Building IoT-based solutions 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 how to build and deliver an end-to-end solution with Azure Blockchain Workbench. It is part of a series of guide to help developers gaining expertise on the 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/.

MICROSOFT DISCLAIMS ALL WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, IN RELATION WITH THE INFORMATION CONTAINED IN THIS WHITE PAPER. The white paper is provided "AS IS" without warranty of any kind and is not to be construed as a commitment on the part of Microsoft.

Microsoft cannot guarantee the veracity of the information presented. The information in this guide, including but not limited to internet website and URL references, is subject to change at any time without notice. Furthermore, the opinions expressed in this guide represent the current vision of Microsoft France on the issues cited at the date of publication of this guide and are subject to change at any time without notice.

Publication: June 2019
Version 1.0

About this guide

Welcome to the **Building solutions with Azure Blockchain Workbench** guide for developers.

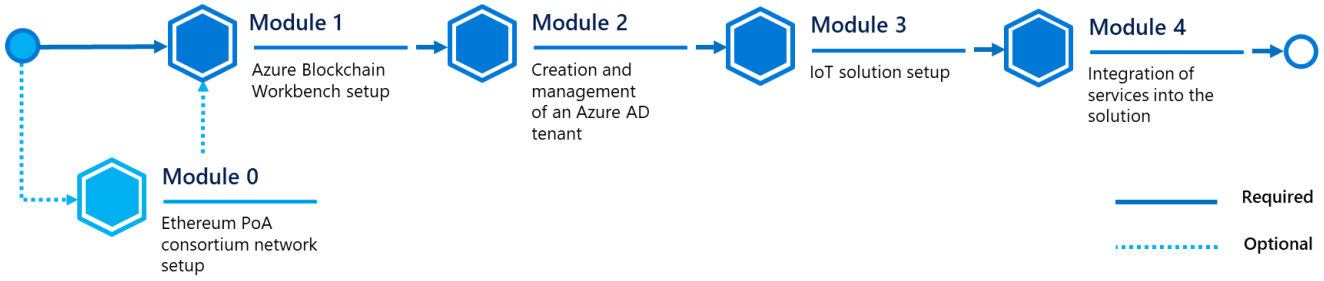
Since May 2018, Azure Blockchain Workbench[1] (ABW) is available as a public preview. The main objective of this service is to allow users to deploy a blockchain consortium network with just a few clicks, with all the related infrastructure, integrations and extensions to the other Azure cloud services and consuming apps that usually pertain to a working solution.

Such a prebuilt network and infrastructure both simplify the development and ease experimentations in order to efficiently conduct Proof-Of-Concept (PoC). As a developer, you can thus focus on decentralized applications (ĐApps) development instead of the main blockchain consortium network specifics and related problematics.

Thanks to the above mentioned thight integration in Azure, Azure Blockchain Workbench can be smoothly used with a number of others Azure services, such as Azure Active Directory (Azure AD) to manage identities and access policies to the ĐApps and the workbench, Azure Logic Apps to create powerful apps on top of Azure Blockchain Workbench through Azure Event Grids, etc. to name a few.

This self-paced guide is designed to take you from initial awareness of Azure Blockchain Workbench to being able to address a real-life scenario and thus to show you how to implemement an end-to-end supply chain solution for refrigerated products throughout a series of modules.

Each module within the guide builds on the previous. However, since the solution is intended to be fully customizable, some modules or module contents are optional. Whenever this is the case, you will an "(optional)" statement at the end of the title.



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

- Create an Ethereum Proof-Of-Authority (PoA) network within Azure (optional).
- Deploy a Azure Blockchain Workbench on the above Ethereum PoA network (optional) or on a fresh blockchain network.
- Create and manage an Azure Active Directory (Azure AD) tenant to manage members of the Workbench and applications' participants.

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

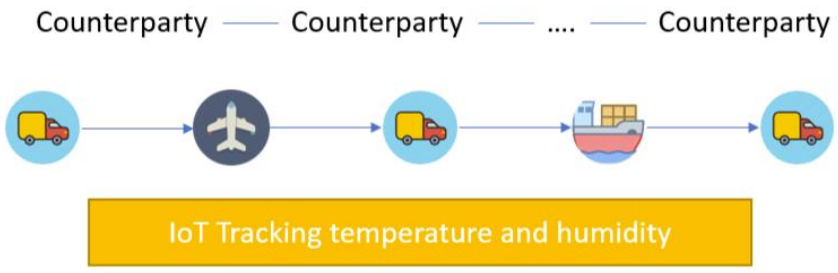
- Build an IoT-enabled solution by deploying an instance of IoT Azure Hub, a Service Bus/Queue and a Logic App, and then by programming and running (syntethic) connected devices.
- Integrate various Azure services into the solution (and notably different types of Logic Apps) as well as Dynamics 365 services.

Guide scenario

Think a while about a supply chain of refrigerated products, such as meat, frozen prepared dishes, pharmaceuticals products, etc.

In the suggested scenario, the transported product must not be exposed to high temperatures or humidity, otherwise, they will expire and they could be dangerous for human health if eaten. They need to respect certain compliance rules throughout the duration of the transport process.

To keep tracking of temperature and humidity, an Internet of Things (IoT) device with adequate sensor(s) can be used. Attached to the goods during all the transport process, its role aims at reporting the measures in near real-time (NRT) and check for the telemetry if one of those values reaches an out-of-range value. If yes, the shipment is automatically marked as "out-of-compliance" and the owner of those products is informed that they can't be sold or used anymore.



In this scenario, Azure Blockchain Workbench is used as an infrastructure backbone which connects all the supply chain participants (which are the seller and other counterparties like shippers). Even if they don't trust each other, the blockchain network is here to keep track of the (shared) truth during all the transportation process.

To do so, whenever a shipment order is created, a smart contract is created at the same time inside the blockchain. It specifies the initiating counterparty, the device responsible for measuring both the temperature and humidity inside the shipment and sending the telemetry, and eventually, the owner of the product being shipped. Optionally, observers can be declared too, in order to monitor the supply chain, for example, a government agency.

All participants can view the state and details of the contract at any point in time. The counterparty doing the transportation will specify the next counterparty in charge, and the device will ingest temperature and humidity data, which gets written to the chain. This allows the supply chain owner and supply chain observer to pinpoint which counterparty did not fulfill the compliance regulations if, at any point in the process, either the temperature or humidity requirements were not met.

Note You can find more details about the scenario and its Solidity/JSON code at this link[2]. This sample code is indeed part of the series of samples and technology demos that are already available on Github[3], but our objective here aims at illustrating the implementation of a complete customized end-to-end (e2) solution rather than building blocks.

Guide elements

In most 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.
- **Sample applications, and files**. A downloadable version of the application that you will use in this guide, and other files you will need as well as part of the wakkthrough. **Please go to https://aka.ms/ABWDevGuideSamples to download all necessary assets.**

Guide prerequisites

To successfully set up this solution in this guide, you will need:

- A Microsoft account[4].
- An Azure subscription. If you don't have an Azure subscription, create a free account[5] before you begin.

  Nothing you do in this guide should result in any billing to your Azure subscription.

- The project which contains the code for every part of this guide, you can download it or clone it here from Github[6].

Optionally, you may need additional bits for each module, and the related step-by-step directions, in this guide. Check out the tab below if you want to know in advance the requirements needed for each module.

Note Some requirements can be redundant depending of parts choosed.

[2] Refrigerated Transportation Sample Application for Azure Blockchain Workbench: https://github.com/Azure-Samples/blockchain/blob/master/blockchain-workbench/application-and-smart-contract-samples/refrigerated-transportation/readme.md

[3] Applications and Smart Contract Samples: https://github.com/Azure-Samples/blockchain/tree/master/blockchain-workbench/application-and-smart-contract-samples

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

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

[6] Refrigerated Supply Chain samples: https://aka.ms/ABWDevGuideSamples

Module/Part	Requirements
Ethereum PoA Consortium Setup	• Any MetaMask compatible browser like Mozilla Firefox - You can download it here[7] -.
IoT solution setup > Simulated device	• Node.js v4 or later. If you don't installed it yet, you can download it here[8]. • A code editor of your choice (like Visual Studio[9] or Visual Studio Code[10]).
IoT solution setup > MXChip IoT DevKit (optional)	• A physical MXChip IoT DevKit[11]. • ST-Link drivers, available here for Windows[12], **Note** For Linux users, you will have to type command lines available later in this guide, and for MacOS users no drivers are required. • Arduino IDE, which can be found here[13]. • Visual Studio Code. See above.
IoT solution setup > Azure Sphere (optional)	• A physical Azure Sphere[14] device such as the Azure Sphere MT3620 Development Kit (that can be ordered here[15]), along with the so-called Grove Starter Kit for Azure Sphere MT3620 Development Kit (that can be ordered here[16]). • Visual Studio. See above. • Azure Sphere SDK Preview for Visual Studio, available here[17].
IoT solution setup > Raspberry Pi (optional)	• A physical Raspberry Pi[18] 3 or above device. • Peripherics for the Raspberry Pi (keyboard, mouse, ethernet access and screen). • An empty MicroSD card with enough storage (8GB+) to install Raspbian OS. • A disk image of Raspbian OS, which can be downloaded here[19].

[7] Mozilla Firefox: https://www.mozilla.org/en-US/firefox/new/

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

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

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

[11] MXChip IoT DevKit: http://www.mxchip.com/az3166

[12] STSW-LINK009: https://www.st.com/en/development-tools/stsw-link009.html

[13] Arduino: https://www.arduino.cc/en/Main/Software

[14] Azure Sphere: https://azure.microsoft.com/en-us/services/azure-sphere/

[15] Azure Sphere MT3620 Development Kit: https://www.seeedstudio.com/Azure-Sphere-MT3620-Development-Kit-EU-Version-p-3134.html

[16] Grove Starter Kit for Azure Sphere MT3620 Development Kit: https://www.seeedstudio.com/Grove-Starter-Kit-for-Azure-Sphere-MT3620-Development-Kit.html

[17] Azure Sphere SDK Preview for Visual Studio: https://aka.ms/AzureSphereSDKDownload

[18] Raspberry Pi: https://www.raspberrypi.org/products/

[19] Raspbian: https://www.raspberrypi.org/downloads/

	• A software to flash disk images, like Etcher[20]. • Grove SHT31 humidity and temperature sensor (available here[21]) • 4 male-to-female breadboard cables.
Integration of services > Connected Field Services	• An Office 365 subscription. • A Dynamics 365 subscription. **Note** if you don't have those licences, there is a Office 365 and Dynamics 365 30-days trial available that can be used instead. Follow the guide to find out how to setup it.
Integration of services > React web app	• Node.js v4 or later. See above. • A code editor of your choice (such as Visual Studio or Visual Studio Code).

[20] balenaEchter: https://www.balena.io/etcher/

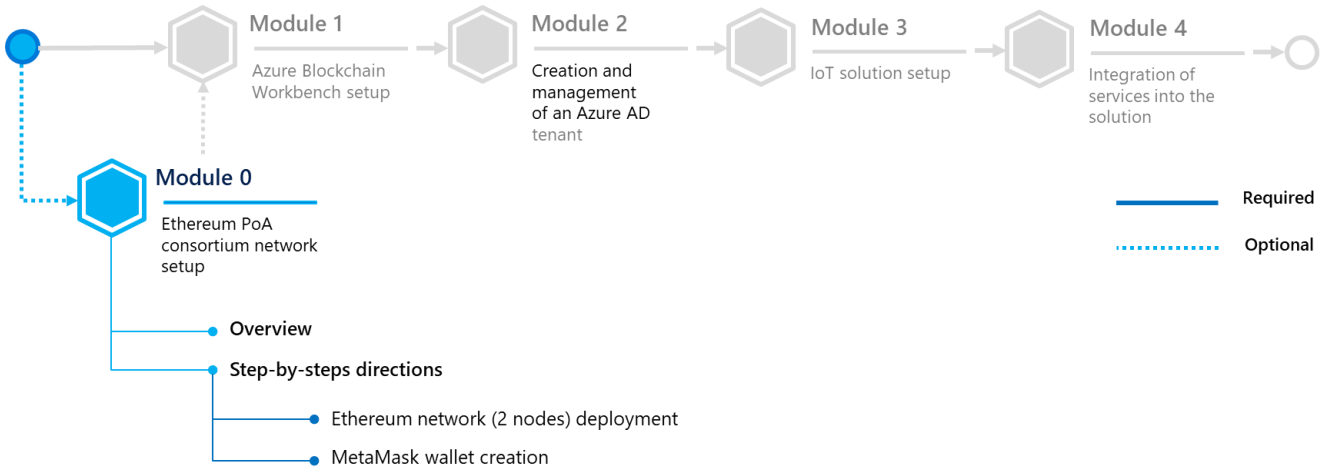
[21] Grove – Temperature & Humidity Sensor: https://store.arduino.cc/grove-temperature-humidity-sensor-sht31

Module 0: Ethereum Proof-of-Authority Consortium network setup (optional)

Overview

When you create an Azure Blockchain Workbench, you can choose to set up a new network which will be an Azure Blockchain Service[22] node. If this is your goal and you don't want to bother about the network, feel free to skip this part and read the next section.

However, if you want to deploy this set of nodes by yourself (or if you already have and existing Ethereum Proof of Authority[23] (PoA) or Ethereum Quorum[24] blockchain) to better understand how it works or to customize the network, you can follow the directions of this section. They will guide you to set it up and then, get the Endpoint URL which will be then used in the next module by Azure Blockchain Workbench.



In this module, you will learn to:

- Deploy an Ethereum PoA network.
- Create a key wallet to access to your blockchain by using MetaMask.

Step-by-step directions

This module covers the following activities:

1. Deploying the consortium network (optional).

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

[23] Ethereum Proof-of-Authority Consortium: https://docs.microsoft.com/fr-fr/azure/blockchain/templates/ethereum-poa-deployment

[24] Go Quorum: https://www.goquorum.com/

2. Installing Metamask.
3. Completing the deployment of the consortium network.

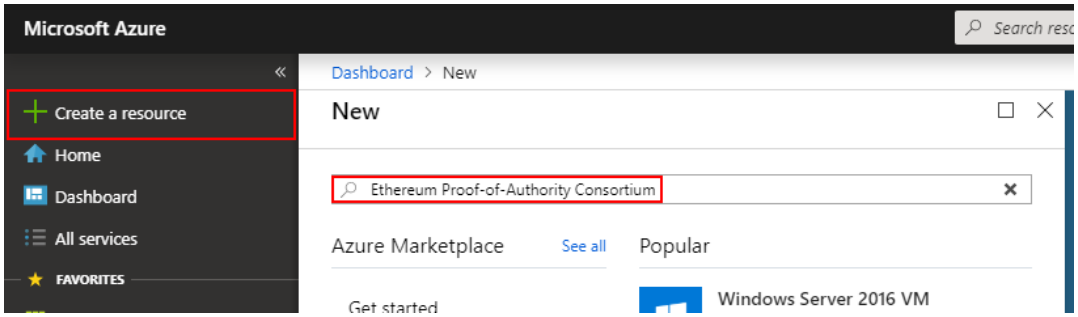
Each activity is described in order in the next sections.

Deploying the consortium network

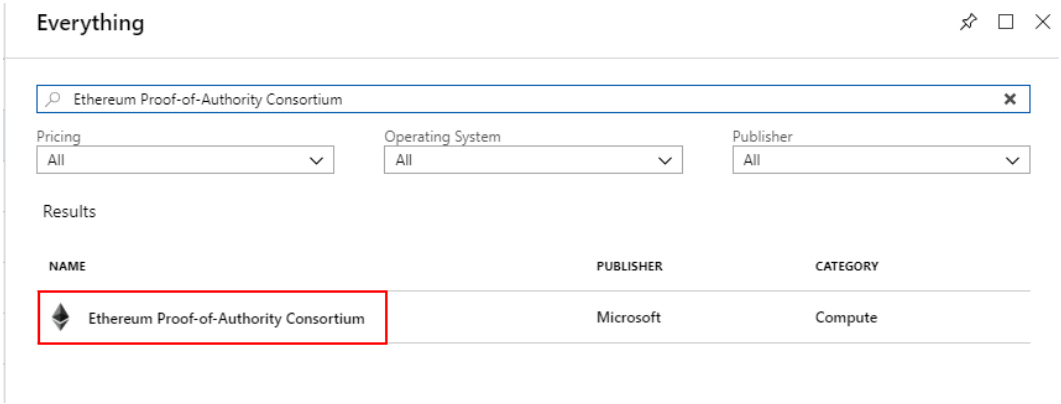
First of all, you need to sign in into the Azure portal in order to create an Ethereum Proof-of-Authority (PoA) Consortium.

Perform the following steps:

1. Sign in to the Azure portal[25].
2. Select your account in the top-right corner, and switch to the desired Azure AD tenant where you want to deploy your Ethereum Proof-of-Authority (PoA) consortium.
3. In the left pane, select **Create a resource**. Search for "*Ethereum Proof-of-Authority Consortium*" in the **Search the Marketplace** search bar.

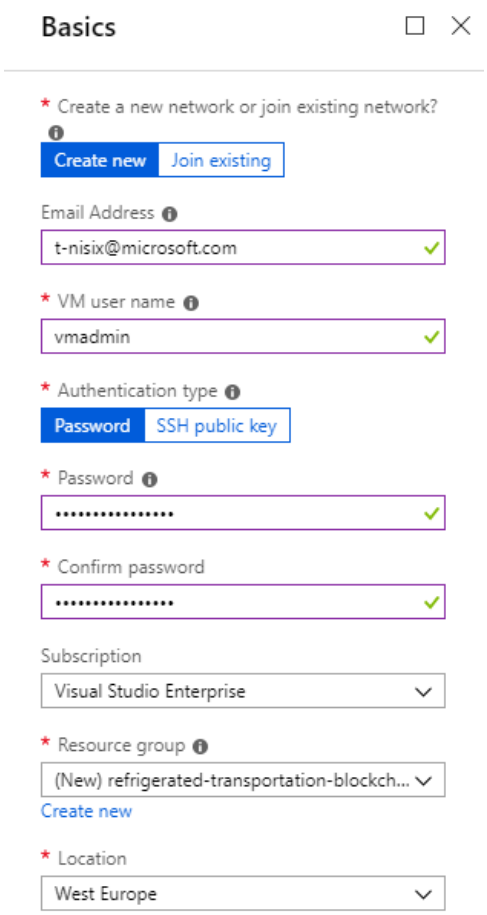


4. Select **Ethereum Proof-of-Authority Consortium**.



5. Select **Create**.
6. Complete the basic settings.

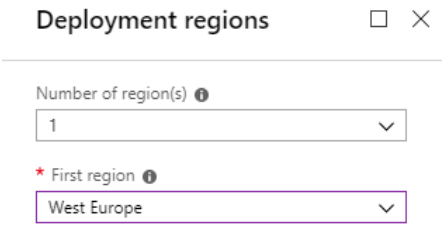
[25] Azure portal: https://portal.azure.com/



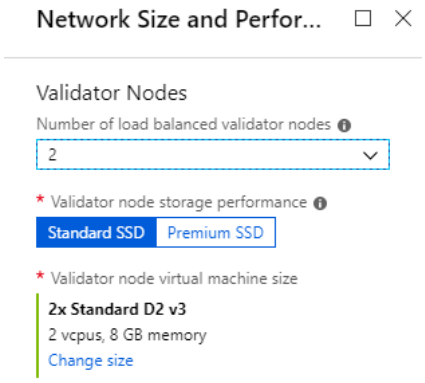
Setting	Description
Create new or Join existing	Select whether if you want to create a new Ethereum PoA blockchain network or if you want to join an existing network.
Email Address	Specify an email address that you will use to receive a confirmation email with additional information when the blockchain deployment will complete.
VM user name	Specify the user name that you use to connect as an administrator for all the virtual machines (VMs).
Authentication type	Select if you want to use a password or a SSH key for connecting to VMs.
Password	Depending on the above setting, specify the password being used for connecting to VMs.
SSH	Depending on the above **Authentication type** setting, specify the RSA public key in the single-line format beginning with "*ssh-rsa*" (you can use instead the multi-line PEM format -.

	Note You can generate SSH keys using `ssh-keygen` on Linux and MacOS, or by using PuTTYGen[26] on Windows. More information on SSH keys, see article HOW TO USE SSH KEYS WITH WINDOWS ON AZURE[27].
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 in which you want to deploy the framework.

7. Complete the **Deployment regions** settings by choosing the number of region(s) to deploy VMs into and specify the regions below this choice. Keep the number of regions by default if you don't have any specific need.



8. Complete the **Network Size and Performance** settings by selecting the number of nodes needed and their properties. Keep it by default if you don't have any specific requirement.



At this point, you will arrive on **Ethereum** settings. This section will require you to have a public Ethereum address. This address is a 64-character hex string generated, which will represent your identity on the blockchain.

In order to get it, you will need to install MetaMask.

---

[26] PuTTYGen: https://www.puttygen.com/

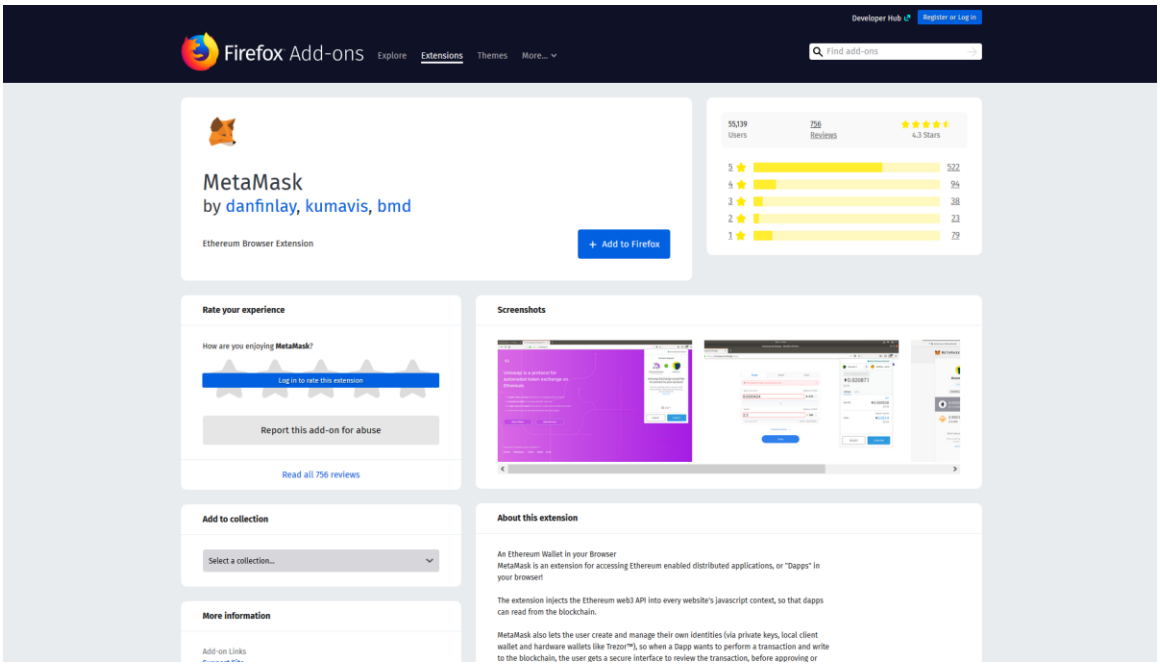
[27] HOW TO USE SSH KEYS WITH WINDOWS ON AZURE: https://docs.microsoft.com/en-us/azure/virtual-machines/linux/ssh-from-windows

Installing Metamask

MetaMask is a browser extension compatible with Edge (Chromium), Chrome, Firefox, Opera and Brave. In this guide, you will install it on Firefox, for the sake of this walkthrough but the extension can work on any browser previously mentioned.

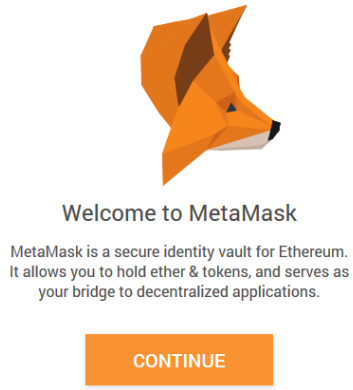
Perform the following steps:

1. Open a new tab if you are on a compatible browser or choose another one if not.
2. Go on the [MetaMask website](https://metamask.io/)[28], 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**.



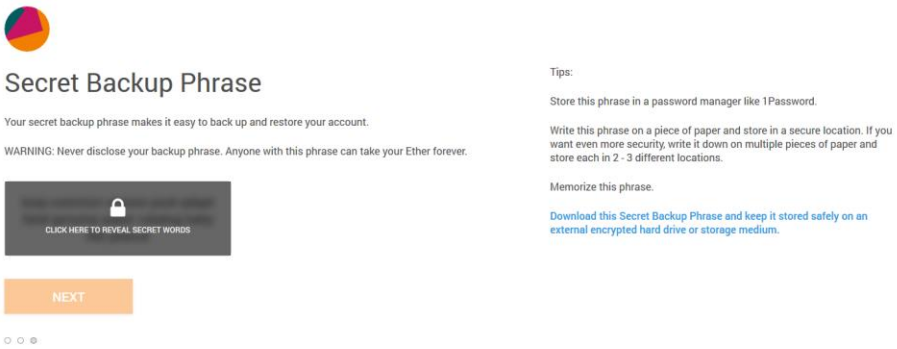
3. After that, the extension will open up, and prompt you to begin the installation. Click on **Continue.**

[28] MetaMask: https://metamask.io/

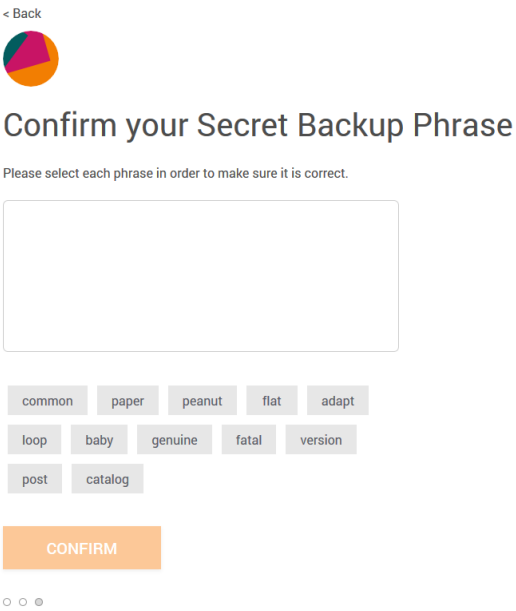


4. It will ask you to create a password. Type one of your choice and click on **Create,** then **Next** and eventually accept the **Terms of Use, Privacy Notice & Phishing Warning**.
5. You will arrive on the **Secret Backup Phrase** page. Click on the lock to reveal the secret words. This phrase REALLY important, in so far as it represents your Ethereum Account. Indeed, thanks to a cryptographic algorithm, you can generate your Ethereum Wallet from this phrase.

