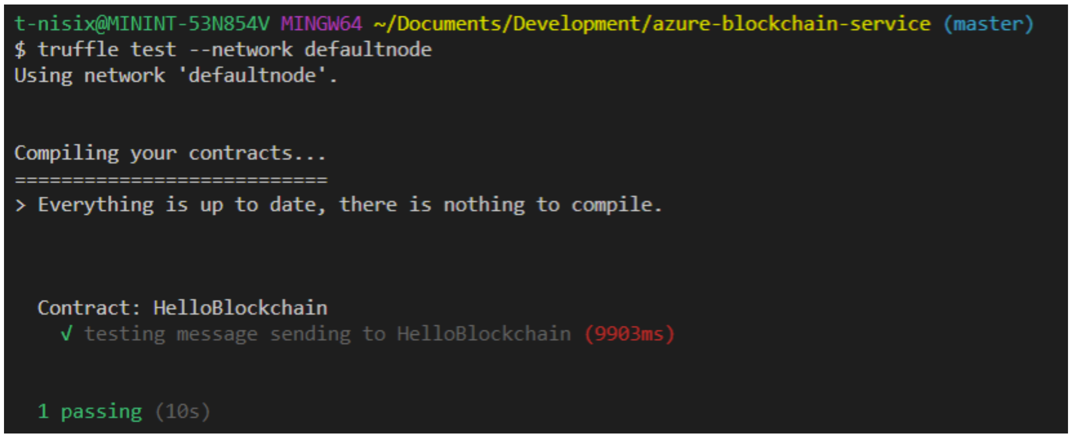
Each test is done inside a 'it' function, those functions which are embed into 'contract' functions. This is to split and order different tests and so, the result prompted into the console when executed.

Also, there are functions such as *beforeEach* or *beforeAll* which execute themselves before each test or once before all tests. This is useful to reset the contract before testing for example.

To run this test, type:

```
truffle test -network defaultnode
```

You will have to wait a bit for completion. When it's done, the test should be marked as 'pass'.



```
t-nisix@MININT-53N854V MINGW64 ~/Documents/Development/azure-blockchain-service (master)
$ truffle test --network defaultnode
Using network 'defaultnode'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



  Contract: HelloBlockchain
    √ testing message sending to HelloBlockchain (9903ms)


  1 passing (10s)
```

You're free to find and implement more tests if you want to understand the concept. But, for now, you saw a wide panel of Truffle possibilities from inside Visual Studio Code.

This concludes this guide for developers.