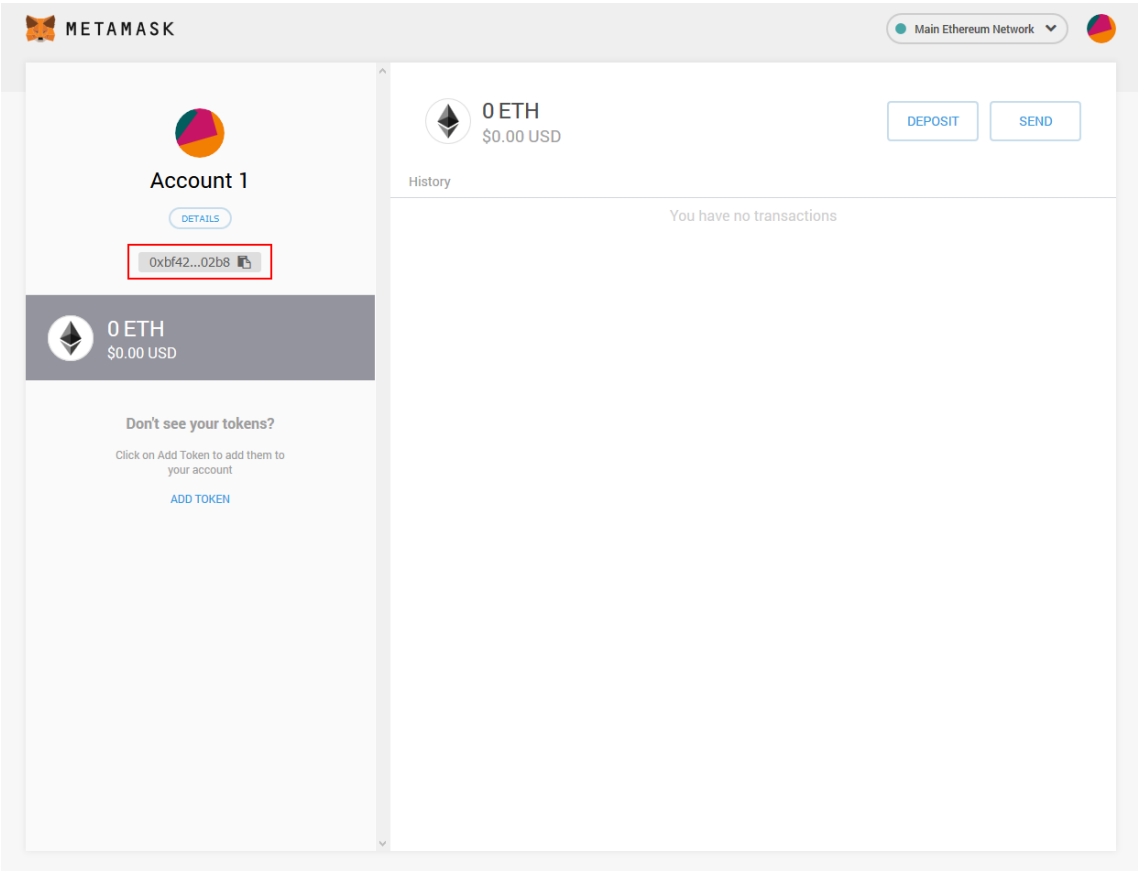
Note if you want to use this wallet in the future and have the possibility to retrieve your account in any case, **please keep this phrase SECRET** as anybody who has this phrase is able to connect to your account**, and save it somewhere** like on a paper or in a password manager.



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



7. Now that the installation is completed, you just have to copy the generated address by clicking on it inside the application. Remember that this will be your identity on Azure Ethereum PoA Consortium network.

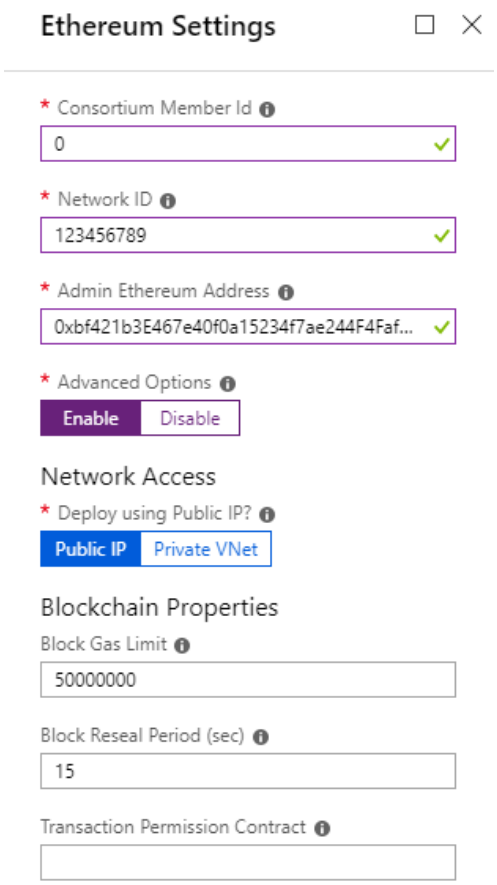


Note In brief, MetaMask act as an account manager for Ethereum networks. If you only want to use Azure Blockchain Workbench, you will not have to use it but if, in any case, you want to access to the network administration, you will have to use the browser where MetaMask is installed to access the administration panel by signing in with MetaMask.

Completing the deployment of the consortium network

Perform the following steps:

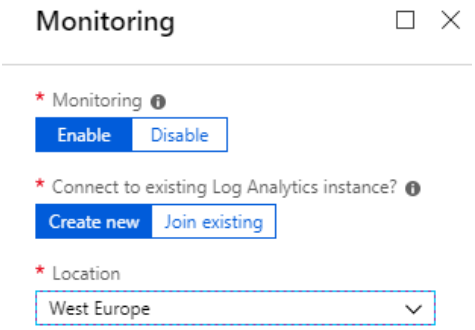
1. Go back on your **Ethereum Settings** tab and complete it.



Parameter name	Description	Allowed values
Consortium Member ID	Specify the ID associated with each member participating in the consortium network. It is 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 a note of the value of this parameter since you will need to share it with other joining members to ensure there's no collision.	0-255

Network ID	Specify 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
Admin Ethereum Address	Specify the Ethereum account address that is used for participating in PoA governance. Paste the address generated by MetaMask.	42 alphanumeric characters starting with 0x
Advanced Options	Enable or disable Advanced options for Ethereum settings	Enable or Disable
Public IP (Advanced Options = Enable)	Select this option to deploy 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
Block Gas Limit (Advanced Options = Enable)	Set the starting block gas limit of the network	Any numeric
Block Reseal Period (sec)	Set 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
Transaction Permission Contract (Advanced Options = Enable)	Bytecode for the Transaction Permissioning contract. Restricts smart contract deployment and execution to a permissioned list of Ethereum accounts.	Contract bytecode

2. Complete the **Monitoring Settings.** 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.



Parameter name	Description	Allowed values
Monitoring	Enable or disable monitoring	Enable or Disable
Connect to existing Azure Monitor logs	Create a new Azure Monitor logs instance or join an existing instance	Create new or Join existing
Monitor Location (Connect to existing Azure Monitor logs = Create new)	Specify the region where the new Azure Monitor logs instance will be deployed	All Azure Monitor logs regions
Existing Log Analytics Workspace Id (Connect to	Specify the Workspace ID of the existing Azure Monitor logs instance	

existing Azure Monitor logs = Join Existing)		
Existing Log Analytics Primary Key (Connect to existing Log Analytics = Join Existing)	Specify the primary key used to connect to the existing Log Analytics instance	

3. The summary of your choices should appear now. Wait for the automatic validation, then click on **OK** if everything is correct. On the next window, click on **Create**.

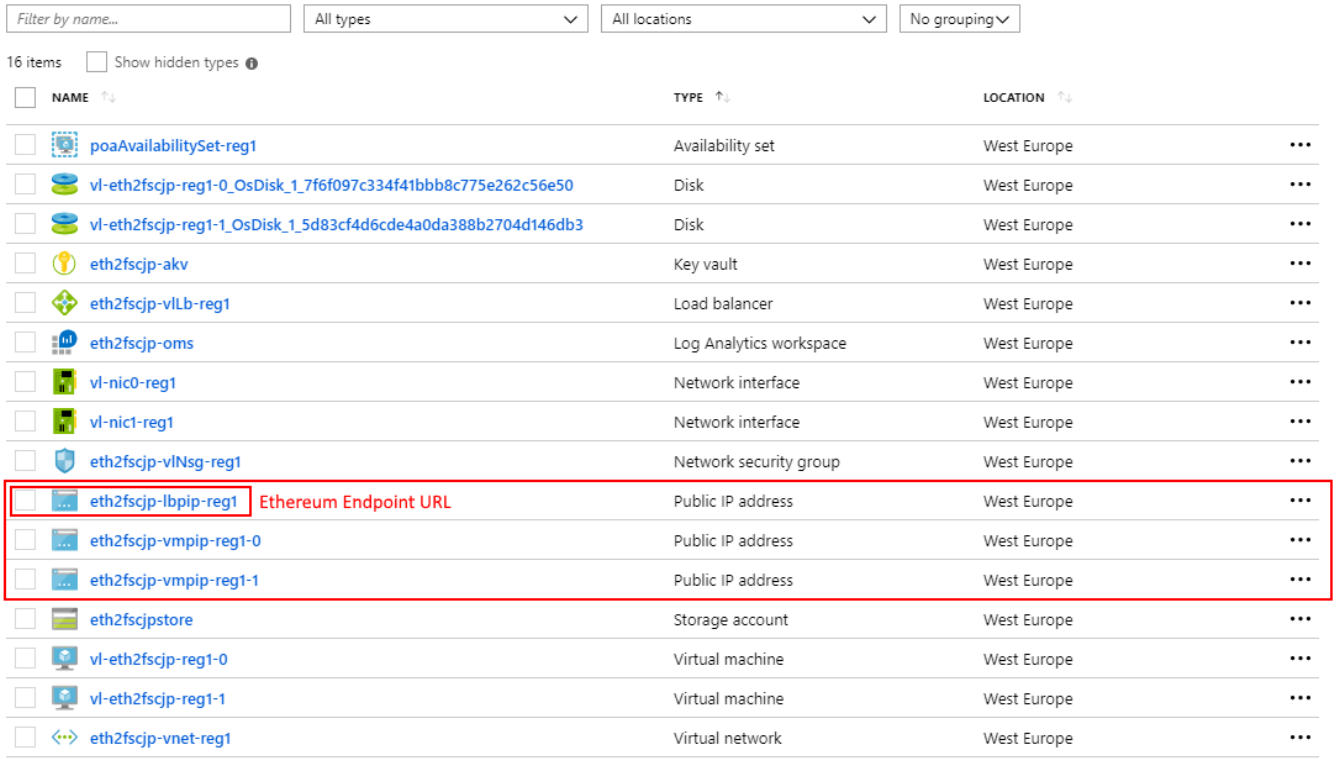
The deployment will take up to 50 minutes.

When completed, you will receive an email on the address specified in the very first step, describing the various parameters of your blockchain.

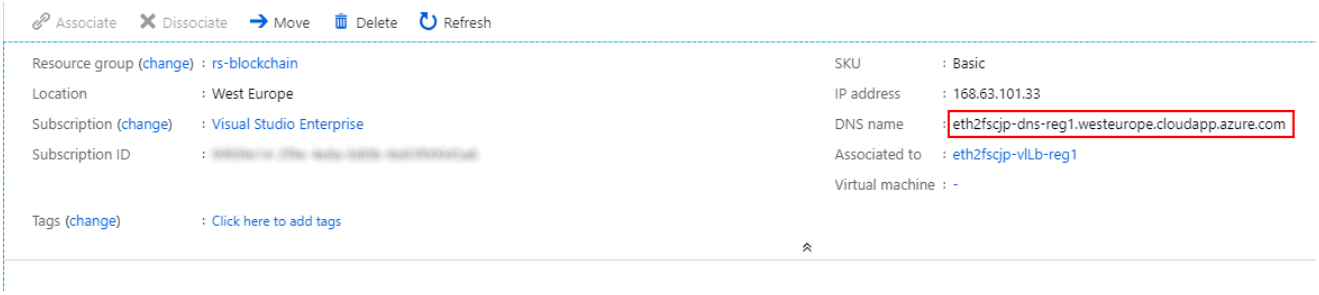
Now that your Ethereum network is online, you are ready to get the Ethereum Endpoint URL, which will be used later by the workbench.

On the Azure portal, go on the **Resource Groups** section, then click on the name of the resource group you've created for your blockchain. The list of all the blockchain resources will appear, showing different kind of resources.

To find the address, you need to check the **Public IP address** resources. Indeed, one of those addresses corresponds to the Ethereum Endpoint URL, and the others are the URLs of your nodes composing the network. The one that you're searching is the one whose name finished by *"-lbpip-reg1"*. Search and click on it.



The URL is displayed in front of the **DNS name** field. Save the URL, you will use it when you will setup Azure Blockchain Workbench. You are ready to jump on the next step.


Associate
Dissociate
Move
Delete
Refresh
Resource group (change) : rs-blockchain
Location : West Europe
Subscription (change) : Visual Studio Enterprise
Subscription ID :
Tags (change) : Click here to add tags
SKU : Basic
IP address : 168.63.101.33
DNS name : eth2fscjp-dns-reg1.westeurope.cloudapp.azure.com
Associated to : eth2fscjp-vlLb-reg1
Virtual machine : -

Module 1: Azure Blockchain Workbench setup

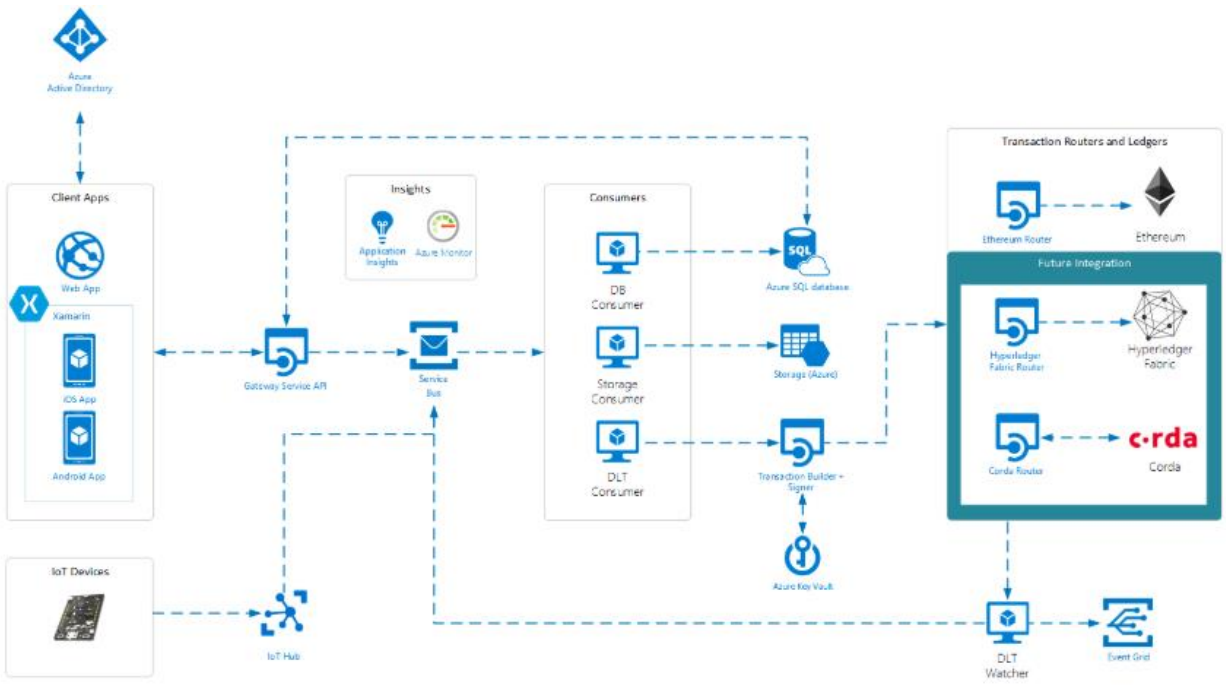
Overview

Whether you already created an Ethereum PoA consortium network or not, you're now ready to deploy and set up the cornerstone of your end-to-end solution: Azure Blockchain Workbench.

As suggested before, this service is able to plug itself to an existing Ethereum PoA or Quorum network or to create a new one, and, as such, adds a high-level layer on top of the blockchain. With Azure Blockchain Workbench, you indeed benefit from a graphical UI that allows you to quickly deploy new decentralized applications (ĐApps), or use the existing ones.

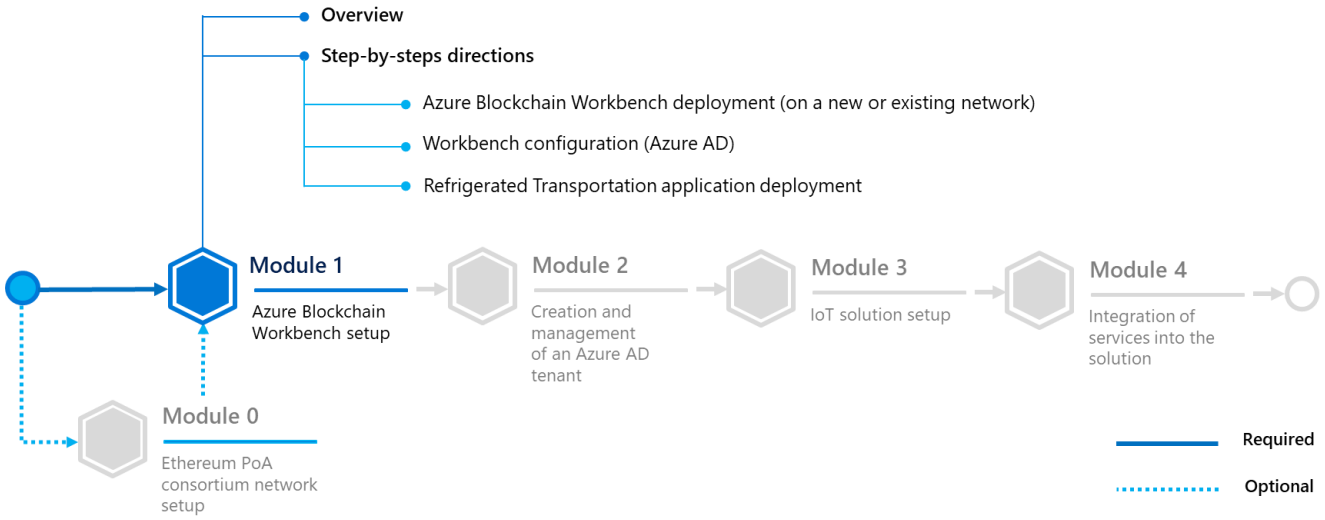
You can also connect existing applications to the Workbench API, Workbench SQL Database or Egress Queue, which allows your applications to request Workbench data.

The following architecture scheme represents Azure Blockchain Workbench, which is, as you can see, a combinaison of many Azure services.



This really simplifies the development of new blockchain applications, because you don't have to bother about the blockchain network infrastructure, key management, etc.

You can focus on creating new decentralized applications (ĐApps).



In this module, you will learn to:

- Deploy an Azure Blockchain Workbench (on a new or already existing network).
- Configure the link between Azure AD and your Workbench, using Azure Shell (Azure CLI).
- Deploy a Solidity application into Azure Blockchain Workbench.

Step-by-step directions

This module covers the following activities:

1. Creating an Azure Blockchain Workbench.
2. Configuring your Azure Blockchain Workbench.
3. Deploying the supply chain of refrigerated products application.

Each activity is described in order in the next sections.

Creating an Azure Blockchain Workbench

Creating an instance of Azure Blockchain Workbench

Perform the following steps:

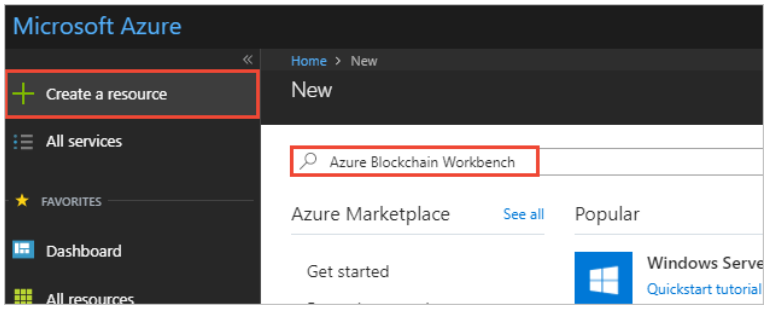
> **Note** The whole documentation about Azure Blockchain Workbench is available at this link[29] if needed. This guide is inspired by this documentation and summarizes it.

1. Sign in to the Azure portal.

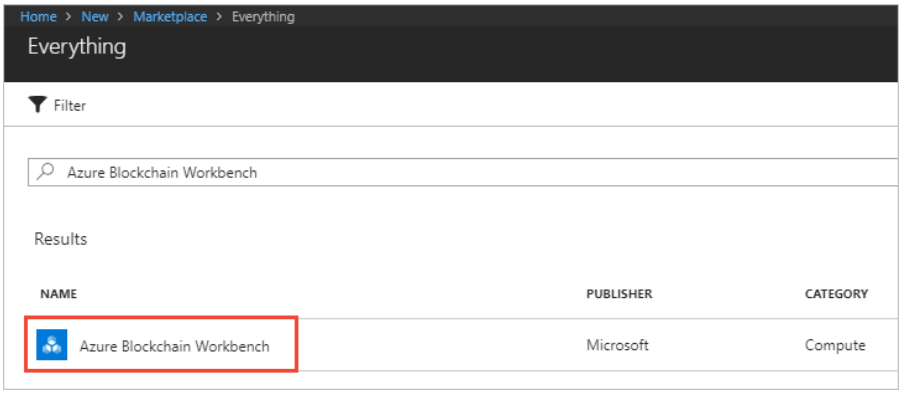
[29] AZURE BLOCKCHAIN WORKBENCH DOCUMENTATION: https://docs.microsoft.com/en-us/azure/blockchain/workbench/

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.

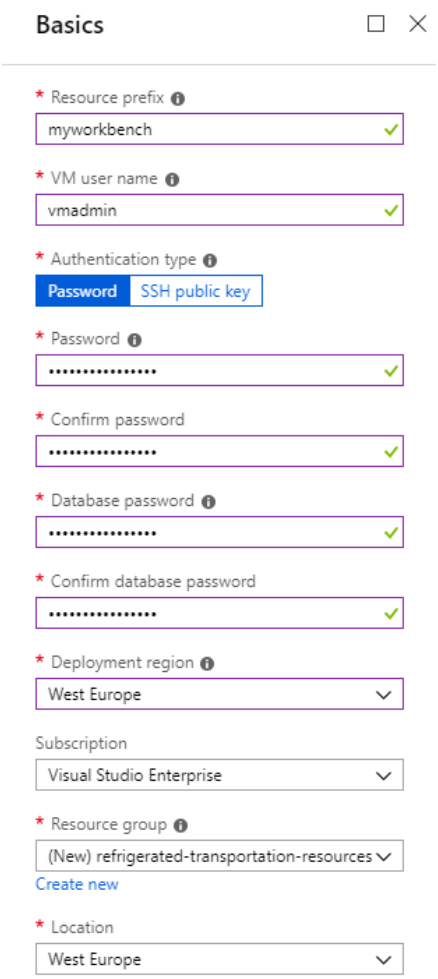


4. Select **Azure Blockchain Workbench**.



5. Select **Create**.

6. Complete the basic settings.



Setting	Description
Resource prefix	Specify a short unique identifier for your deployment. This value is used as a base for naming resources.
VM user name	Specify the user name that you use to connect as an administrator for all the virtual machines (VMs).
Authentication type	Select if you want to use a password or a SSH key for connecting to VMs.
Password	Depending on the above setting, specify the password being used for connecting to VMs.
SSH	Depending on the above **Authentication type** setting, specify the RSA public key in the single-line format beginning with "*ssh-rsa*" (you can use instead the multi-line PEM format -.

	Note You can generate SSH keys using `ssh-keygen` on Linux and MacOS, or by using PuTTYGen[30] on Windows. More information on SSH keys, see article HOW TO USE SSH KEYS WITH WINDOWS ON AZURE[31].
Database password / Confirm database password	Specify the Azure subscription you wish to use for your deployment.
Deployment region	Specify where to deploy Azure 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 in which you want to deploy the framework.

7. Select **OK** to finish the basic setting configuration section.

As of this writing, an Azure Blockchain Workbench need to rely on an Ethereum Proof-of-Authority (PoA) or a Quorum network.

If you don't already have one, you can choose to create a Workbench on a new blockchain network by choosing the option **Create new** in **Advanced Settings** and follow the next point DEPLOYING A NEW BLOCKCHAIN NETWORK. It will create a Azure Blockchain service which contains an Ethereum Quorum node.

But if you already created a network, you can choose **Use Existing** and skip the next point to read part DEPLOYING OVER A PRE-EXISTING BLOCKCHAIN NETWORK instead.

Keep in mind that the deployment can take up to 90 minutes regardless of your choice.

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.

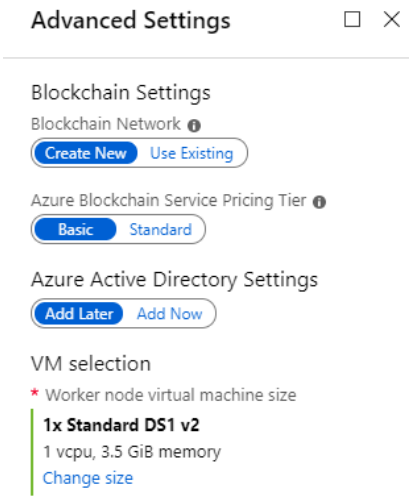
Deploying a new blockchain network

The **Create New** option creates a Azure Blockchain Service which contains a Quorum node.

[30] PuTTYGen: https://www.puttygen.com/

[31] HOW TO USE SSH KEYS WITH WINDOWS ON AZURE: https://docs.microsoft.com/en-us/azure/virtual-machines/linux/ssh-from-windows

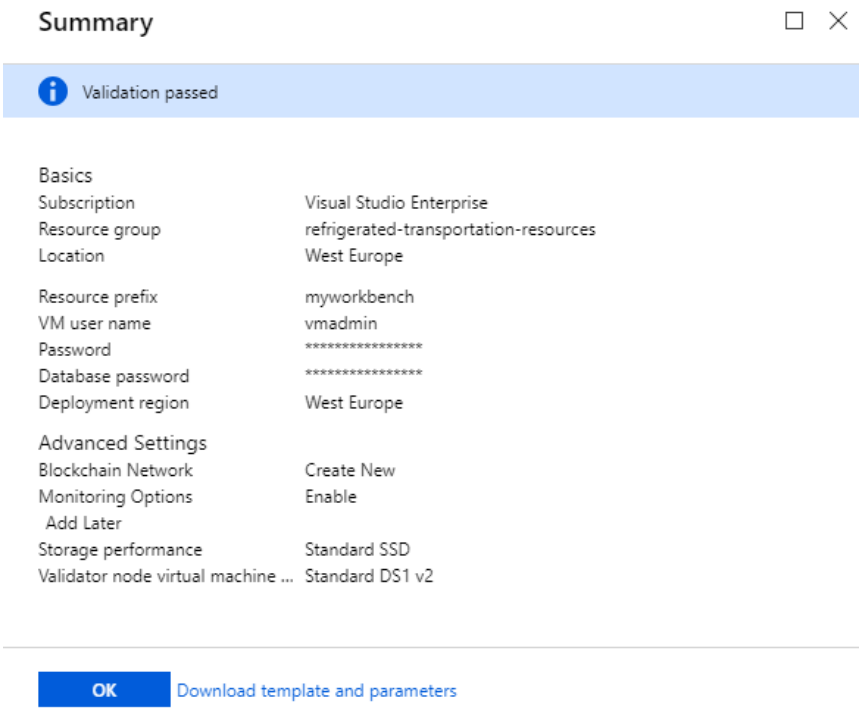
Perform the following steps:



1. Select **Create New**.
2. Specify the following settings.

Setting	Description
Monitoring	Choose whether you want to enable Azure Monitor to monitor your blockchain network
Azure Active Directory Settings	Choose **Add Later**.
VM selection	Choose the preferred 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 an Azure free tier.

1. Select **OK** to complete the **Advanced Settings**.
2. Review the summary to verify your parameters are accurate, and then click **OK** to launch the deployment.



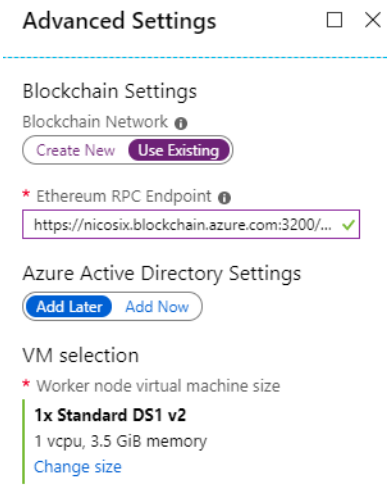
Deploying over a pre-existing blockchain network

The **Use Existing** option allows you to specify an Ethereum PoA or Quorum blockchain network endpoint.

If you created an Ethereum Proof-of-Authority (PoA) Consortium network or a Azure Blockchain Service on Azure before, you can use it here. As already outlined, you can also use your own external Ethereum network if it follows the following requirements:

- The endpoint must be an Ethereum Proof-of-Authority (PoA) or Quorum 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.

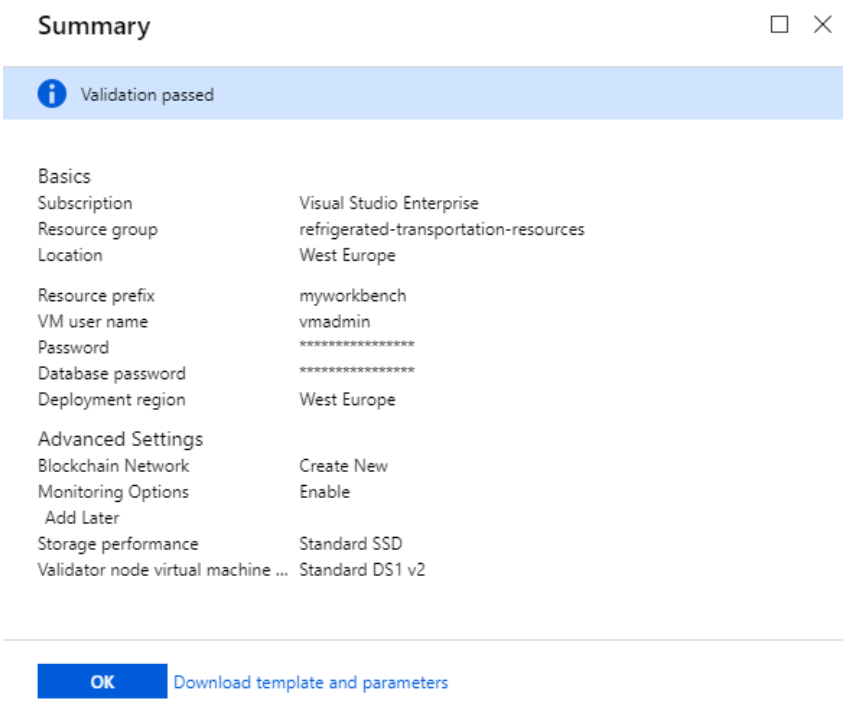
Perform the following steps:



1. Select **Use Existing**.
2. Specify the following settings.

Setting	Description
Existing network	Specify the URL of your Ethereum RPC Endpoint. If there isn't any port at the end of the URL, it will add ":8540" to it. It corresponds to the default port of an Ethereum endpoint.
Azure Active Directory settings	Choose **Add Later**.
VM selection	Choose the preferred 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 an Azure free tier.

3. Select **OK** to complete the **Advanced Settings**.
4. Review the summary to verify your parameters are accurate, and then click **OK** to launch the deployment.



Configuring your Azure Blockchain Workbench

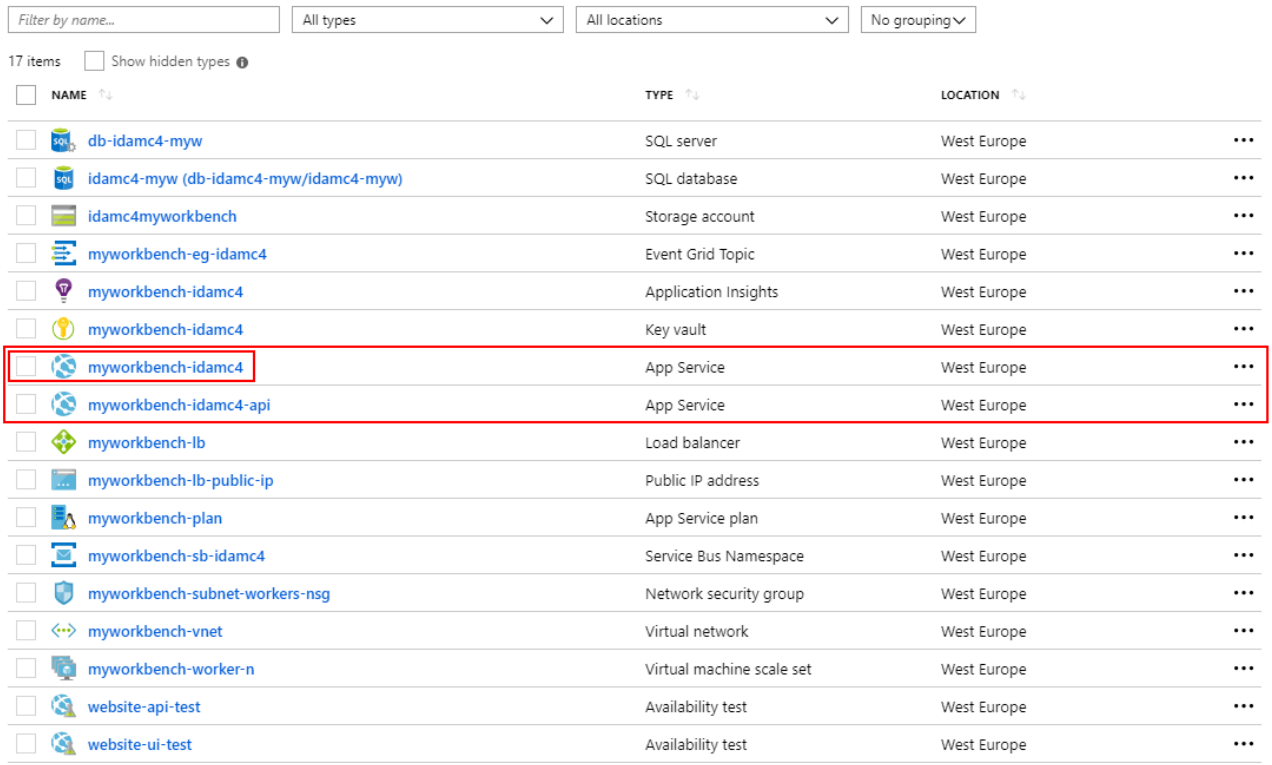
Linking your Workbench to your Azure AD tenant

Perform the following steps:

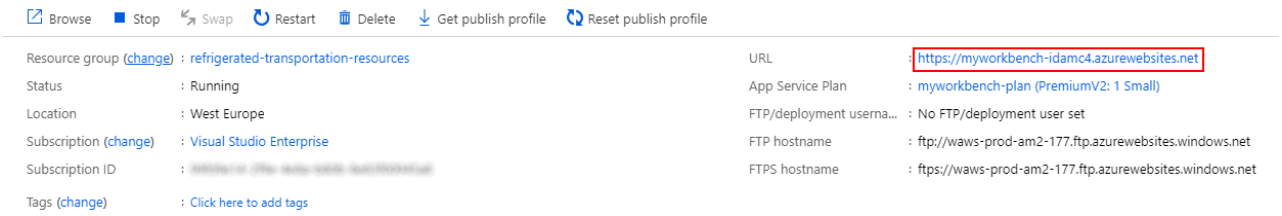
1. Retrieve your Workbench URL.

   To get the URL of your deployed workbench, go on the **Resource Groups** section, then click on the name of the resource group you've created for your workbench. The list of all the workbench resources will appear, showing different kind of resources.

   To find the address, you need to search for the **App Service** resources. One of the two App service resources has its name that ends up with "-api". This resource will be useful later if you want to use the API, the other one is the workbench. Click on it.



The URL of the workbench is displayed at the top, in front of the URL field. Click on it to go to your workbench.



2. Add Azure Blockchain Workbench as an Azure AD application.

   At this point, you will arrive on the workbench home page. It prompts you to launch Azure Cloud Shell and run a command. The goal of this command is to link your Azure AD tenant to Azure Blockchain Workbench. Once completed, you will have the ability to manage the users of your workbench by adding or removing people in your own Azure AD tenant.

   To run the command, click on **Launch Cloud Shell**, and a console prompt will open up in a new tab. Copy the command in the workbench tab and paste it in the Shell.

Welcome to Azure Blockchain Workbench

Lets get your instance setup

Step 1. Launch Cloud Shell and run the following command

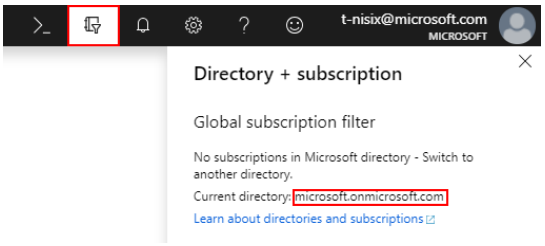
Azure Cloud Shell | Copy | Launch Cloud Shell

cd; Invoke-WebRequest -Uri https://aka.ms/workbenchAADSetupScript -OutFile workbenchAADSetupScript.ps1; ./workbenchAADSetupScript.ps1 -SubscriptionID [redacted] -ResourceGroupName refrigerated-transportation-resources -DeploymentId idamc4

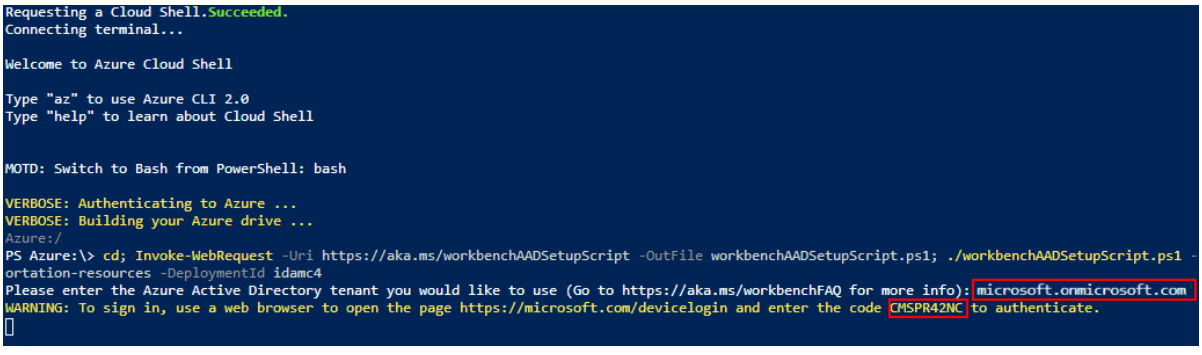
Step 2. Refresh your browser

This will open your Workbench

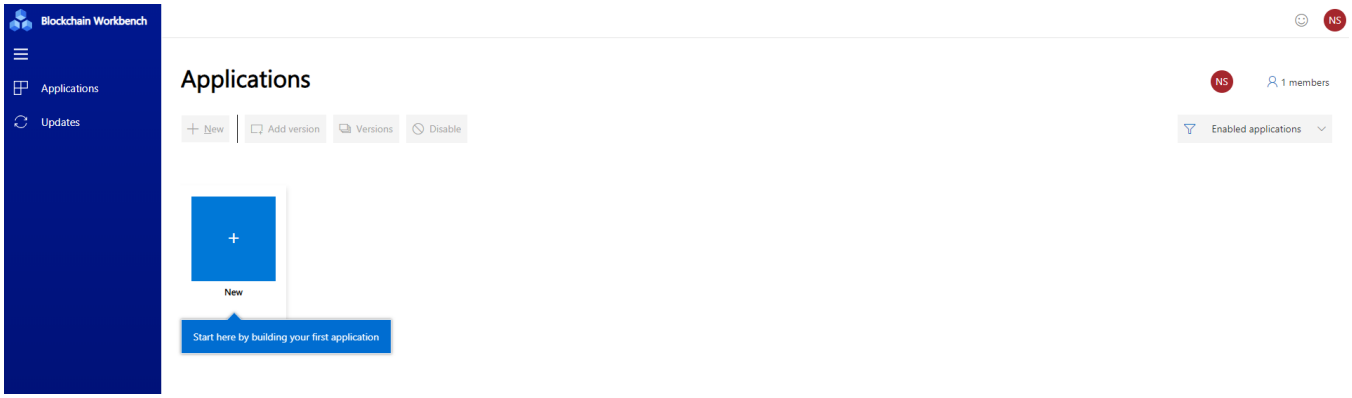
It will prompt you for a tenant name. You can find it by opening a new Azure tab, then clicking on the directory icon at the top of the window (see red square below).



Enter this tenant name on the shell and press ENTER. The shell will ask you to authenticate against Azure AD. Proceed by clicking on the *microsoft.com* link and a new tab will appear. Enter the code displayed in the shell into this tab.



After that, your workbench will be linked with your Azure AD tenant. You can now go back on your workbench, and then refresh it to complete your deployment.

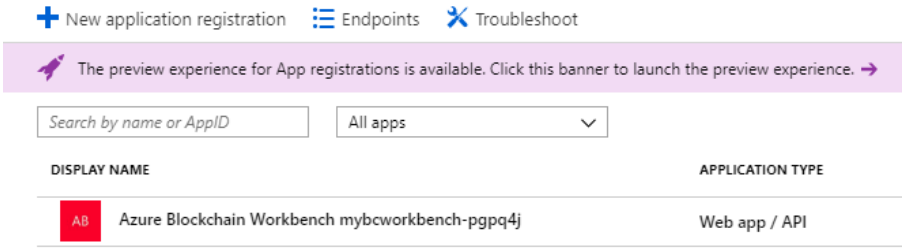


Auto-assigning the Administrator role to yourself

For some reasons, you may not directly have the administrator role. If, in the Workbench, you see that you can add new applications, you can skip this part.

If not, to fiw the situation and get the Administrator role, perform the following steps:

1. Go into the Azure AD that you linked with the Workbench and select **App registrations**.
2. Click on the Workbench applications. If, in this tab, you can't see the Workbench, click on the button **View all applications** and it should appear.



3. A tab with information about Workbench will appear, click on the link below **Managed application in local directory**.
4. You are now in the section **Identity and Access Management** of the application. Click on **Users** and groups, then click on **+ Add user**.
5. Add yourself as an Administrator, then click on **Save**.
6. Sign out and sign in again as yourself on the Azure Blockchain Workbench and you should now be an Administrator.

Note	The above steps also allow you to add new Workbench administrators.

Deploying the supply chain of refrigerated products application

Now that your Azure Blockchain Workbench is up and running, and online, it's time to deploy your first application on it!

To do so, perform the following steps:

1. Get the application files.

   To deploy an application on the workbench, you need two files:

   i. **A configuration file (.json)**: The configuration metadata defines the high-level workflows and interaction model of the blockchain application.

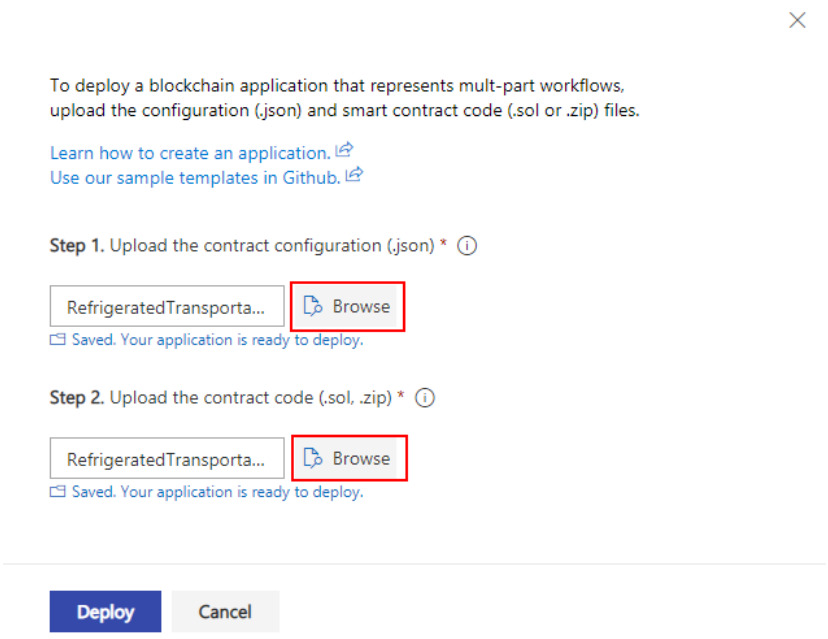
   ii. **A smart contract code file (.sol)**: Smart contracts represent the business logic of the blockchain application. Currently, the workbench supports Ethereum as a ledger and so it uses Solidity contracts to work.

   For your end-to-end solution, you will use the Refrigerated Transportation Sample Application for Azure Blockchain Workbench, available on Github. More specifically, you need to download the two files hereafter:

   - Configuration file[32].
   - Smart contract code file[33].

2. Deploy your application.

   Go back to your workbench, then click on **New** to create your first application. You will need the two above files here: the former will be used is the configuration file. Click on **Browse**, then select your file on your computer. Then, the UI will ask you for the latter, i.e. the smart contract code file. Perfom the same steps.



[32] RefrigeratedTransportation.json: https://github.com/Azure-Samples/blockchain/blob/master/blockchain-workbench/application-and-smart-contract-samples/refrigerated-transportation/ethereum/RefrigeratedTransportation.json

[33] RefrigeratedTransportation.sol: https://github.com/Azure-Samples/blockchain/blob/master/blockchain-workbench/application-and-smart-contract-samples/refrigerated-transportation/ethereum/RefrigeratedTransportation.sol

Note Each time you select a file for the application, the workbench will verify or compile it. If any errors are found during the process, they will be displayed, and you will have to fix them before deploying your application.

If everything goes well, your application will be now deployed, and you can access it by clicking on its icon. But at the moment, you are still the only one who has access to the workbench and the application.

The main objective of this guide is to set up an application shared by multiple organizations and users. Thus, in the next part, you will see how to create new users and invite external users from other organization inside your Azure AD tenant, and by extension your application.

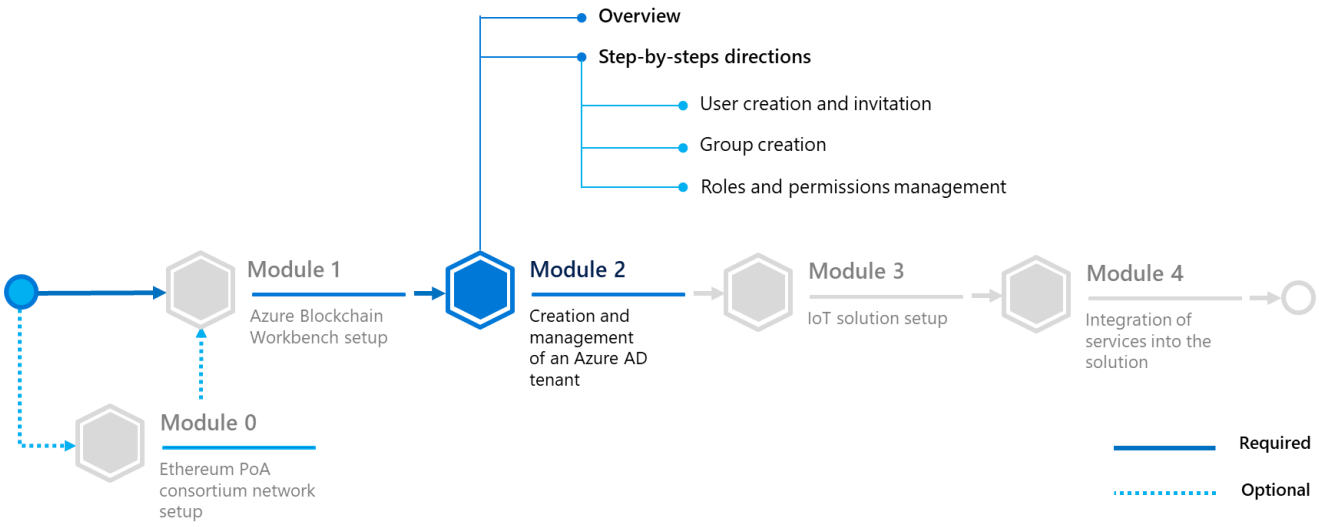
Module 2: Creation and management of an Azure AD tenant

Overview

An Azure Blockchain Workbench is always linked to an Azure AD tenant. This feature allows user management within Azure, so an Administrator doesn't have to bother about key management or user identities on the Workbench.

Everything is indeed done directly on Azure AD and an Administrator can just click on **+Add** on a Workbench Application to add new (external) users (if they already exist inside another Azure AD tenant).

Your objective in this module consist in configuring an Azure AD environment, which represents a pseudo-consortium between two fictitiuous companies, namely Contoso and Woodgrove.



In this module, you will learn to:

- Create a user within the Azure AD portal (you will get a user account and his temporary password).
- Invite an external user into an Azure AD tenant.
- Create groups and assigng users into them.
- Manage Workbench roles and permissions (Workbench user, application user, Workbench administrator).

Step-by-step directions

This module covers the following activities:

1. Adding new users in your Azure AD tenant.
2. Adding new groups inside your Azure AD.
3. Managing roles and permissions.

Each activity is described in order in the next sections.

Adding new users in your Azure AD tenant

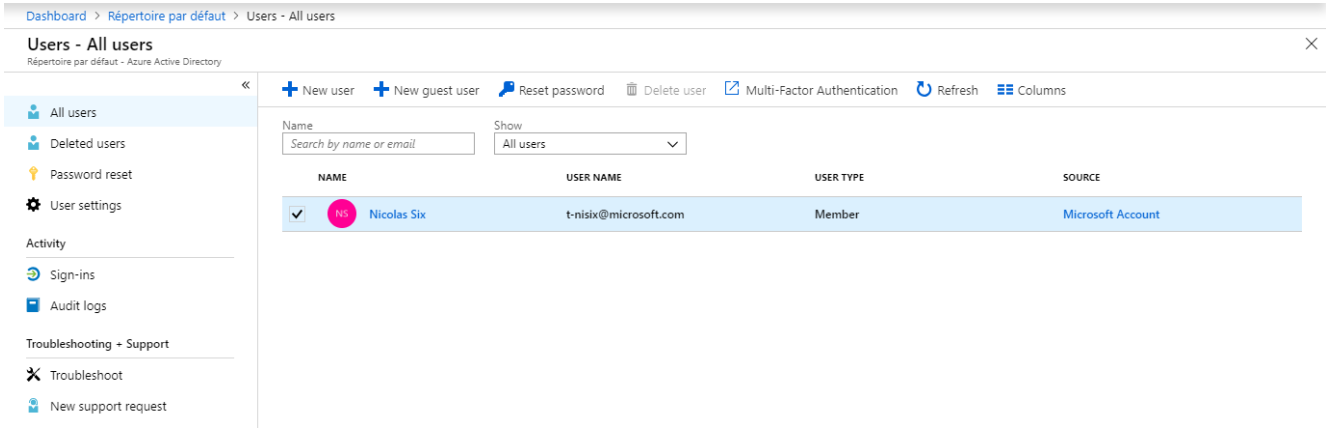
Creating a user in your Azure AD tenant

First of all, you're going to create device and user accounts directly in your Azure AD administration panel. (This will be convenient for creating your IoT devices identities for this guide.)

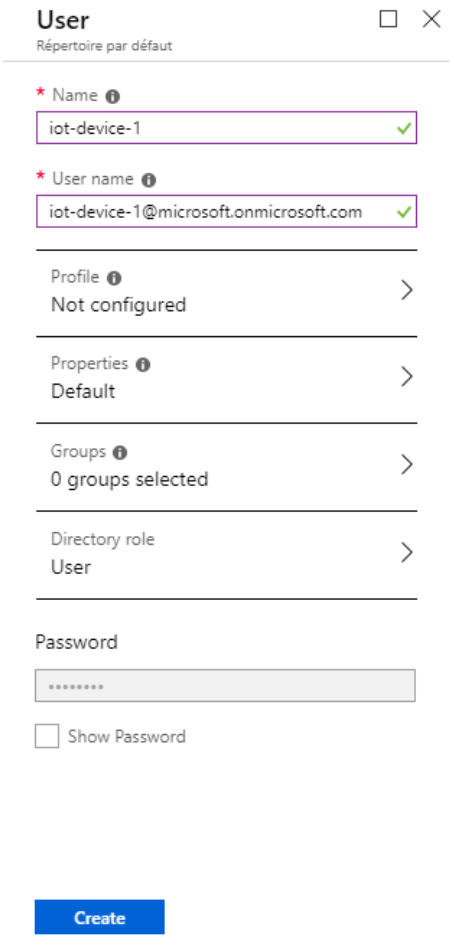
Perform the following steps:

> **Note** For a real-life production scenario, it is possible to create identities in bulk for IoT devices, but, since you're going to create only four IoT devices in your Azure AD tenant for this guide, this is not necessary in this case and will be not illustrated.

1. In your Azure administration panel, search for "*Azure Active Directory*". Once inside it, you will be in your default directory.



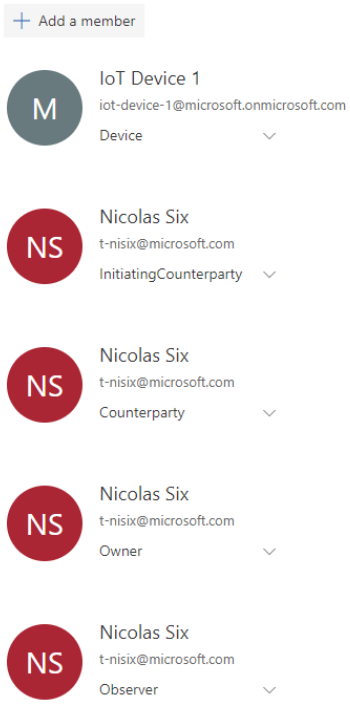
2. Click on **+New User**. Fill in the form below to create an account for an IoT device: iot-device-1.



Setting	Description
Name	Specify the name of your new "user", for example "iot-device-1"
User name	Specify the email address of the new "user". It must finish by @<your tenant name>.onmicrosoft.com.
Profile	Click if you want to fill more information about the "user" (for example, their name, job, department).
Properties	Let it by default.
Groups	This is where you can assign groups to a user. Here, let it by default, because you're going to create and assign groups later.
Directory role	Let it by default, as you won't create new administrators. But keep in mind that you can create new Administrators directly by choosing this role here.
Password	This is not a field to fill, click on **Show Password** and the temporary password will appear in it. Make a note of this password for your IoT devices. You will use this password right away to sign in into the Workbench as if you were that "user". For a real person, you will have to share it with the person so that he/she can connect.

3. Click on **Create**. The iot-device-1 "user" account is now created.

4. Now, the above accounts can be added inside your Workbench application. Sign in into your Workbench, click on the **Refrigerated Transportation** application, then click on **Members** on the top-right of your screen. To simplify testing, you will add four times yourself as all the different roles except for Device and add your IoT device' account as **Device**.



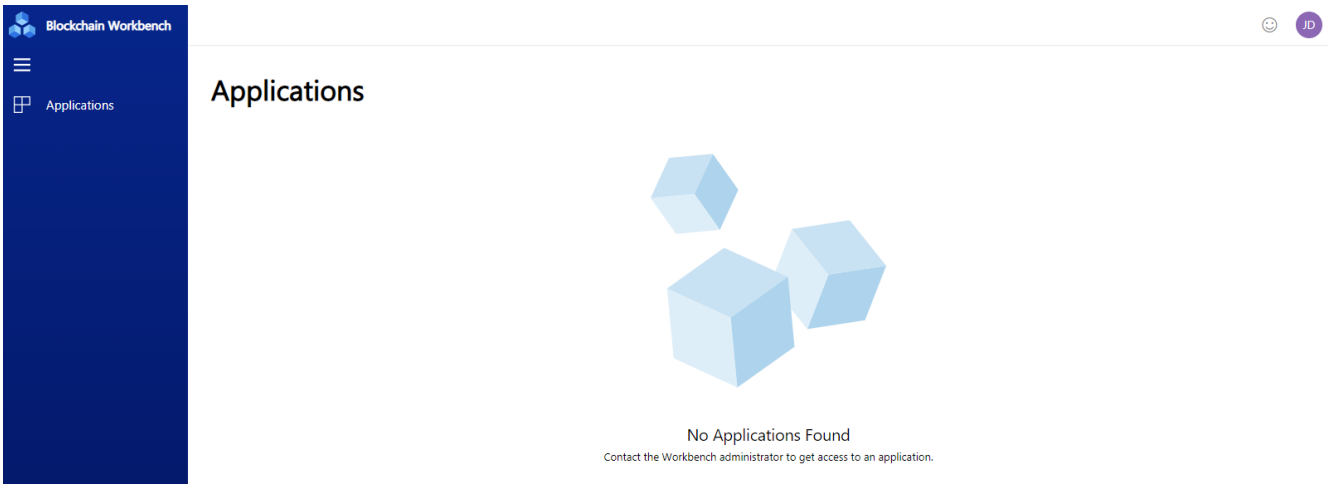
5. Repeat steps 2 and 3 to create the iot-device-2, iot-device-3, and iot-device-4 device accounts. You are going to use them later when you will implement your IoT architecture in the next module.
6. Repeat steps 2 and 3 to create a user account, for example "John Doe" as illustrated after.

To try the above user account, you're going to sign in with it on your Azure Blockchain Workbench instance.

Open a Workbench tab (preferably in a private session of your browser of choice). it will ask you to sign in. Fill the email with the yser name that you gave while creating the user, then the temporary password as the password.

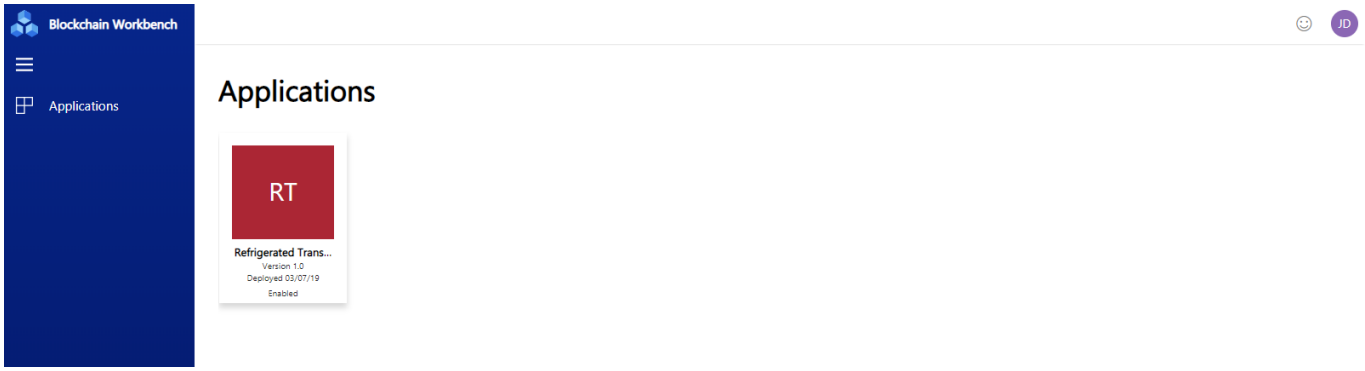
On the next tab, you will have to change that password, do it and click on **Connect**.

You are now logged into the workbench, but you can't see any application. This is because a user needs to be primarly added on an application by an administrator.



Go back to your browser with your Administrator account logged in: click on your **Refrigerated Transportation** application, then click on **Members** and add "John Doe" (or your custom name user).

If you refresh your private browser with "John Doe" logged in, you will now see the **Refrigerated Transportation** application.

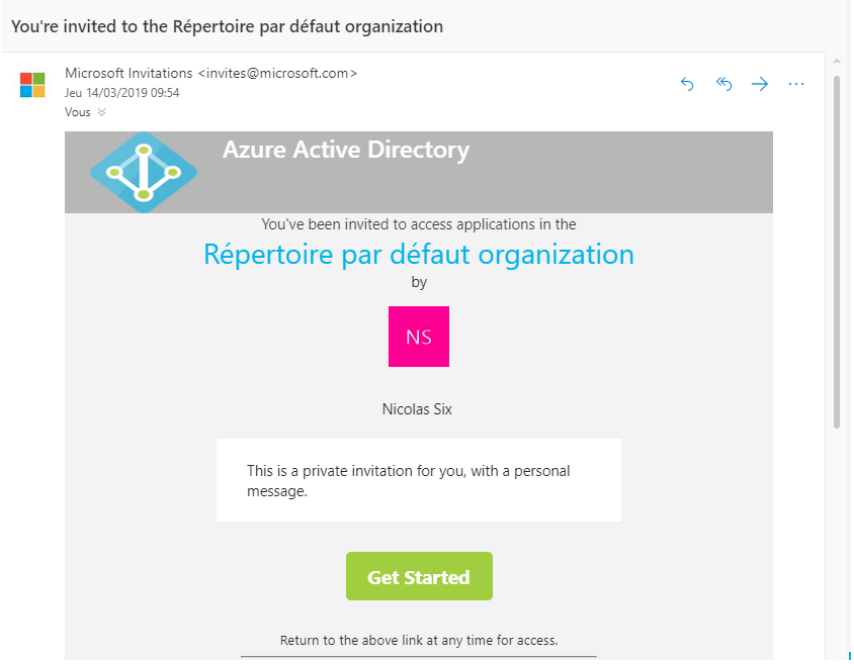


Inviting a guest user in your Azure AD tenant

Instead of creating manually new users, you can directly invite external people to join your Azure tenant. Therefore, guests will be able to login with their own (professional) email, and if this email corresponds to an existing Azure AD tenant/Office 365 subscription, they will get an access to the Workbench on their Azure AD home page like any other application.

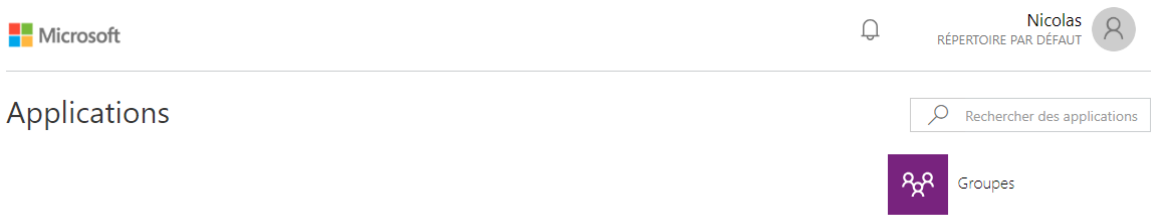
Perform the following steps:

1. Click on **+New guest** user. Fill the form with the guest email (use your own email for testing here) and type a personal welcome message if you want to add one. Then, click on **Invite**.
2. Verify that you received the email in your inbox.

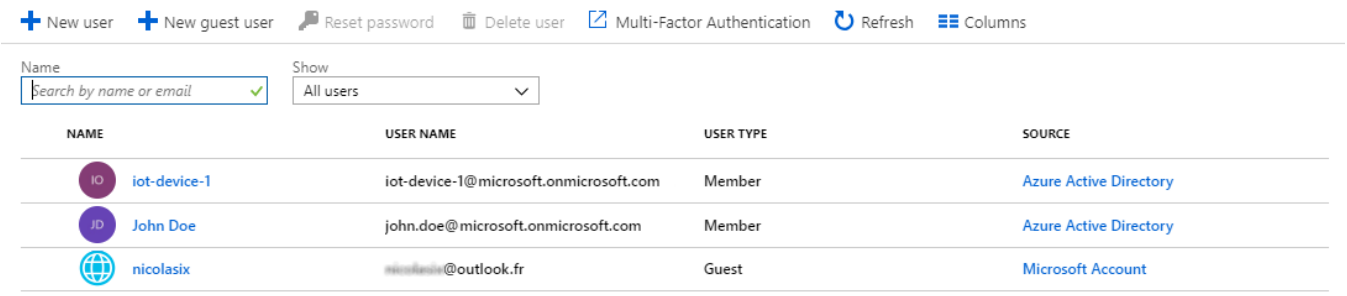


Click **Get Started**, and follow the instructions. (Depending on the email address that was specified for the invite, you may have to create a Microsoft account.)

Ath the end, you will be redirected to the Azure AD Access Panel home page at https://myapps.microsoft.com.



Note You can see the difference between a regular used (created inside your Azure AD tenant) and a guest user (invited from outside) when looking inside the user access panel on Azure AD).



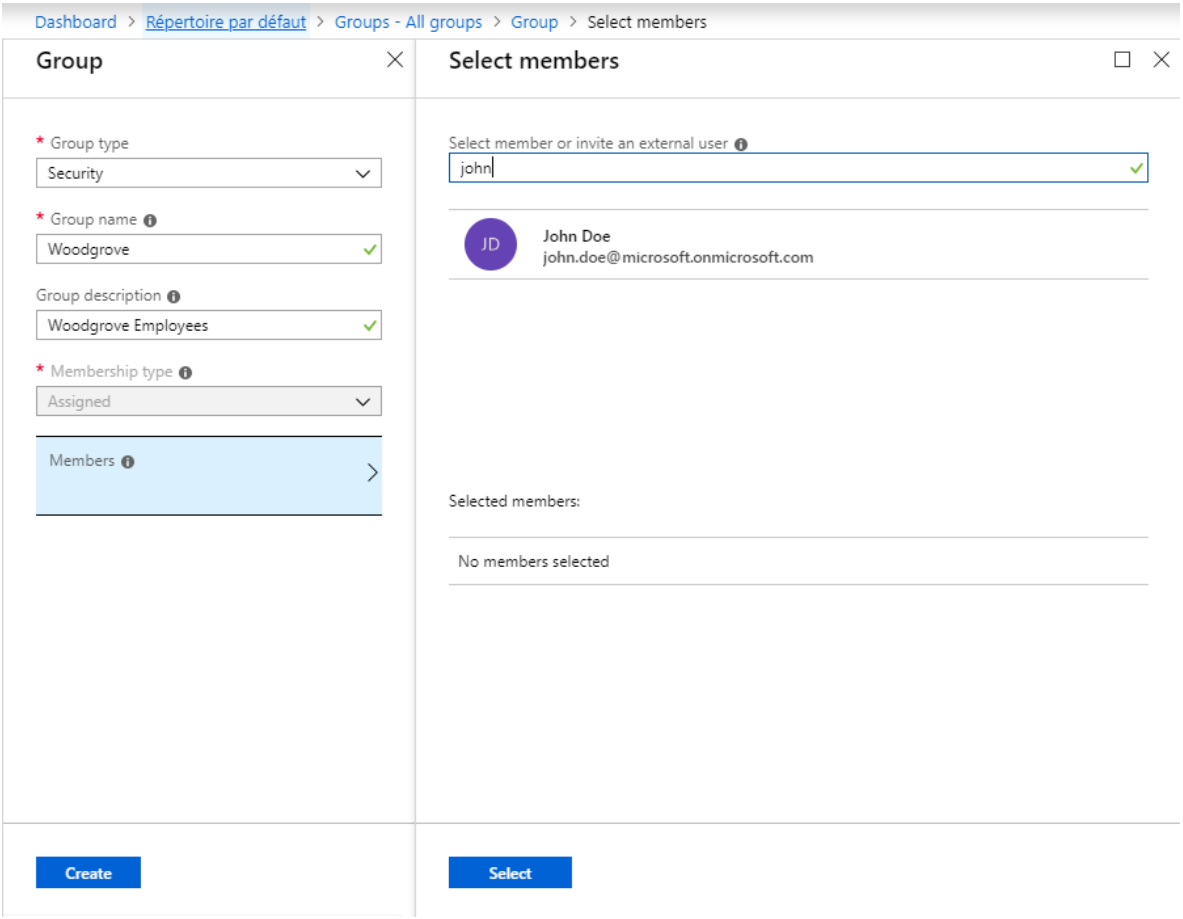
Note For more information, see [Azure Active Directory B2B Documentation][34].

Adding new groups inside your Azure AD

Now that you're able to create and invite new users, a good idea would be to create groups which represents their company. It would be easier to manage when adding or deleting user, and giving roles to them.

Perform the following steps:

1. Go inside your Azure AD tenant, then click on **Groups**.
2. Once inside the panel, click on **+New group**.
3. Fill the form with accurate information.



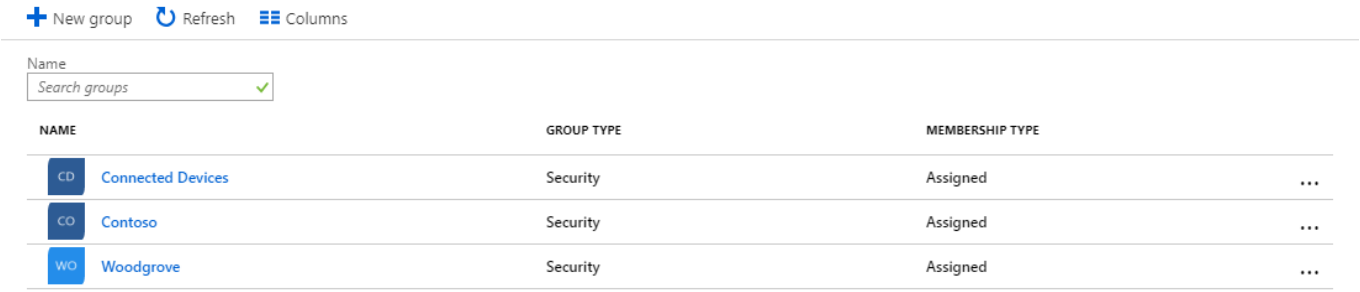
[34] Azure Active Directory B2B Documentation: https://docs.microsoft.com/en-us/azure/active-directory/b2b/

Setting	Description
Group type	Specify the type of the group: • **Security.** Used to manage member and computer access to shared resources for a group of users. For example, you can create a security group for a specific security policy. By doing so, you can give a set of permissions to all the members at once, instead of having to add permissions to each member individually. For more info about managing access to resources, see article MANAGE ACCESS TO RESOURCES WITH AZURE ACTIVE DIRECTORY GROUPS[35]. • **Office 365**. Provides collaboration opportunities by giving members access to a shared mailbox, calendar, files, SharePoint site, and more. This option also lets you give people outside of your organization access to the group. For more info about Office 365 Groups, see article LEARN ABOUT OFFICE 365 GROUPS[36].
Group name	Specify the name of the new group.
Group description	Specify the description of the new group.
Membership type	Let it by default (Assigned).
Members	Set the Members of the group. Clicking on this field allows you to directly add existing (guest) users inside your group during its creation. On the right panel, search for users and click on their name to add them to the group, then click on **Select** when finished.

For the sake of this guide for developers, you will creat three groups:

- *One for future connected devices named "Connect Devices",*
- *And two for invited guest users from their respective (fictitious) company, named "Contoso" and "Woodgrove".*

Once done, your newly created group should appear inside your group list as illustrated hereafter.



[35] MANAGE ACCESS TO RESOURCES WITH AZURE ACTIVE DIRECTORY GROUPS: https://docs.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-manage-groups

[36] LEARN ABOUT OFFICE 365 GROUPS: https://support.office.com/article/learn-about-office-365-groups-b565caa1-5c40-40ef-9915-60fdb2d97fa2

Managing roles and permissions

Managing roles and permissions for workbench is quite easy, as you can only be a regular user or an administrator. A user can only access to applications if he/she was invited by an Administrator to join them. Therefore, an aAdministrator can add users to applications and give them defined roles inside them. They can also deploy new applications inside the Workbench.

Adding users inside the Workbench is easy to do. (external) users only need to exist inside the Azure AD tenant to have an access granted to the Workbench.

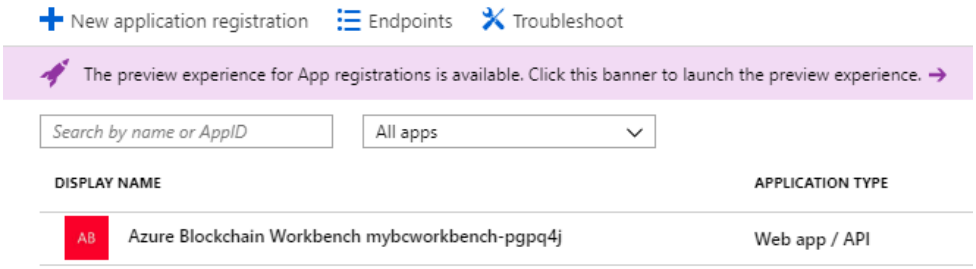
But managing administrators is a bit more difficult. In this part, you're going to see how to add new administrators, or groups of users to automatically grant the zdministrator role.

Managing Workbench roles and permissions

To manage Workbench roles and permissions, you need to go inside your Workbench management app.

Perform the following steps:

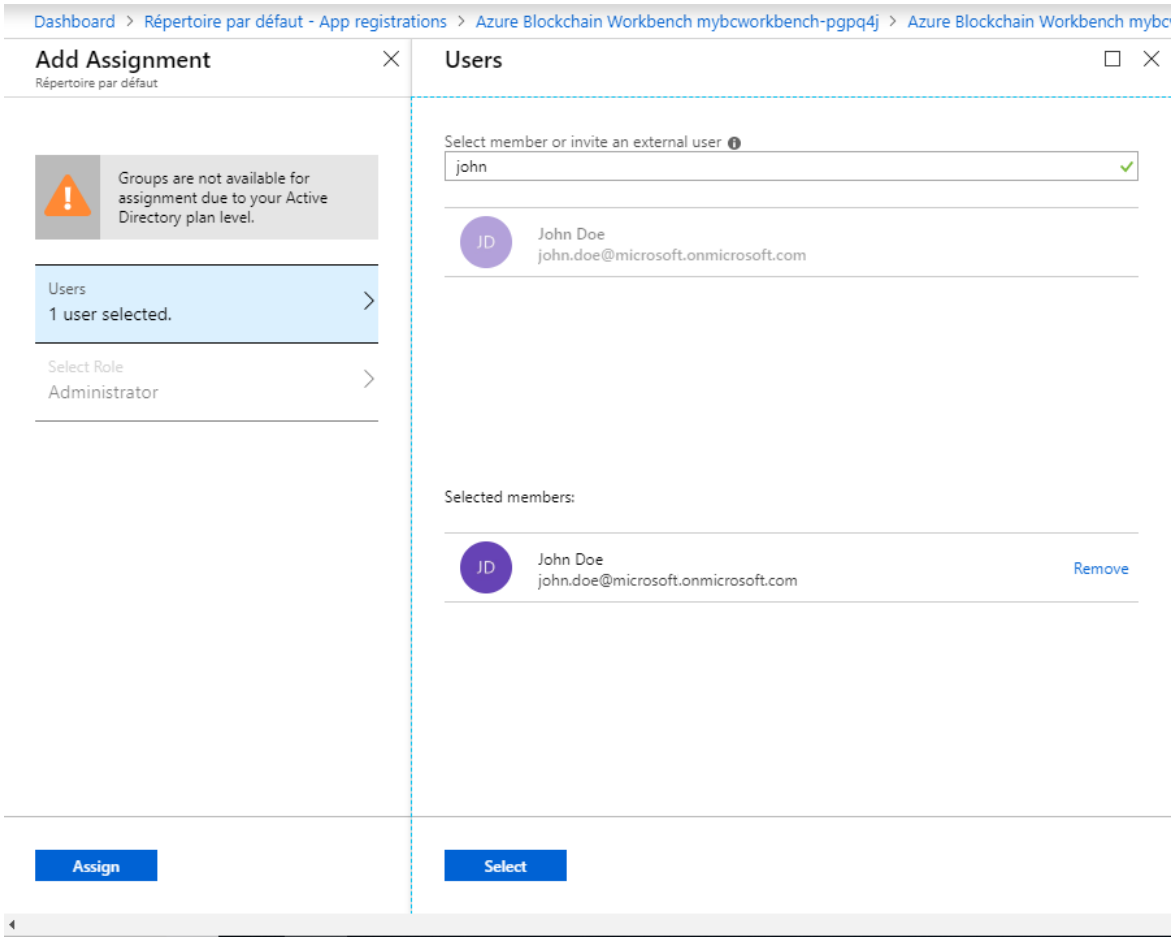
1. Go into the Azure AD tenant that you linked with the Workbench and select **App registrations**.
2. Click on the Workbench applications. If, in this tab, you can't see the Workbench, click on the button **View all applications** and it should appear.



3. A tab with information about Workbench will appear. Click on the link below **Managed application in local directory**.
4. You are now in the **Identity and Access Management** section of the application. Click on **Users and groups**.

   In this panel, you can see the list of every user of the Workbench. You can click on **+Add User** to allow Administrator role to users (or groups).

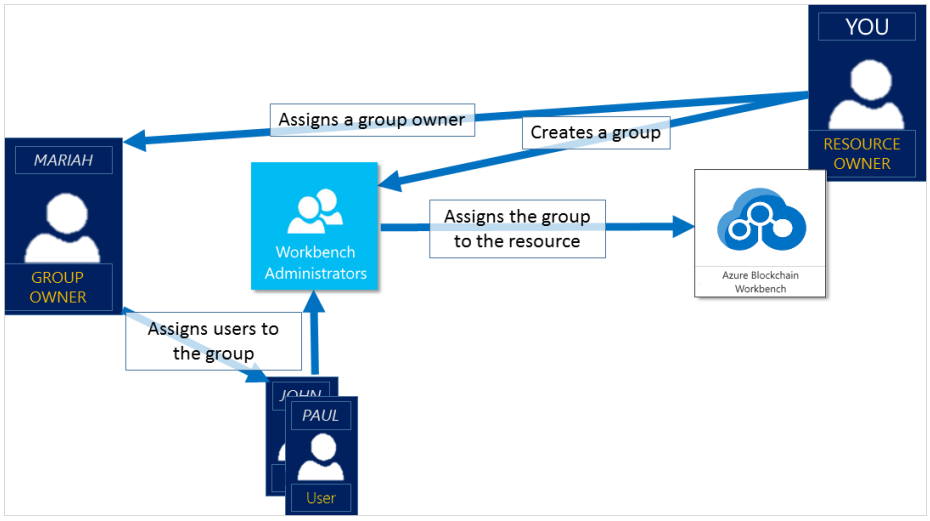
   Search and click the users and the groups that you want to add to the Workbench, then click on **Select** and **Assign**.



For a small consortium, adding users as administrators is the way to go. But you maybe want to grant the permission to invite new users to an Administrator group, or even grant the permission to add new (external) users into your Azure AD tenant in order to delegate Workbench management task.

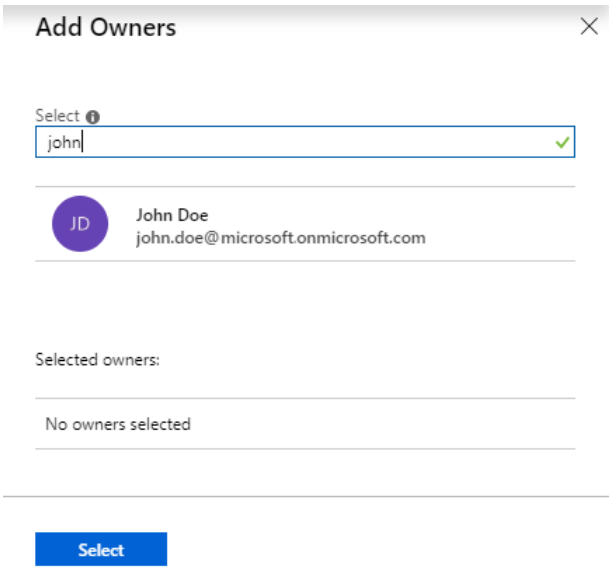
Managing Azure AD roles and permissions

In this part, you're going to look how to assign someone as a group owner in your Azure AD tenant, in order to let him/her in turn invite users into groups, in order to automatically assign them as administrators or just to regroup users into groups which represents their own company.



Perform the following steps:

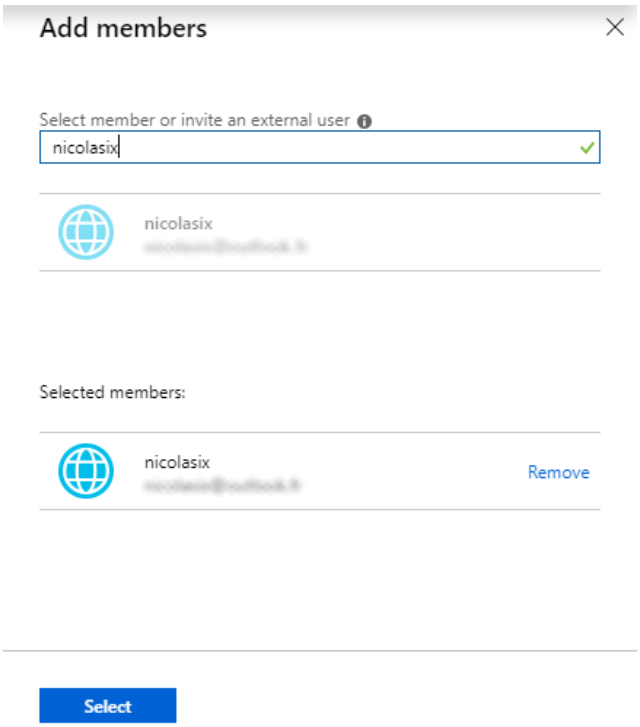
1. Go inside your **Azure Active Directory**, then click on **Groups**.
2. Select a group which is inside the list, for example **Woodgrove**.
3. You're going to add a new Group Owner to this group. Inside your group panel, click on **Owners**, then once inside it click on **+Add owners**.
4. Search for the user you want to add as an owner, then click on it and click on **Select**.



John Doe is now the owner of the group, and he is able to add users inside it.

You can also add new members into the group.

1. Inside your group panel, click on **Members**, then once inside it click on **+Add members**.
2. Search for the user you want to add as a member, then click on it and click on **Select**.



Now, you are able to invite new users into your Workbench and grand them Administrator role. If you want to go further with Azure AD, feel free to read the whole documentation here[37].

Otherwise, you're now ready to build your IoT solution.

[37] CREATE A BASIC GROUP AND ADD MEMBERS USING AZURE ACTIVE DIRECTORY: https://docs.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-groups-create-azure-portal

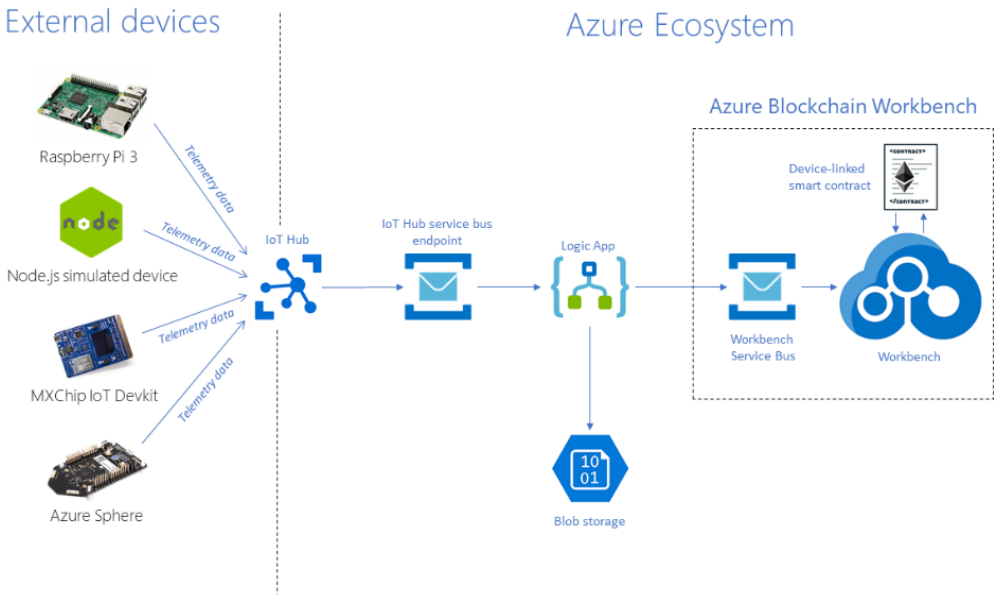
Module 3: IoT solution setup

Overview

This module should be considered as one of the most important in this guide.

The goal is to enable communication between an IoT part, which will be constituted by some connected device linked to an IoT Hub, and the Workbench. This is not the easiest part: you will have to take care of device identity management through your Workbench, as well as message routing.

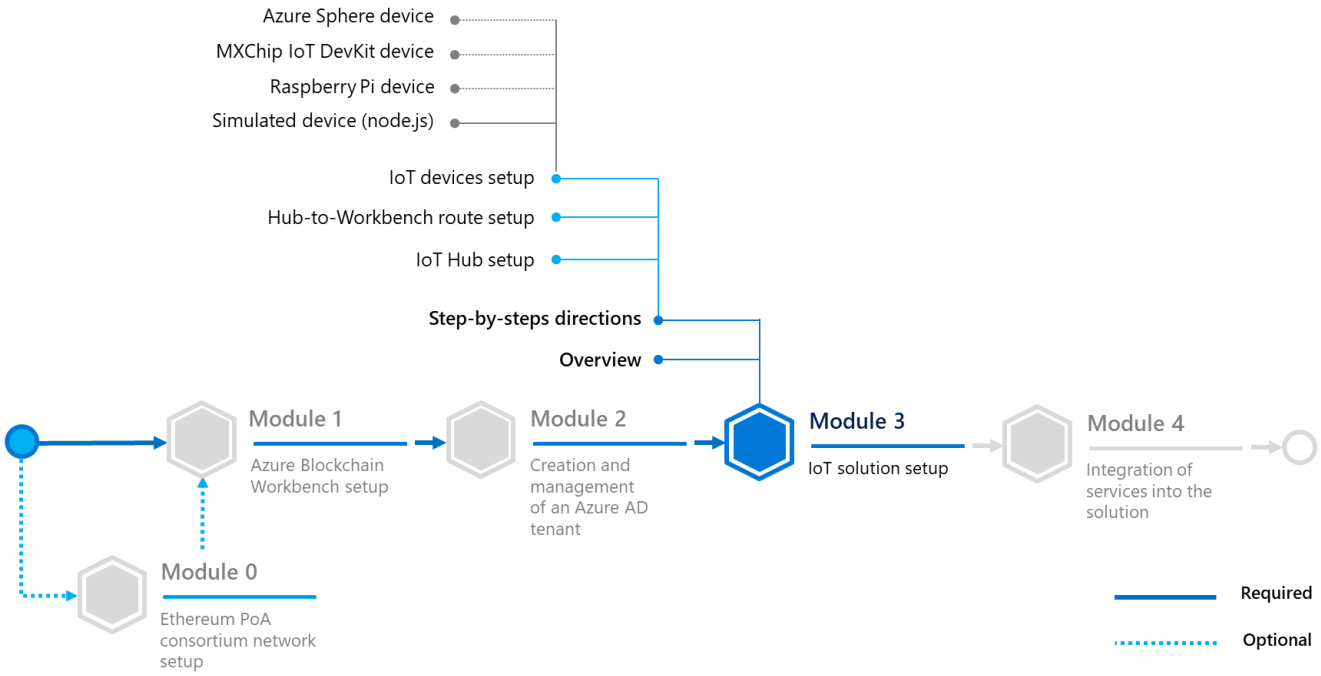
To share an overview of the solution, let's have a look at this diagram.



You will start from your devices and connect them to the IoT Hub.

After that, you will need to use a Logic App to bind the device identity contained in your message with an Azure AD identity, required to identify users in the Workbench.

Thanks to that service, you will be able to route the message in Azure Blockchain Workbench but also in a storage which will allow you to build data charts with Power BI later on.



On this module, you will learn to:

- Deploy an IoT architecture, constituted by an IoT Hub, a Service Bus/Queue, a Logic App and connected devices. The resulting IoT solution wil be able to route messages from your end devices to the Workbench you previously setup.

Step-by-step directions

This module covers the following activities:

1. Setting up of the IoT Hub.
2. Routing messages from the Hub to the Workbench.
3. Adding IoT devices to your solution.

Each activity is described in order in the next sections.

Let's begin with your first core component: an IoT Hub.

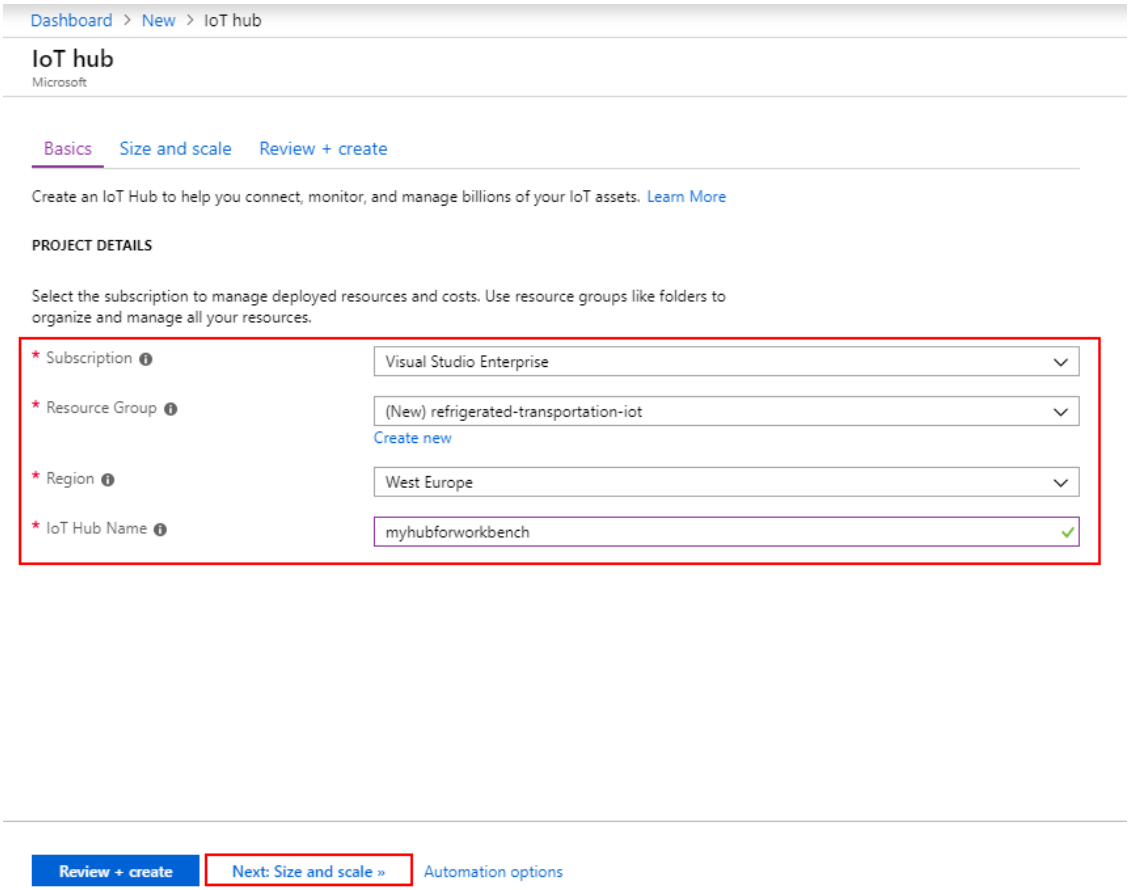
Setting up of the IoT Hub

This section describes how to create an IoT hub using the Azure portal. The IoT hub will allow your connected devices to communicate with Azure Blockchain Workbench.

Perform the following steps:

Note The whole documentation about Azure IoT Hub is available [at this link](38) if needed. This guide is inspired by this documentation and summarizes the key steps to follow.

1. Sign in to the Azure portal.
2. Choose +**Create a resource**, then choose **Internet of Things**.
3. Click **IoT Hub** from the list on the right. You see the first screen for creating an IoT Hub.

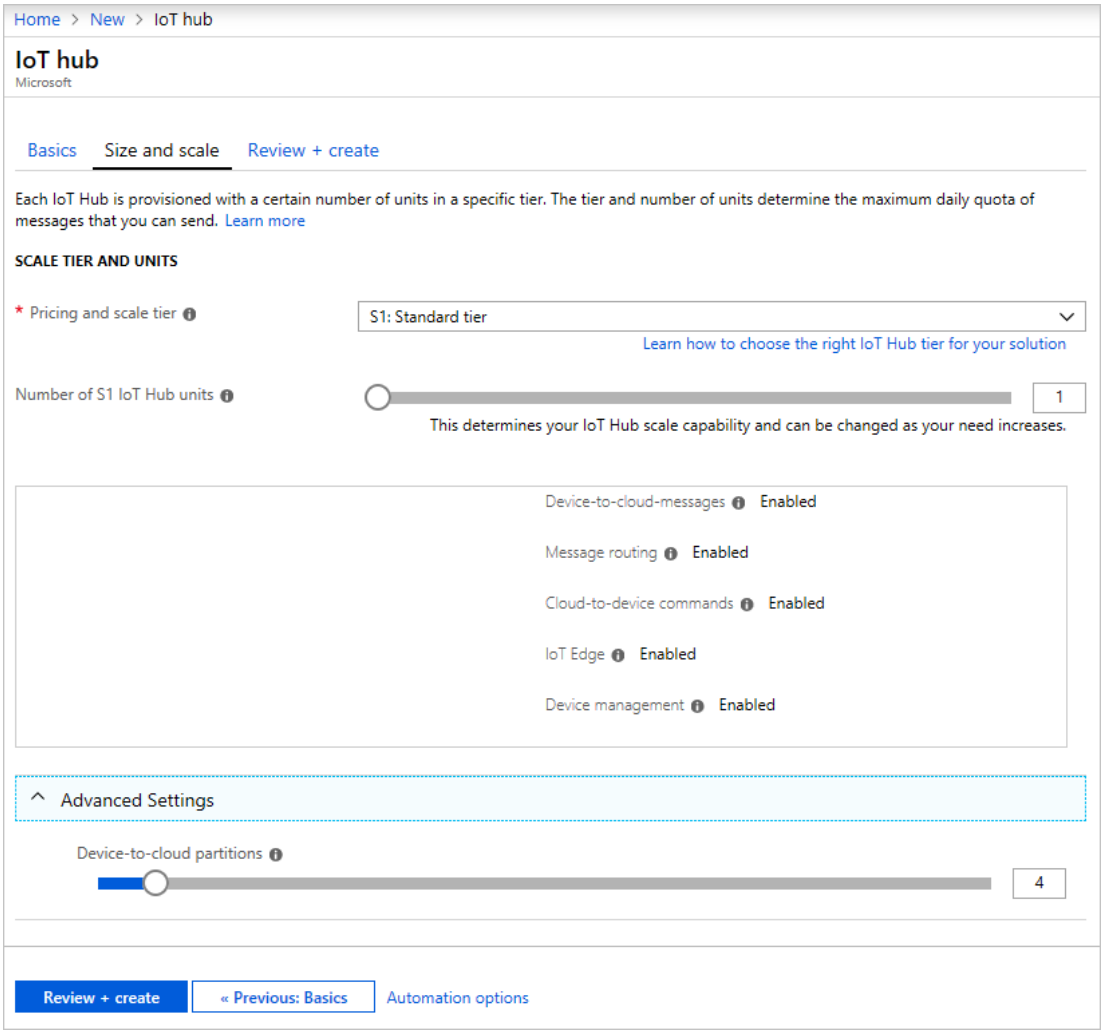


Setting	Description
Subscription	Select the subscription to use for your IoT hub.
Resource Group	You can create a new resource group or use an existing one. To create a new one, click **Create** new and fill in the name you want to use. To use instead an existing resource group, click **Use existing** and select the resource group from the dropdown list.

[38] IoT Hub Documentation: https://docs.microsoft.com/en-us/azure/iot-hub/

Region	This is the region in which you want your hub to be located. Select the location closest to you from the dropdown list.
IoT Hub Name	Put in the name for your IoT Hub. This name must be globally unique. If the name you enter is available, a green check mark appears.

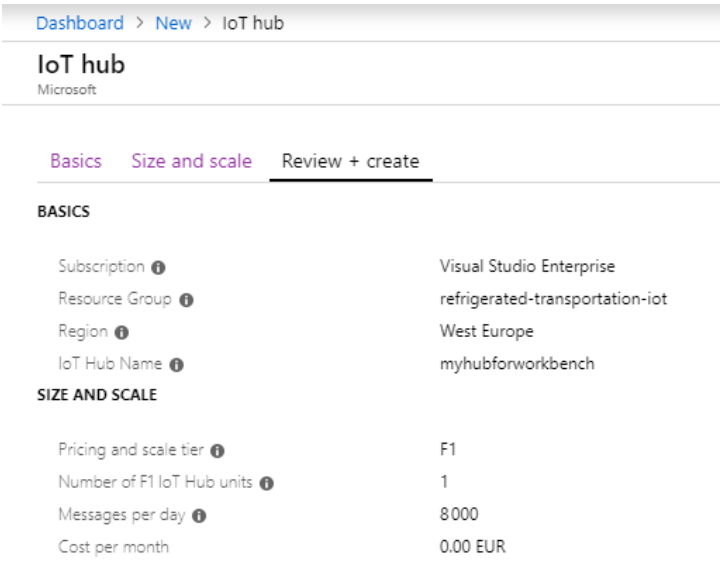
4. Click **Next: Size and scale** to continue creating your IoT Hub.



Setting	**Description**
Pricing and scale tier	Specify the tier to use. You can choose from several tiers depending on how many features you want and how many messages you send through your solution per day. For this guide, the free tier should be enough. Keep in mind that an Azure account can only have one free tier hub.
Number of S1 IoT Hub units	Specify the number of units. The number of messages allowed per unit per day depends on your hub's pricing tier. You don't need more than one unit for this guide.

Device-to-cloud partition	Specify the number of partitions. This property relates the device-to-cloud messages to the number of simultaneous readers of the messages. Most IoT Hubs only need four partitions, you can keep it by default.

5. Click **Review + create** to review your choices. You should see something similar to this screen hereafter.



6. Click on **Create** to create your new IoT hub. Creating the hub takes a few minutes.

Routing messages from the Hub to the Workbench

Setting up a service bus for message routing

Before connecting new devices to your newly created IoT hub, you need to set up message routing from the IoT hub to suitable services. All the messages will be stored into a blob storage for analysis with Power BI and ingested by the Workbench. If the temperature or the humidity reaches an out-of-bounds value, the smart contract will set its state to "out-of-compliance".

Note For this part, this guide is inspired by a tutorial available on Github at this link[39].

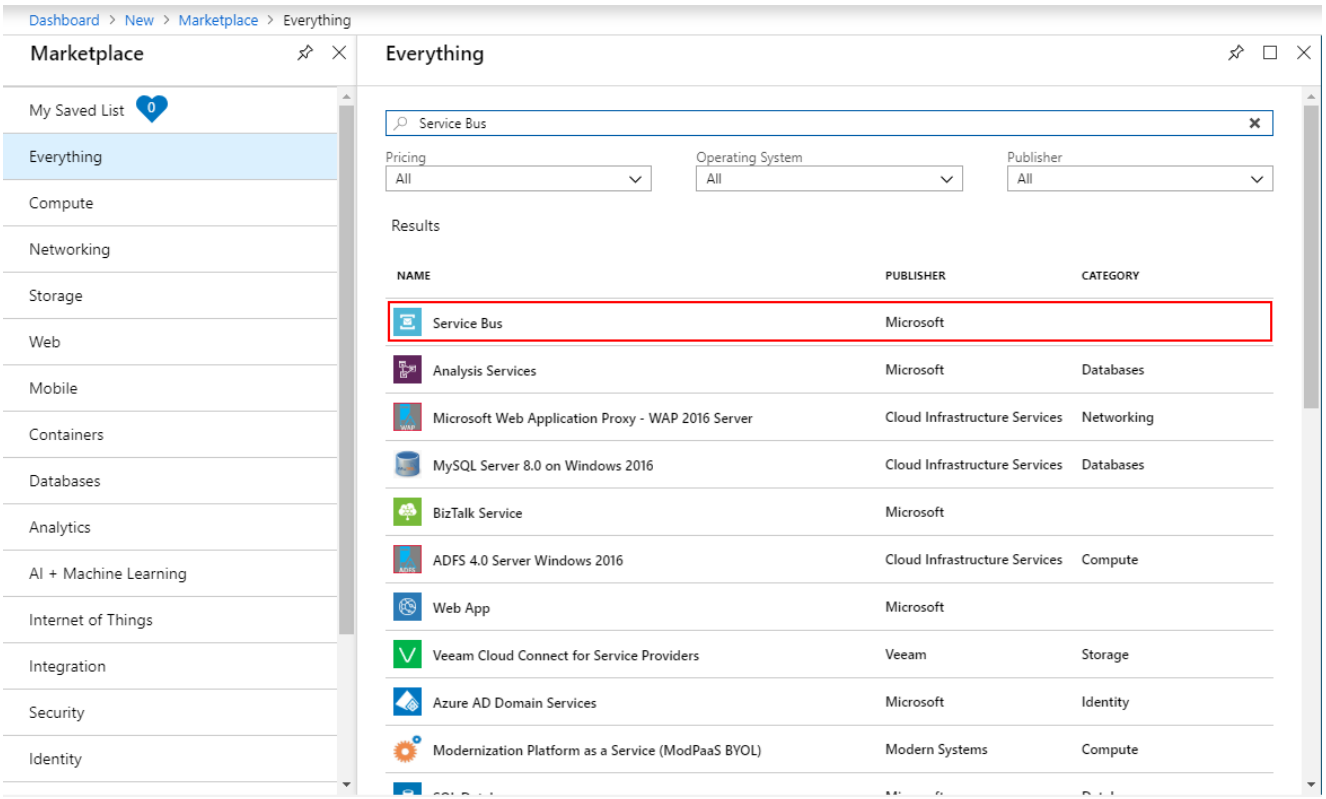
First of all, you are going to create the route between the IoT Hub and Azure Blockchain Workbench. To do so, you will create a Service Bus.

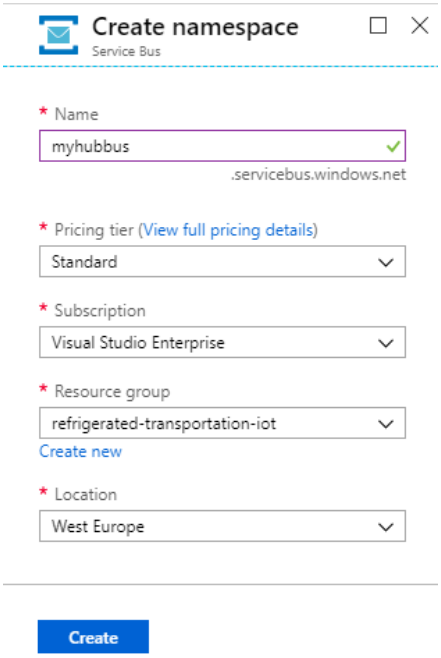
Service Bus queues support a brokered messaging communication model. When using queues, components of a distributed application do not communicate directly with each other; instead, they exchange messages via a queue, which acts as an intermediary (broker).

[39] DELIVERING DATA FROM IOT HUB TO AZURE BLOCKCHAIN WORKBENCH: https://github.com/Azure-Samples/blockchain/blob/master/blockchain-workbench/iot-integration-samples/ConfigureIoTDemo.md

To create the Service Bus, perform the following steps:

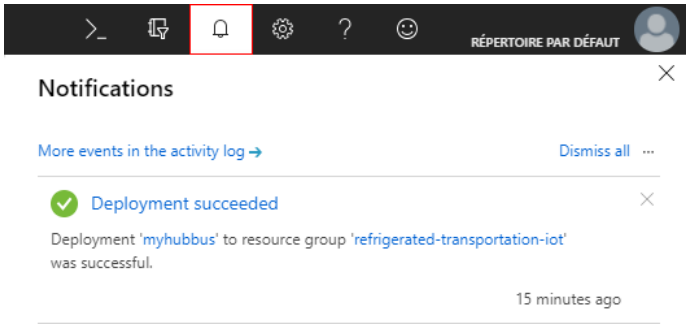
1. Sign in to the Azure portal.
2. In the left navigation pane of the portal, select **+ Create a resource**, select **Integration**, and then select **Service Bus queue**.





Setting	Description
Name	Specify the name of your namespace for the bus. It should be globally unique, so the system will immediately check to see if the name is available.
Pricing Tier	Set the pricing tier of the resource (Basic, Standard or Premium).
Subscription	Select the subscription to use for your Service Bus. You will choose **Standard** for this guide.
Resource group	You can create a new resource group or use an existing one. To create a new one, click **Create** new and fill in the name you want to use. To use an existing resource group, click **Use existing** and select the resource group from the dropdown list. Here, you will use the same group as other IoT resources already created.
Location	This is the region in which you want your Service Bus to be located. Select the location closest to you from the dropdown list.

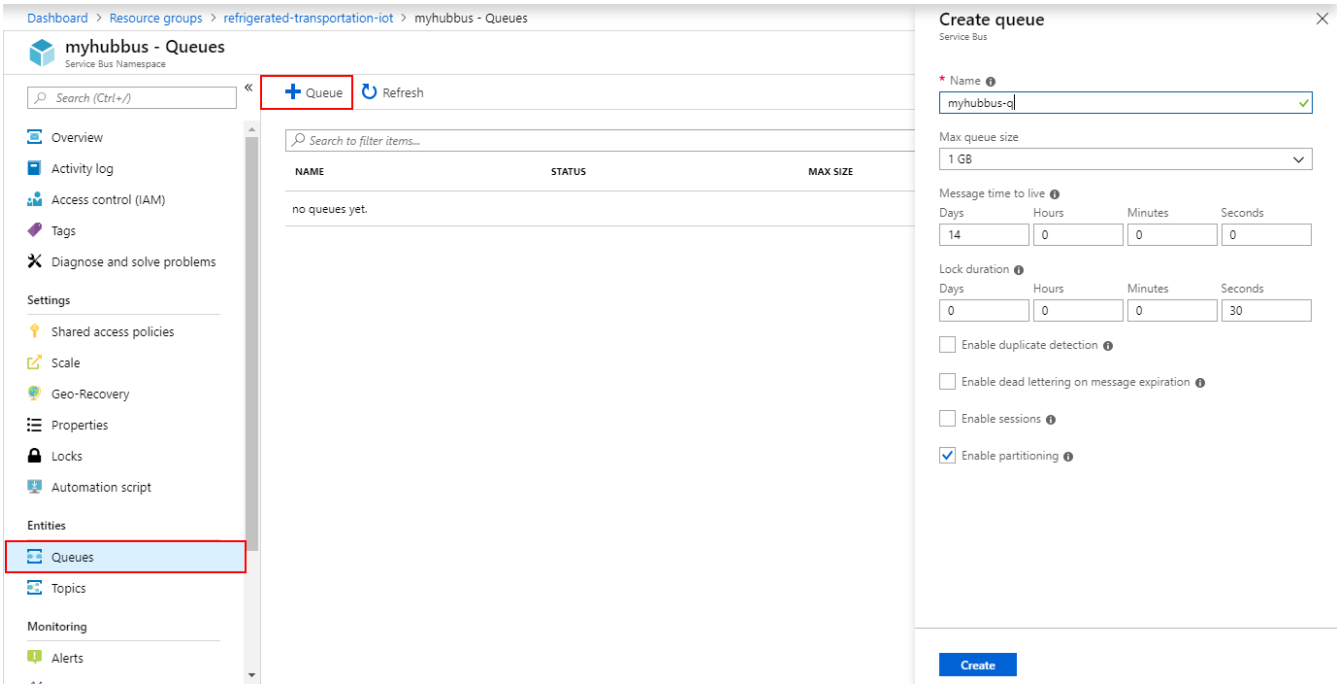
3. You can check that the deployment went well by going on your notification tab. Click on the **bell icon** on the toolbar.



Now, you need to create a queue inside the Service Bus. Go on your deployed Bus by clicking on it inside your notification tab or by searching it inside the **Resource Groups** list.

Perform the following steps:

1. On the Service Bus Namespace page, select **Queues** in the left navigational menu.
2. On the Queues page, select **+ Queue** on the toolbar.
3. Enter a name for the queue and leave the other values with their defaults.
4. Now, select **Create**.

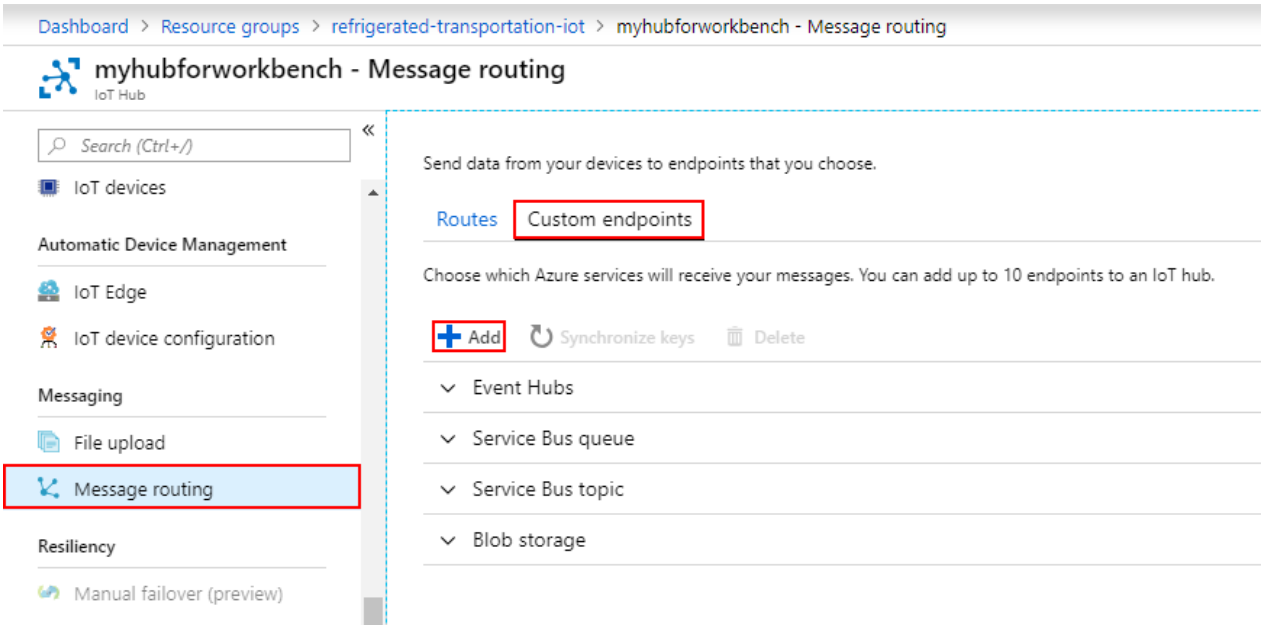


At this stage, your queue is now online and you're ready to configure routing in your IOT Hub and your Service Bus.

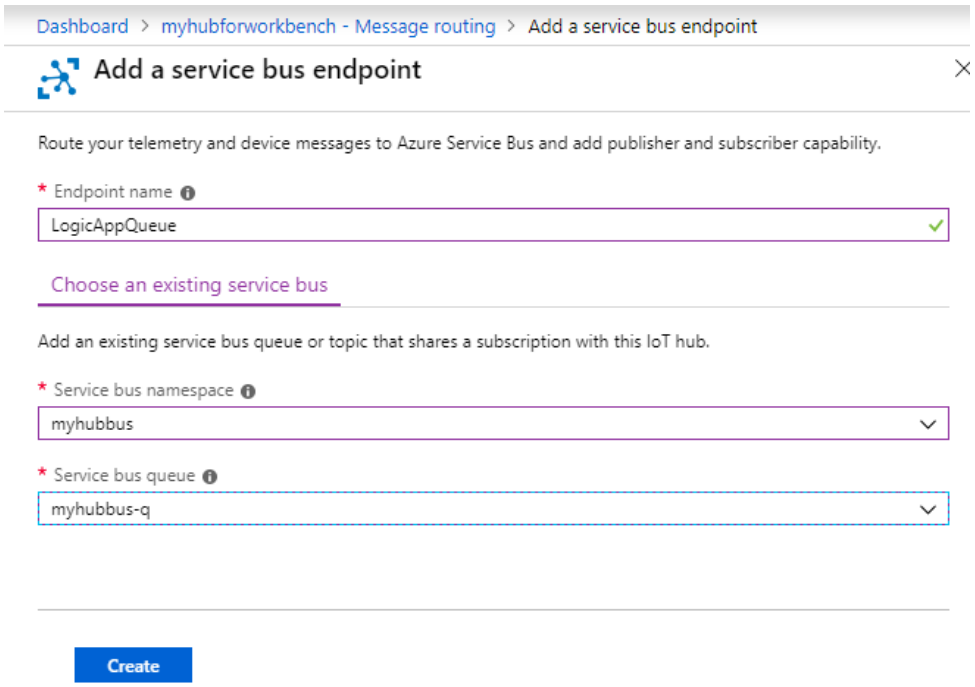
Routing messages to the Service Bus queue

Now that the Service Bus is operational, you can define the routes for incoming messages. Perform the following steps:

1. Go in **Resource Groups**, then click on your IoT hub resource group and click on your IoT hub.
2. Once inside, search for the **Message routing** menu. Click on **Custom endpoint** and add a new endpoint by clicking on **Add**.



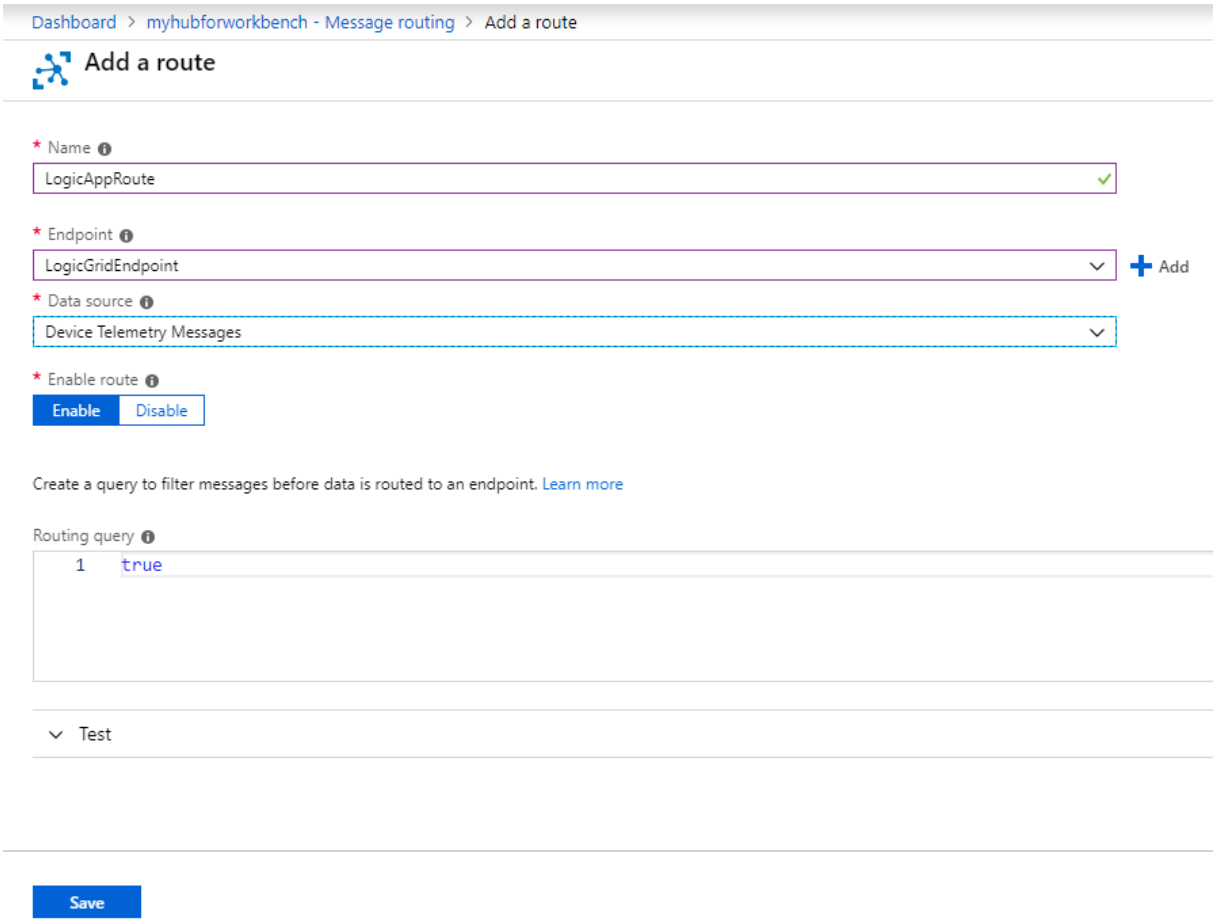
3. Choose **Service Bus queue** on the dropdown list, and a new tab will open up. Complete it and click on **Create**.



Setting	Description
Endpoint name	Specify the name for your future endpoint.
Service Bus namespace	In the dropdown list, choose the name of the Service Bus namespace you've created in the previous part.

Service Bus queue	In the dropdown list, choose the name of the Service Bus queue just created in the previous part.

4. Now, you need to configure the route from the IoT Hub to this endpoint. Click on **Routes** in your **Message Routing** menu, then **+ Add**.
5. Create a route by filling the route information.



Setting	Description
Name	Specify a name for your future route, for example "LogicAppRoute" as illustrated here.
Endpoint	In the dropdown list, select the name of the endpoint you've created in your IoT hub.
Data source	The data source field represents the type of the message which the route applies to. Select **Device Telemetry Messages**.
Enable route	Choose if you want to enable the route or let it as disabled for the moment. Here, you want it **enabled** by default, so click on it.
Routing query	This field can contain a filter for your messages. As you need that all the messages are redirected to your Logic App, let the default **true** variable in.

Congrats! Device messages are now correctly routed to your Service Bus queue. But now, you need additionally to push the message into Azure Blockchain Workbench.

Pushing routed messages to the Workbench

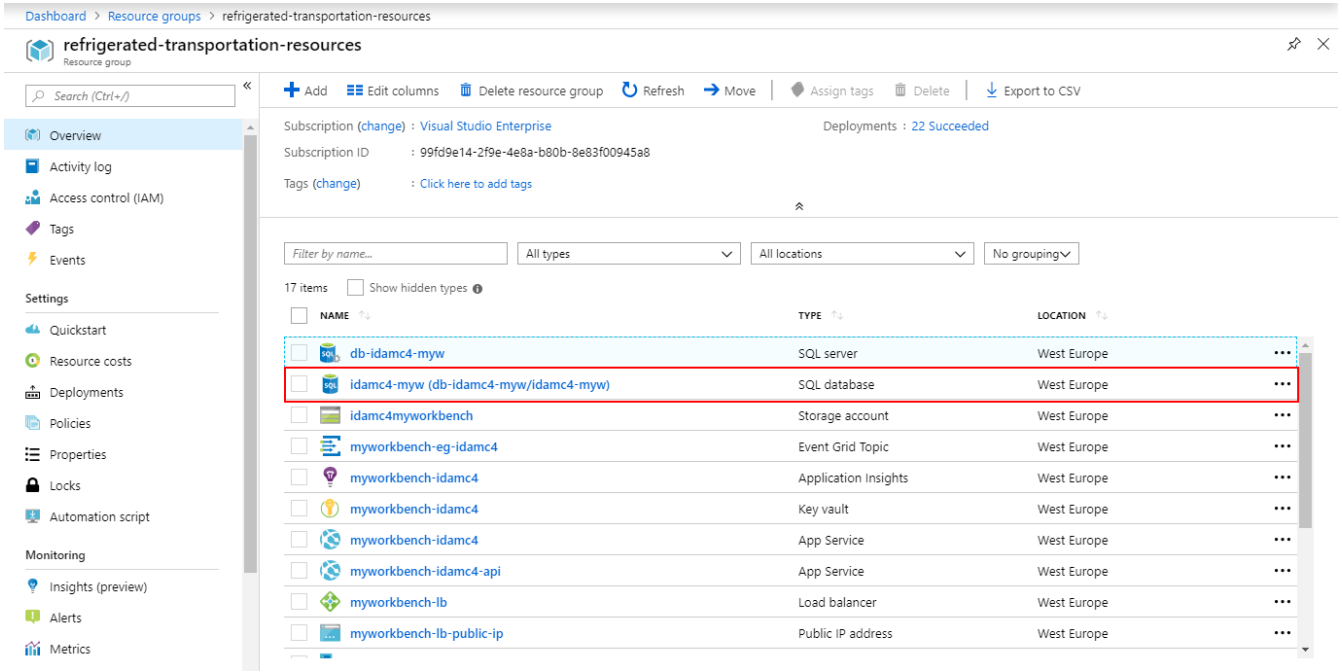
In this part, you will need to deploy a Logic App that will be in charge of processing messages when they are received by the Service Bus queue.

Before that, you will need to first create a stored procedure, because at some point of the Logic App, you will map the device that sends a message to its Azure AD identity.

Creating the required stored procedure

Perform the following steps:

1. Go on **Resource groups**, then the resource group where your Azure Blockchain Workbench is, and search for the database.

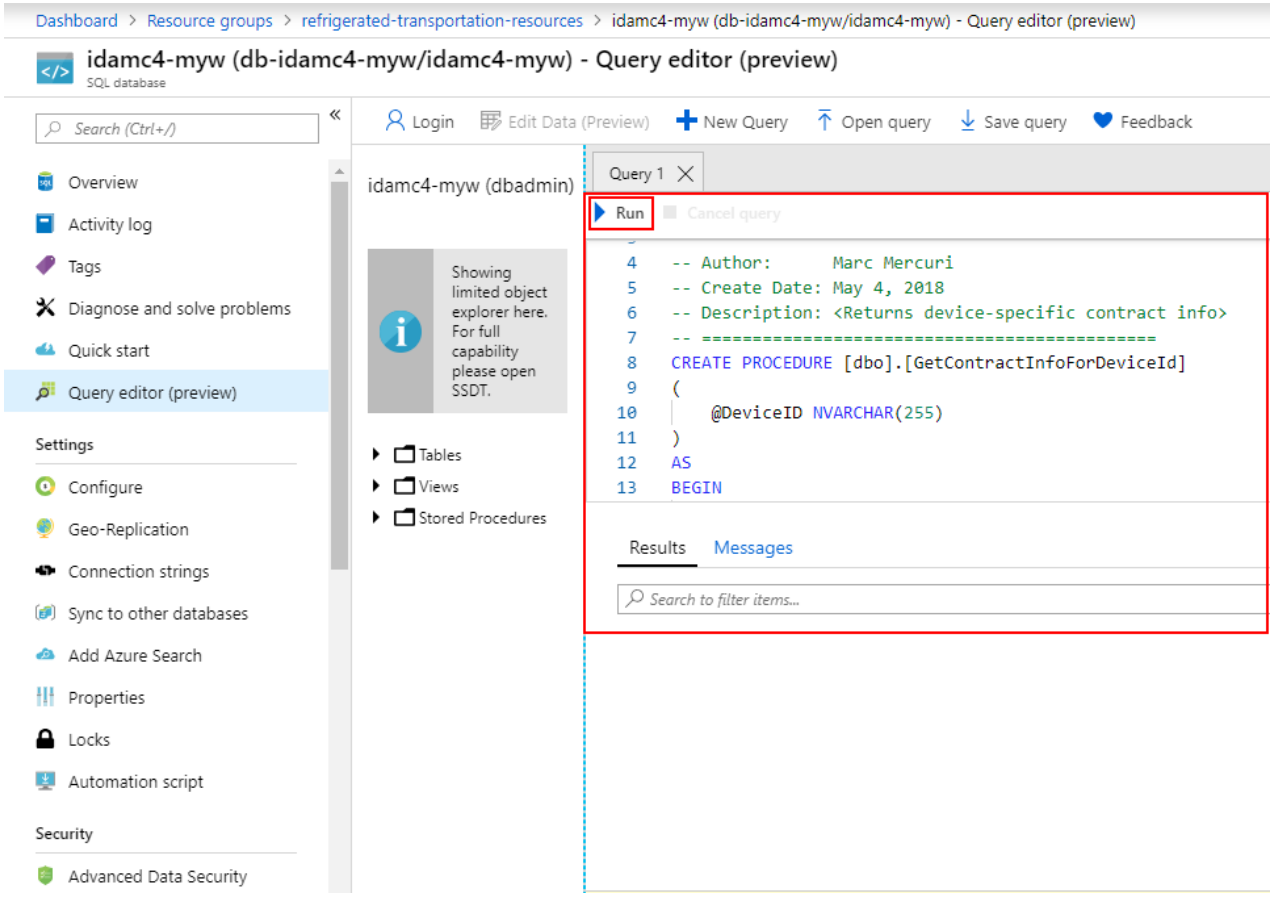


2. Click on it, and inside it, click on **Query Editor**.
3. Sign in using your credential defined when you created your Workbench.
4. Open the *refrigerated-sc-sample* code for this guide that you downloaded before – you can download it or clone it here[40] if you don't have done yet -, then paste the command below and click on **Run**.

```
ALTER TABLE [User] ADD ExternalDeviceId VARCHAR(255);
```

[40] Guide sample code: https://aka.ms/ABWDevGuideSamples

5. Verify the success of the request by checking the line below the editor. You should see *Query succeeded: Affected rows: 0*.
6. Open the *getDeviceContracts.sql* file, which is located inside *IoT setup* directory, paste its content into the editor, and then click on **Run**.



7. Verify the success of the request by checking the line below the editor. You should see *Query succeeded: Affected rows: 0*.

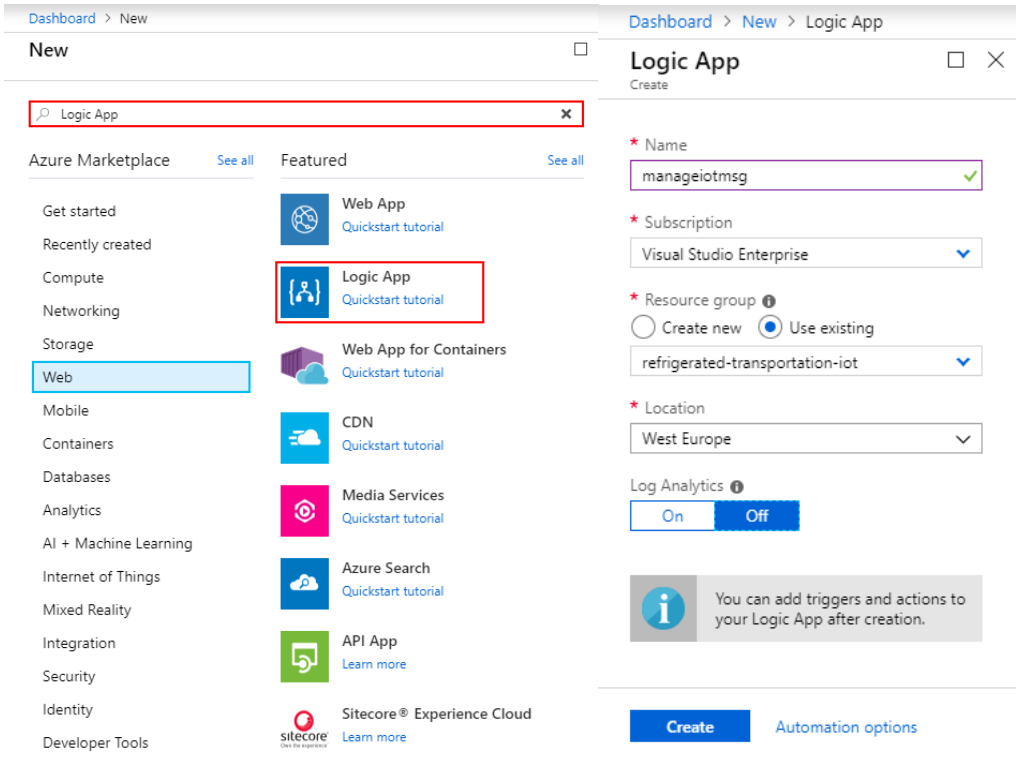
Your request is now available to use. You are ready to create your (first) Logic App.

Creating the Logic app

Perform the following steps:

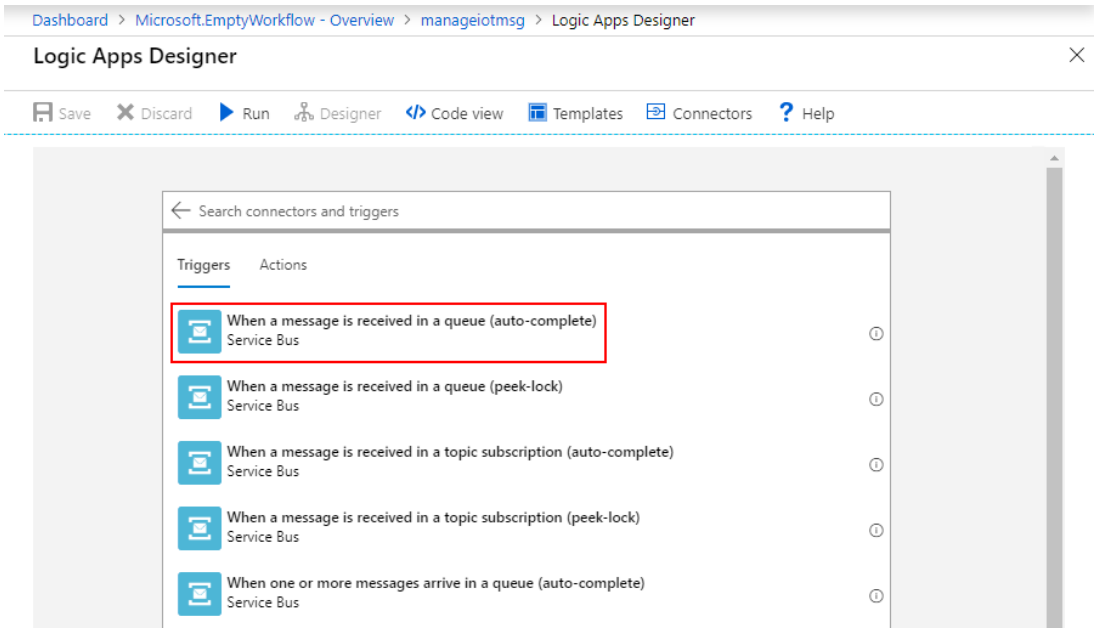
> **Note** When creating a Logic app, you have the possibility to rename each step of the app if you want to. This is useful if you want to give a more explicit name, for example to name a step "Get contract parties" instead of the default name "Execute a stored procedure".

1. Sign in to the Azure portal.
2. Choose +**Create a resource**, then choose **Web**.
3. Click **Logic App** from the list on the right.
4. Fill the **Configuration** tab.

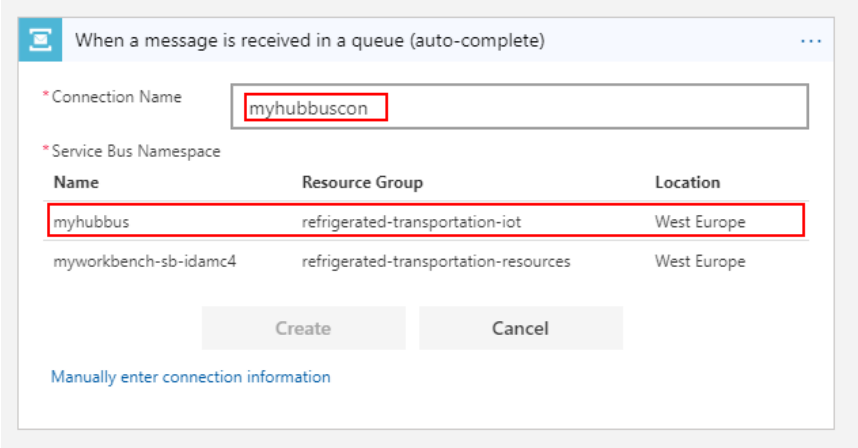


Setting	Description
Name	Specify a name for your future app.
Subscription	Select the subscription to use for your Logic App.
Resource group	You can create a new resource group or use an existing one. To create a new one, click **Create** new and fill in the name you want to use. To use an existing resource group, click **Use existing** and select the resource group from the dropdown list.
Location	This is the region in which you want your Logic App to be located. Select the location closest to you from the dropdown list.
Log Analytics	Keep the **Off** parameter for diagnostic logging.

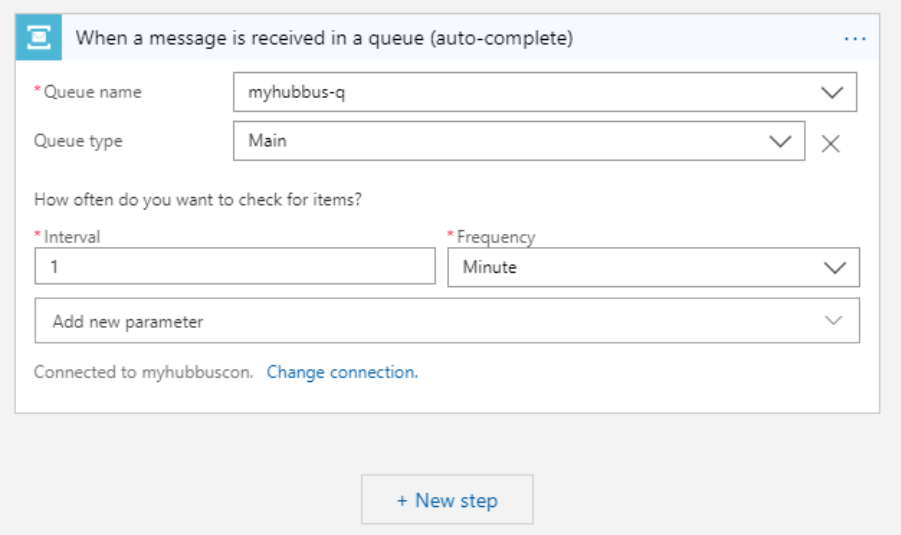
5. Once the creation of the Logic App is completed, go inside the **Logic App Designer**. Click on **Service Bus** and search for **When a message is received in a queue (auto-complete)**, then click on it.



6. Create a connection to the Service Bus by entering a name and selecting the Service Bus in the list.



7. Click on the Service Bus Policy named RootManageSharedAccessKey and then click on **Create**.
8. In the next step, change the trigger interval from the default value to **1 minute,** then click on **+ New Step**.

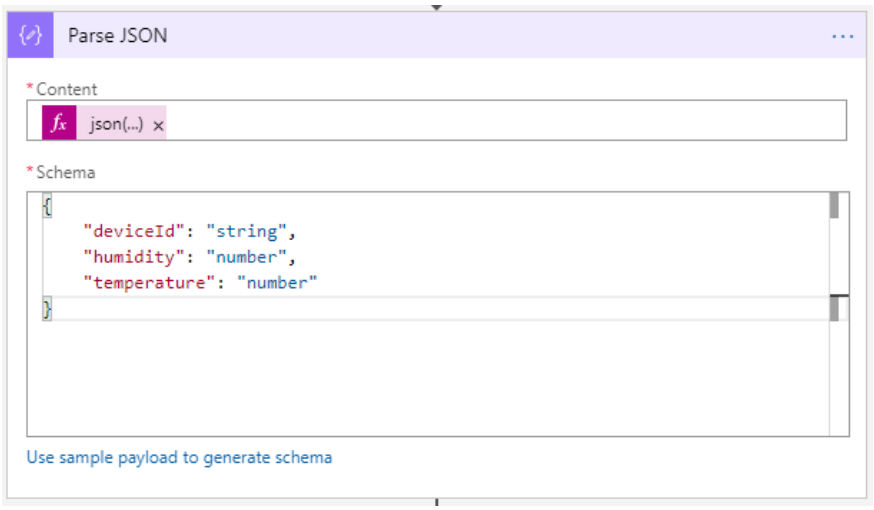


9. Search for **Parse JSON** action, and then paste into the content field the following code snippet:

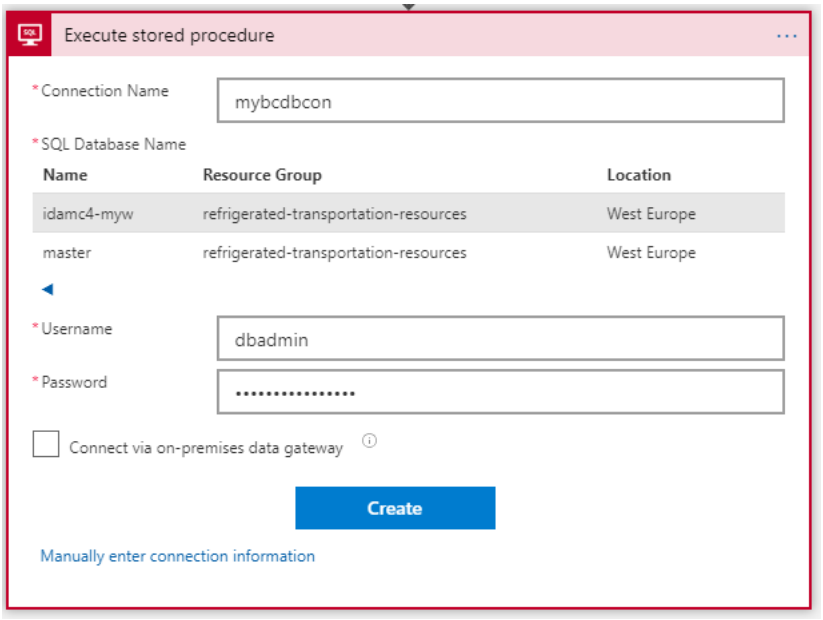
```
json(base64ToString(triggerBody()?['ContentData']))
```

10. And, into the schema property, paste the code snippet below. Finally, click on **+ New step**.

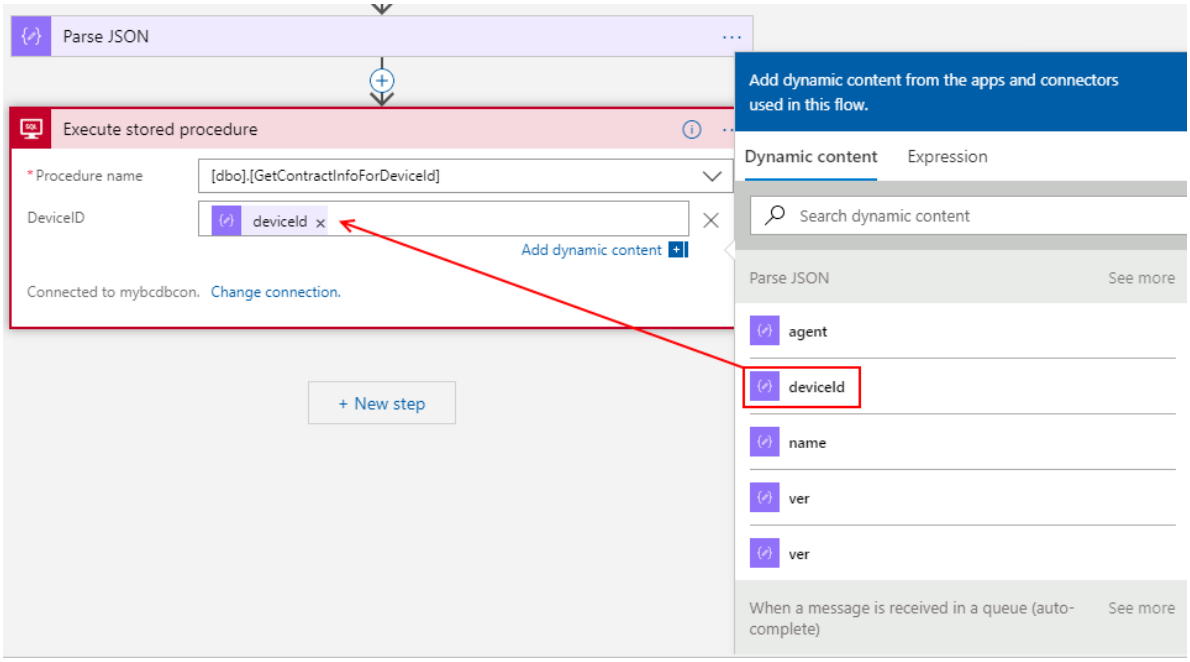
```
{
    "type": "object",
    "properties": {
        "deviceId": {
            "type": "string"
        },
        "temperature": {
            "type": "integer"
        },
        "humidity": {
            "type": "integer"
        }
    }
}
```



11. Search for "*SQL Server*", then select the **Execute stored procedure** action, and click on it. To use it, you need to establish a connection between the Logic App and your Workbench database. Enter a connection name, select the database and enter your credentials.



12. Search and click on the procedure **[dbo].[GetContractInfoForDeviceId].**
13. Click on **Add new parameter** and select the only one available, i.e. **DeviceID**. If you click on the blank field that appears, a new tab will open. Search for "*deviceId*" and select it.

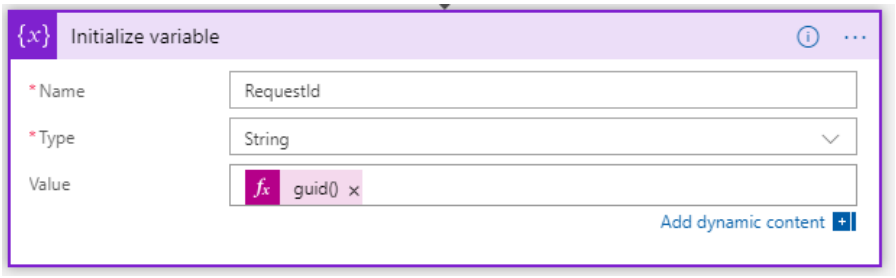


14. Verify, by passing your cursor on the variable `deviceId` that its value is set to *"body('Parse_JSON')['deviceId']"*, and then click on **+ New Step**.

In the case that the value is different, remove `deviceId` and click in the field to add dynamic content, then paste the correct value in the **Expression** tab, and click on **OK**.

15. Select the **Initialize Variable** action and fill it.

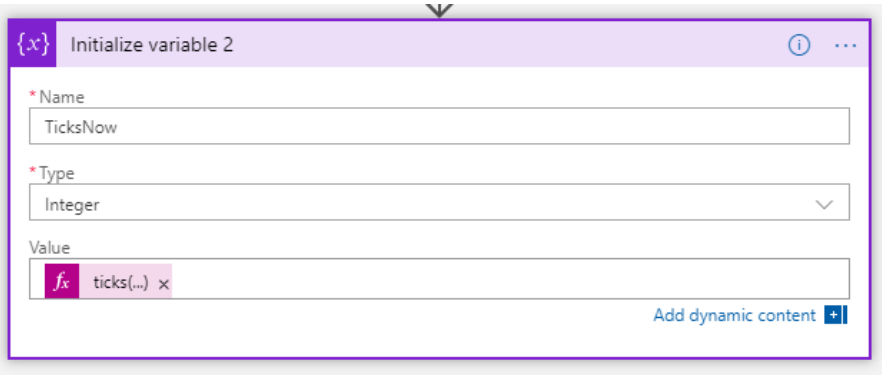
    - Name: `RequestId`
    - Type: String
    - Value: Click on the field, go in the **Expression** tab and type *"guid()"*, and then press ENTER.



Finally, click on **+ New step**.

16. Select the **Initialize variable** and fill it.

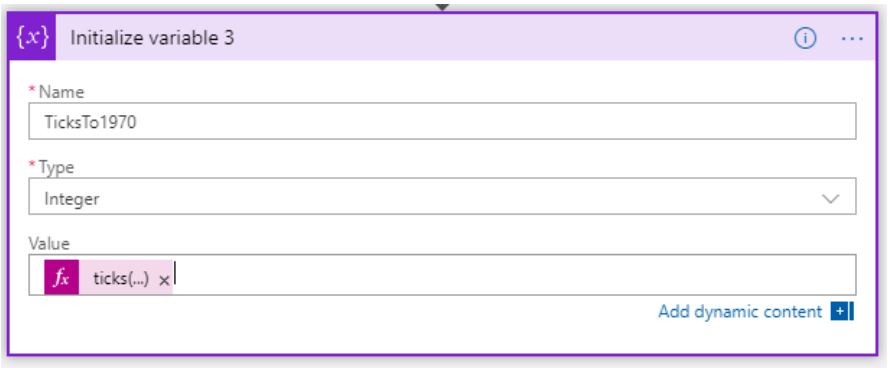
    - Name: `TicksNow`
    - Type: Integer
    - Value: Click on the field, then go in the **Expression** tab and type *"ticks(utcNow())"*, and then press ENTER.



Finally, click on **+ New step**.

17. Select the **Initialize variable** and fill it.

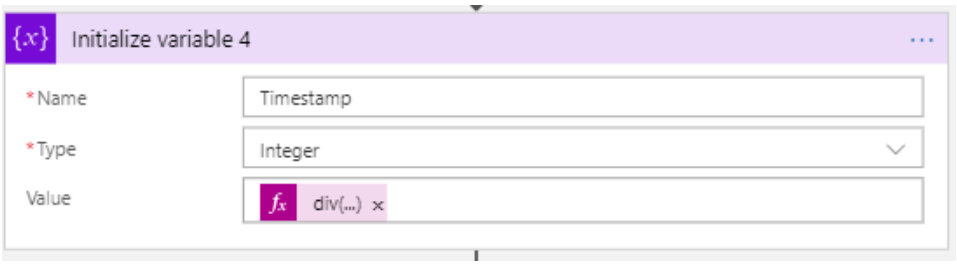
    - Name: `TicksTo1970`
    - Type: Integer
    - Value: Click on the field, go in the **Expression** tab and type *"ticks('1970-01-01')"*, then press ENTER.



Finally, click on **+ New step**.

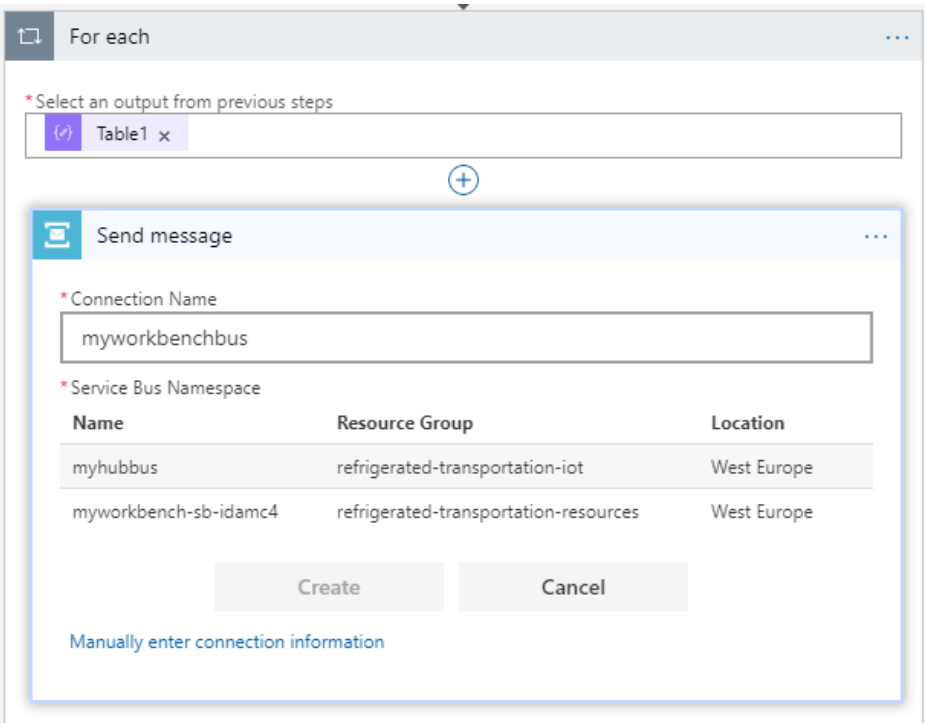
18. Select the **Initialize variable** and fill it.

    - `Name: Timestamp`
    - Type: Integer
    - Value: Click on the field, go in the **Expression** tab and type *"ticks(div(sub(variables('TicksNow'),variables('TicksTo1970')),10000000)"*, and then press ENTER. The difference between those two variables generates a timestamp.



Finally, click on **+ New step**.

19. Select the **Control** action. Click on the **Content** field, search and click for "*For each*". Click on the **output** field and select **Table1**, and then click on **Take an action**.
20. Inside the new action within the *For each* block, click on **Service Bus**, tand hen search for **Send message** action. Check the connection at the bottom of the block. If this is the same connection as the first block of your Logic App, **click on change connection**. If not, you will be prompted to create a new connection.
21. Click on **Add new connection**, then select your Workbench as the data source and name it. In the next tab, click on **RootManageSharedAccessKey**, and finally, click on **Create** to establish the connection.



22. To finish, you must aggregate the device data, the timestamp variable and the Workbench information (like userId, contractID, etc.) to get one complete message which can be used by Azure Blockchain Workbench.

    In the first dropdown menu, select **ingressqueue**. This is the queue used by the Workbench to receive messages from outside.

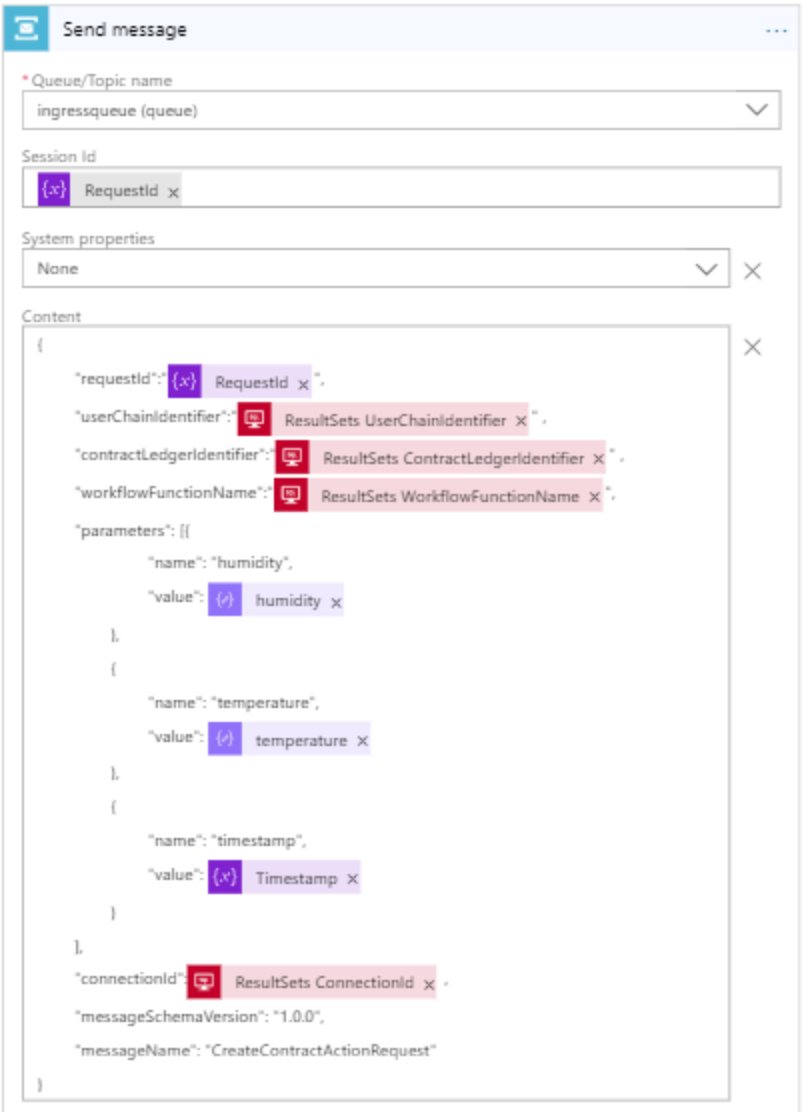
    Click on the **Session Id field**. Select the dynamic variable named `RequestId`.

    Now, you need to add the content of the message. Click on **Add new parameter** and check the box in front of **Content**. A text area will appear, paste the following code snippet inside.

```
{
    "requestId": "",
    "userChainIdentifier": "",
    "contractLedgerIdentifier": "",
    "workflowFunctionName": "IngestTelemetry",
    "Parameters": [
        {
            "name": "humidity",
            "value": ""
        },
        {
            "name": "temperature",
            "value": ""
        },
        {
            "name": "timestamp",
            "value": ""
        }
    ],
    "connectionId": 1,
    "messageSchemaVersion": "1.0.0",
    "messageName": "CreateContractActionRequest"
}
```

Finally, you need to push your variables inside this message. Replace every empty double-quotes string by those variables.

To do so, place your cursor at the center of each double-quotes field and in the **dynamic variables** tab, search for the variable which corresponds to the key. Some variables can be hidden because the Logic App considers them as out of context, but you can display it by clicking on **See more** in the **Dynamic content** tab.



Your Logic app is now complete and ready to forward messages to Azure Blockchain Workbench. Now, at this point, you need to set up some IoT devices to test your application and verify that a device can trigger an "out-of-compliance" state update if temperature or humidity is too high.

Adding IoT devices to your solution

You are now ready to connect external devices to your solution. For the sake of this guide, you will see how to connect and provision an IoT device, and how to send messages to your IoT Hub on four different types of devices:

1. A Node.js simulated device,
2. An MXChip IoT DevKit device (optional),
3. An Azure Sphere DevKit device (optional),
4. A Raspberry Pi device (optional).

The Node.js simulated device enables to cover all the intended integration with Azure Blockchain Workbench, and is sufficient by itself.

Step-by-step directions are also provided for the most common testing devices with IoT Hub but related sections are optional. So you can any of them if you don't have that specific device or the sensor(s).

Using a Node.js simulated device

Overview and prerequisites

The first "device" that you will implement is a Node.js simulated device. It will take the form of a computer CLI program, which will send fake temperature/humidity values at regular intervals.

You can find more documentation about sending telemetry and remotely configuring a device to this link[41].

Before diving into the "device" configuration, check that Node.js is correctly installed on your machine, and also that its version is above v4.x.x as stated in section § GUIDE PREREQUISITES.

You can enter the following command into a console to verify it.

```
node -v
```

Also, open the *refrigerated-sc-sample* that you downloaded before with your console - if you don't have it yet, you can download it or clone it from here[42] -, and once inside the root folder, go through *IoT setup* > *simulated-device*, then type:

```
npm install
```

This will install all the required packages for your application, like Azure IoT device.

Registering the simulated device

To be fully functional, your device needs to sign in to the IoT Hub. This is why you need to create first an identity for your device to work, by adding it to the Hub device list.

Perform the following steps:

1. Launch an Azure Cloud Shell session, by clicking on its button at the top of Azure tab. If asked, choose PowerShell over Bash.

2. Inside the Azure Cloud Shell console, run the following command by replacing *YourIoTHubName* by the name of your IoT Hub and *SimulatedDevice* by a custom name if you want to change it.

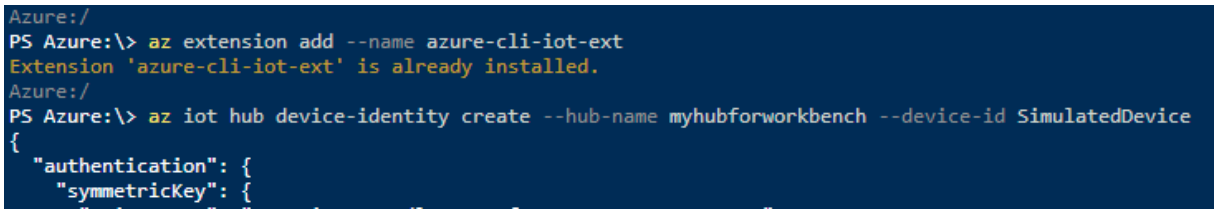
---

[41] QUICKSTART: SEND TELEMETRY FROM A DEVICE TO AN IOT HUB AND READ IT WITH A BACK-END APPLICATION (NODE.JS): https://docs.microsoft.com/en-us/azure/iot-hub/quickstart-send-telemetry-node

[42] Refrigerated Supply Chain samples: https://aka.ms/ABWDevGuideSamples

```
az extension add --name azure-cli-iot-ext
az iot hub device-identity create --hub-name YourIoTHubName --device-id SimulatedDevice
```

You should get a JSON authentication object as a successful response. Your device's identity is now created.



3. You now need to get the connection string that your device will use to connect. Run the following command in your Azure Cloud Shell console.

```
az iot hub device-identity show-connection-string --hub-name YourIoTHubName --device-id
SimulatedDevice --output table
```

Make a note of the device connection string. It looks like the following:

```
HostName=YourIoTHubName.azure-
devices.net;DeviceId=SimulatedDevice;SharedAccessKey={YourSharedAccessKey}
```

4. Go in the root folder of the simulated device and open **SimulatedDevice.js** with your code editor of choice. Search for the connection string line and replace it with your own connection string.

5. Finally, you need to link your device ID to an existing account in your Azure AD tenant, See section § CREATING A USER IN YOUR AZURE AD TENANT. You are going to use the user iot-device-1@<your tenant>.onmicrosoft.com that you've created before. Go on **Resource groups**, then the resource group where your Azure Blockchain Workbench is, and search for the database.
6. Click on it, and inside it, click on **Query Editor**.
7. Sign in using your credential defined when you created your Workbench.

   Inside it, run the following command:

```
Update [User] Set ExternalDeviceId = '<Your device ID>' where EmailAddress = 'iot-device-1@<your
tenant>.onmicrosoft.com'
```

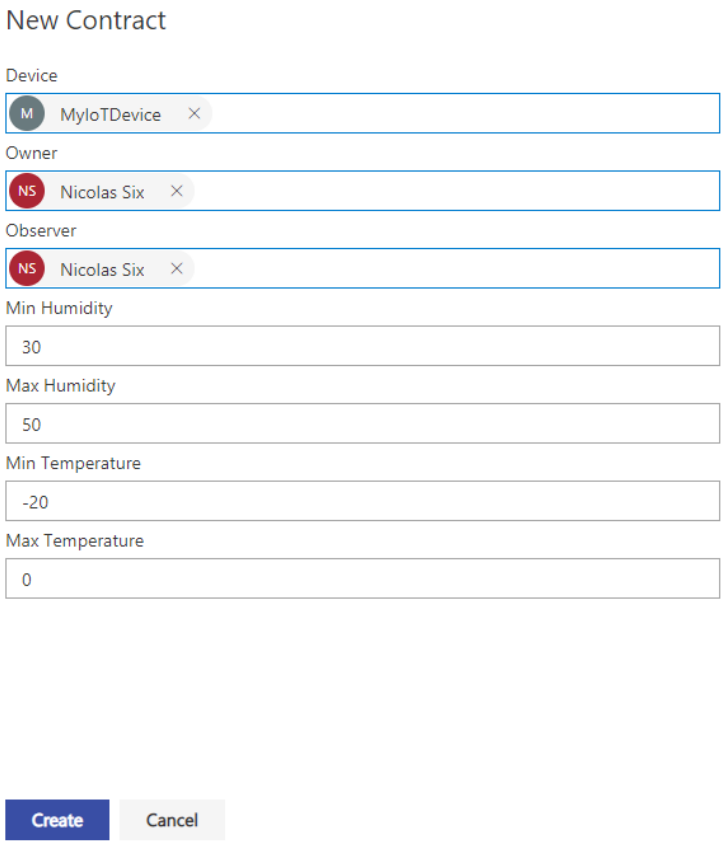
8. Check that the command has been correctly executed by looking into the messages tab below the Query Editor. It should be *Query succeeded: Affected rows: 1*.

Your device is now correctly registered into your Azure IoT Hub but also into your Azure AD tenant and therefore Azure Blockchain Workbench. So, you are ready to test it!

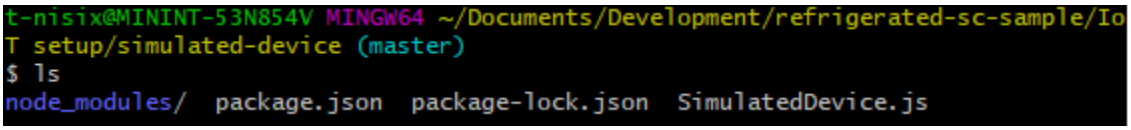
Sending simulated telemetry and testing the application

Perform the following steps:

1. Go inside your Workbench application on your browser and click on your deployed **Refrigerated Transportation** application.
2. On the main application tab, click on **+ New** to create a new contract. Fill all the persona fields with your identity except for the device that you will fill with your IoT device. Set the temperature between -20 and 0 and the humidity between 30 and 50 in order to respect simulated value ranges for the test. Finally, click on **Create** to deploy the contract.



3. Once the contract is deployed, you are ready to start the simulated device. In your computer, start a terminal console and navigate to the root folder of the simulated device.

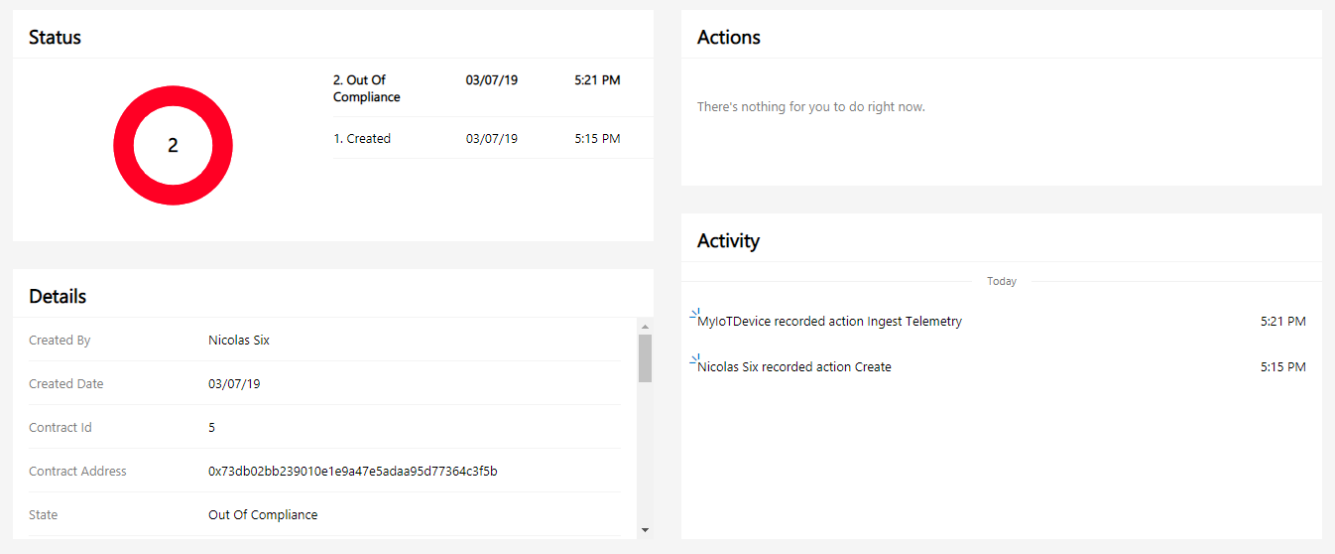


4. In this terminal, type those commands.

```
npm install
node SimulatedDevice.js
```

The program will start and simulate a temperature measurement every 3 seconds. The temperature will be between -20 and 2, and the humidity between 30 and 52.

You will see that simulated values can go above required values, and,as expected, this will trigger an 'out-of-compliance' state update. Let the program run until an out-of-bounds value is sent and check in your application the state of your contract, which should be *'Out-of-Compliance'*.



Congrats! Your simulated device is up and running. You should now have a good understanding of all the processes that occur between your device and the Workbench. You are now ready to apply the same pattern to other physical devices. This is the purpose of the next sections. They are all optional.

Using a MXChip IoT DevKit device (optional)

Overview and prerequisites

In this section, you are going to set up your first physical device, the MXChip IoT Devkit device.

The MXChip IoT DevKit is an all-in-one IoT device kit. You can use it to develop and prototype IoT solutions that leverage Microsoft Azure services.

It includes an Arduino-compatible development board with rich peripherals and sensors, an open-source board package, and a growing projects' catalog. You can find more documentation about DevKit setup at this link[43].

To program the device, you will also use Arduino IDE. If you haven't downloaded it or cloned it yet, you can grab it here[44].

[43] CONNECT IOT DEVKIT AZ3166 TO AZURE IOT HUB: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-arduino-iot-devkit-az3166-get-started

[44] Arduino: https://www.arduino.cc/en/Main/Software

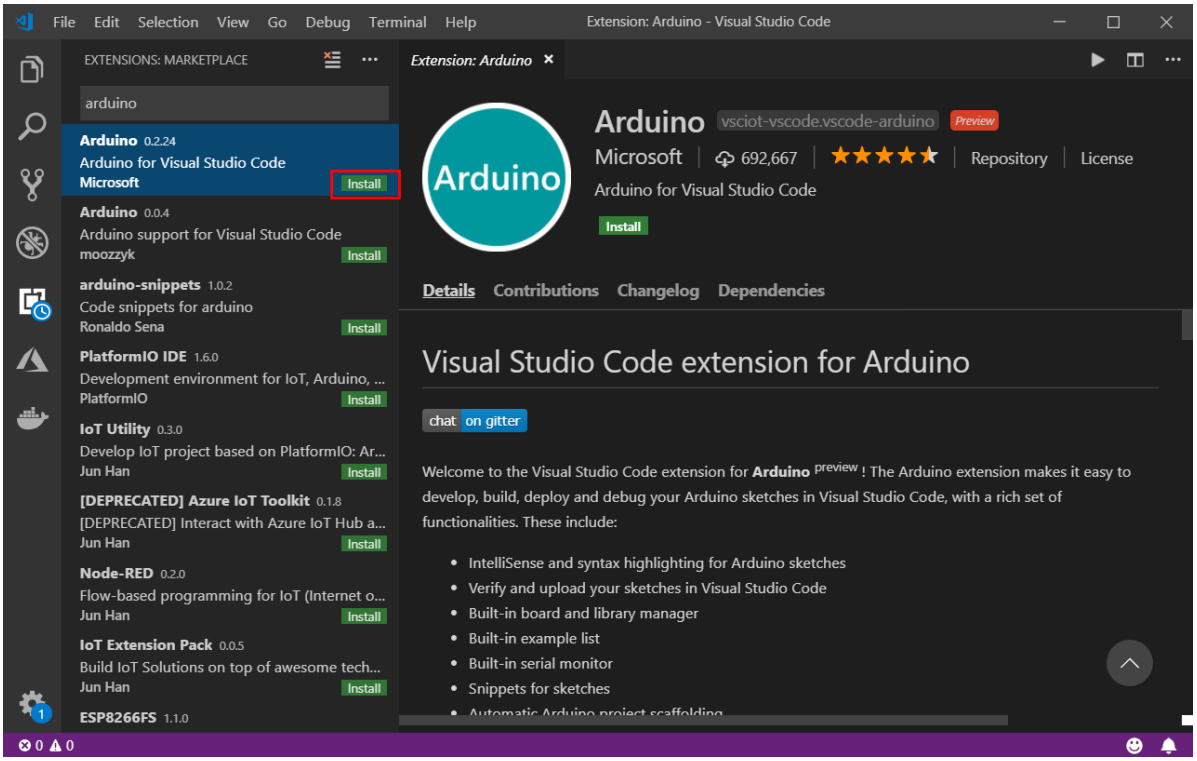
You will also have to install ST-Link drivers, to enable USB communication between the MXChip IoT DevKit and your computer:

- **Windows**. Download and install the USB driver from STMicroelectronics website.
- **MacOS**. No driver is required for MacOS.
- **Ubuntu**. Run the following command in terminal console and sign out and sign in again for the group change to take effect:

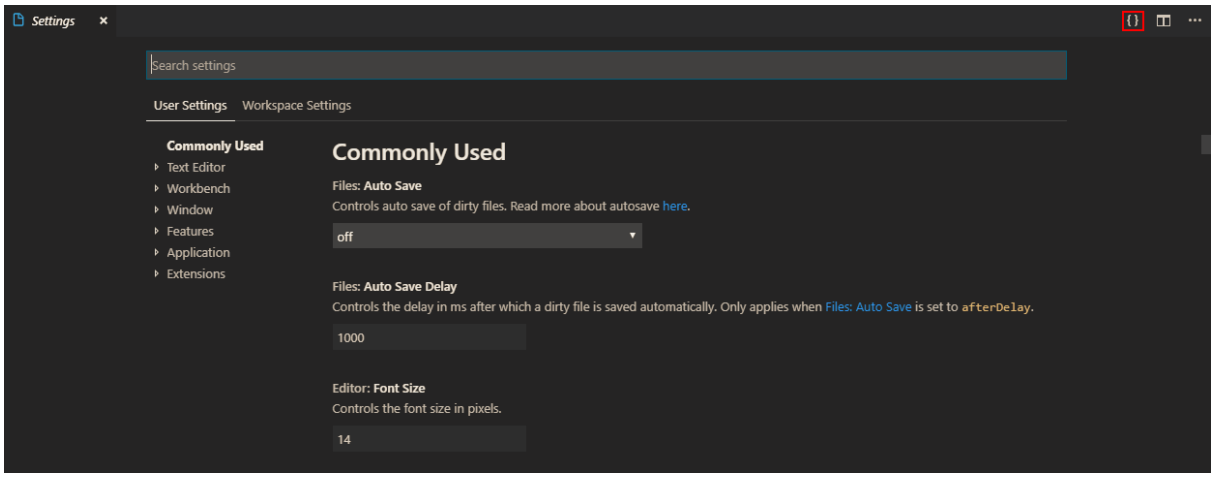
```
# Copy the default rules. This grants permission to the group 'plugdev'
sudo cp ~/.arduino15/packages/AZ3166/tools/openocd/0.10.0/linux/contrib/60-openocd.rules
/etc/udev/rules.d/
sudo udevadm control --reload-rules

# Add yourself to the group 'plugdev'
# Logout and log back in for the group to take effect
sudo usermod -a -G plugdev $(whoami)
```

Finally, you will need to add some extensions to Visual Studio Code in order to support Arduino and Azure development.

Launch Visual Studio Code, search for "*Arduino*" in the extensions marketplace and install it. This extension provides enhanced experiences for developing on the Arduino platform.



Configure Visual Studio Code with Arduino settings. In Visual Studio Code, click **File > Preference > Settings** ,and then click the **{} icon** to open the file *settings.json*.



Add the following lines to configure Arduino depending on your platform:

- **Windows**:

```
"arduino.path": "C:\\Program Files (x86)\\Arduino",
"arduino.additionalUrls":
"https://raw.githubusercontent.com/VSChina/azureiotdevkit_tools/master/package_azureboard_index.json"
```

- **macOS**:

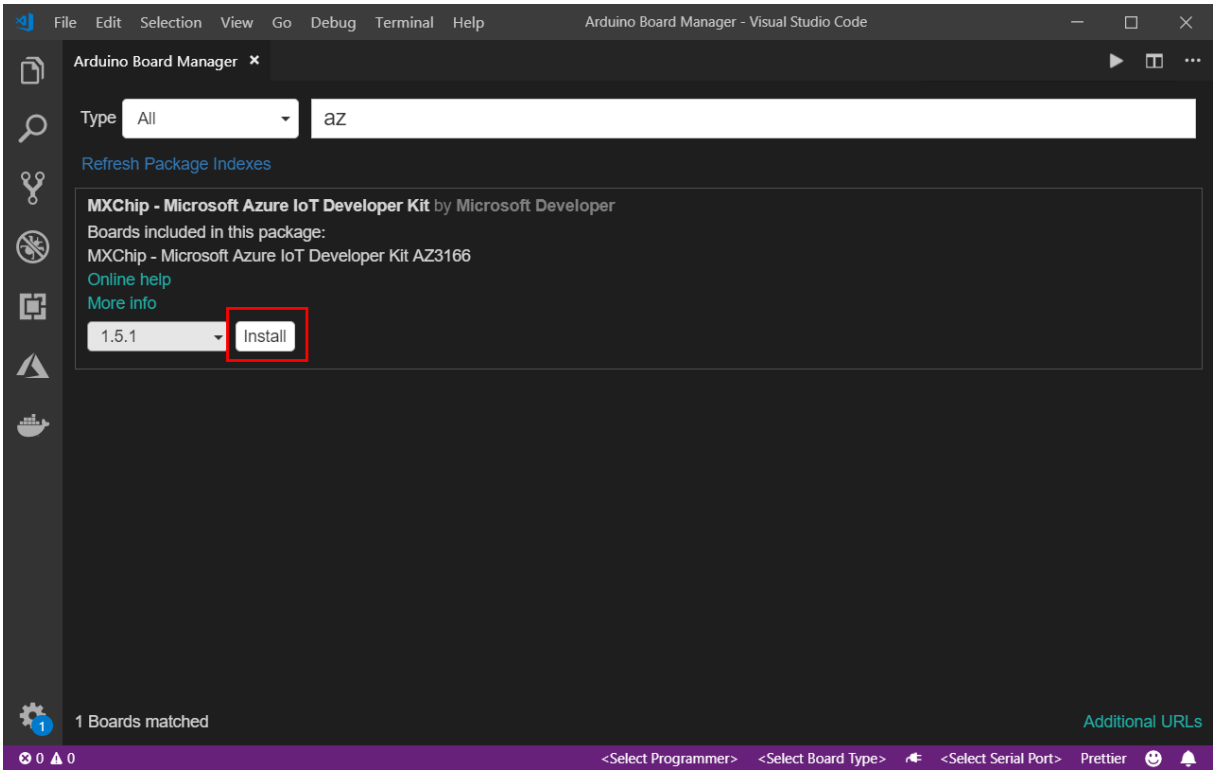
```
"arduino.path": "/Applications",
"arduino.additionalUrls":
"https://raw.githubusercontent.com/VSChina/azureiotdevkit_tools/master/package_azureboard_index.json"
```

- **Ubuntu**:

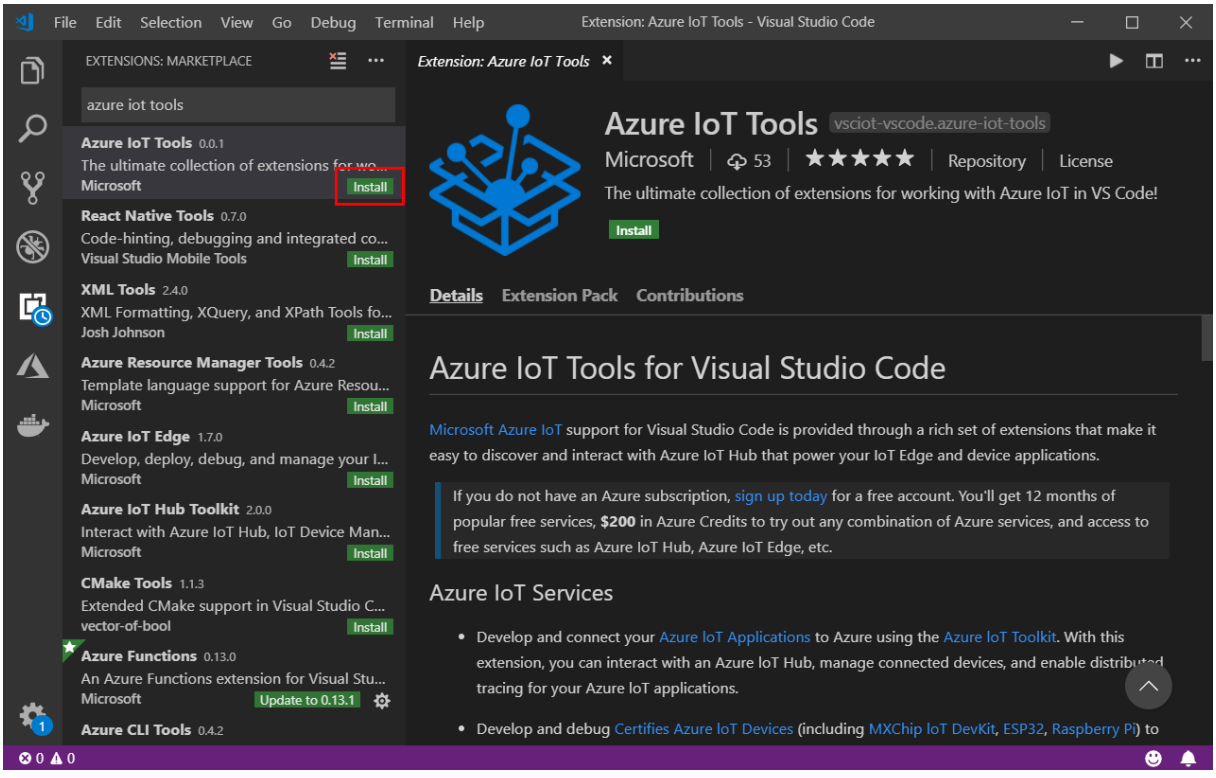
  Replace the **{username}** placeholder below with your username.

```
"arduino.path": "/home/{username}/Downloads/arduino-1.8.8",
"arduino.additionalUrls":
"https://raw.githubusercontent.com/VSChina/azureiotdevkit_tools/master/package_azureboard_index.json"
```

Press F1 to open the command palette, type and select **Arduino: Board Manager**. Search for "*AZ3166*" and install the latest version.



Then search for "*Azure IoT Tools*" in the extensions marketplace and install it.

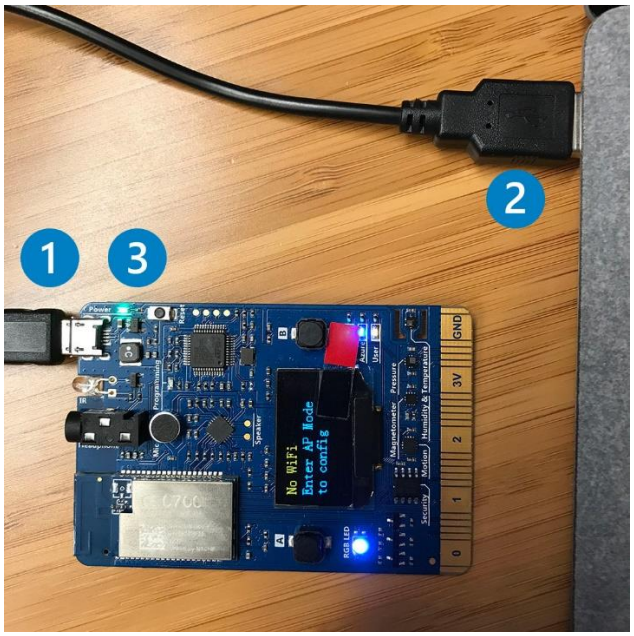


Now that everything is installed, you are ready to open the project and prepare it for your device.

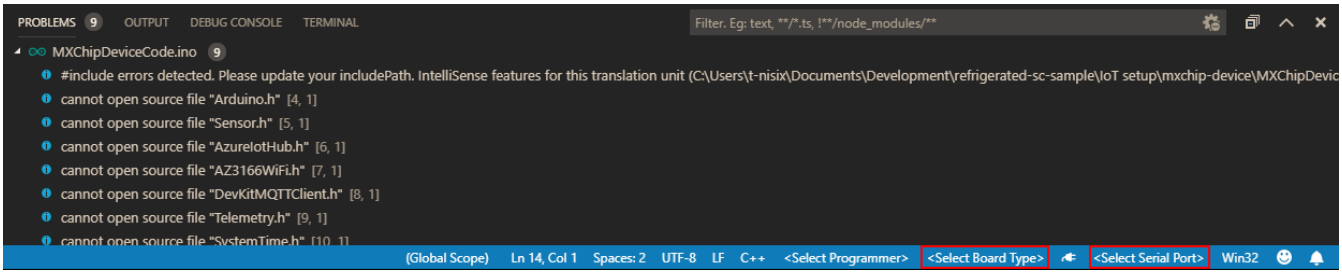
Setting up the project and the device

Perform the following steps:

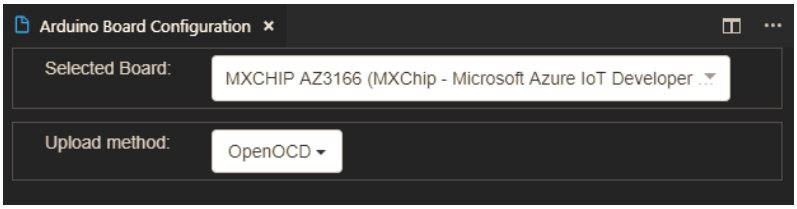
1. In Visual Studio Code, open the *mxchip-device.code-workspace* which is inside *refrigerated-sc-sample* > *IoT setup* > *mxchip-device* directory.
2. Once opened, connect the MXChip device to your computer by plugging the USB connector to your computer (1) and the Micro-USB connector to the device (2). The LED (3) should turn on if the connection succeeded.



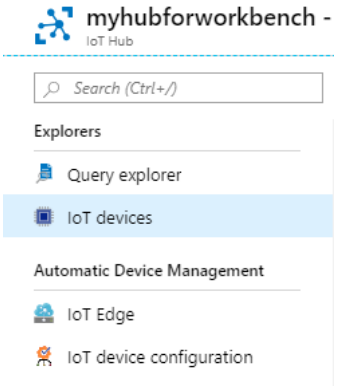
3. Now that the device is plugged in, and if they are not specified yet by Visual Studio Code, you can specify which board and port you're using to program the DevKit. In Visual Studio Code at the bottom of the screen, click on **<Select Serial Port>**, then select **COM3**.



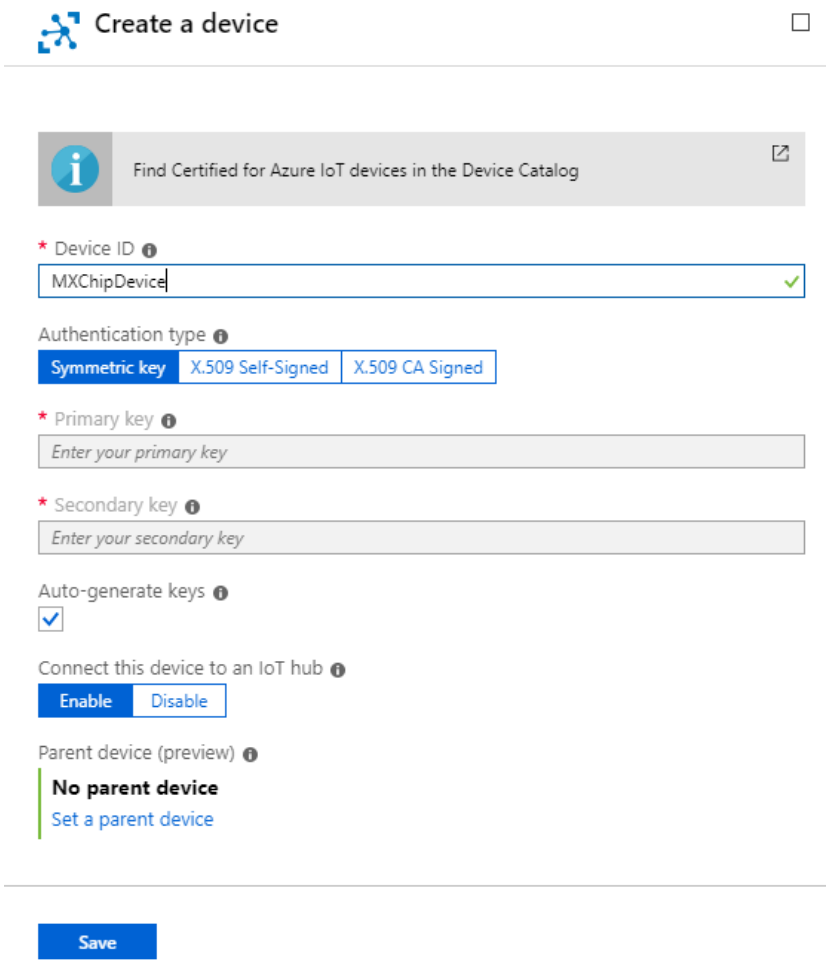
4. Click on **<Select Board Type>.** An Arduino Board Configuration tab will open, search for "*MXCHIP AZ*" and select the one which appears. Leave the default for the other parameters.



5. Now, you need to set up the connection between Azure (your IoT Hub) and your DevKit. Open your **Azure administration panel** on your browser, and search for your **IoT Hub**. Once inside it, search for **IoT devices** and click on it.

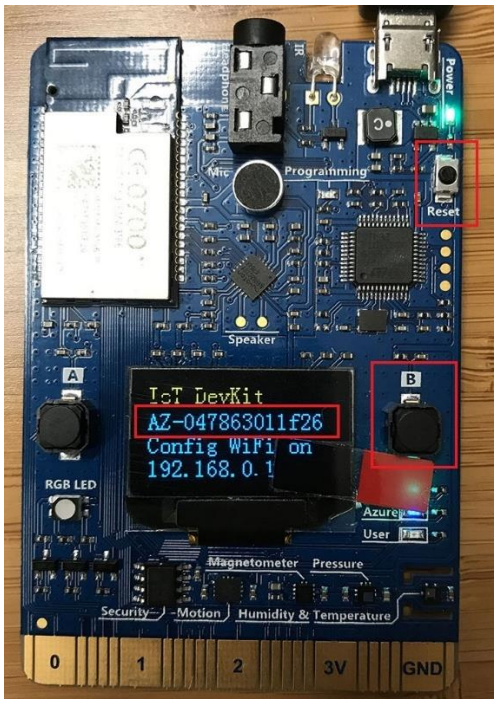


6. Click on **+ Add**. A tab will open, fill the **Device ID** *field* with "*MXChipDevice*" and click on **Save**.

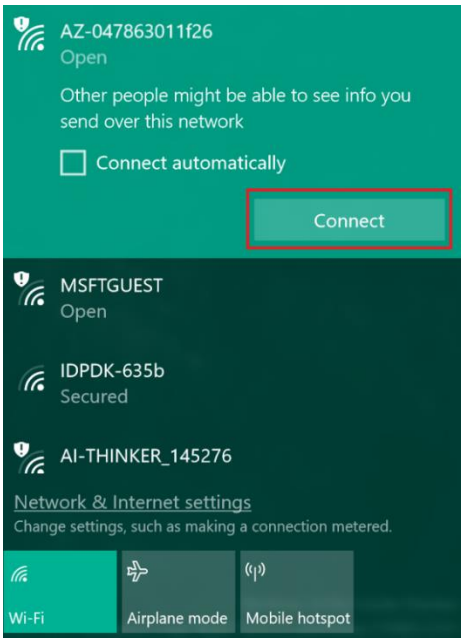


Note The Device ID is set to "MXChipDevice" here for convenience because you will find the same ID inside the code of the device, but you can change it if you want to. Simply don't forget to also change it inside the code and the SQL request that you will see below.

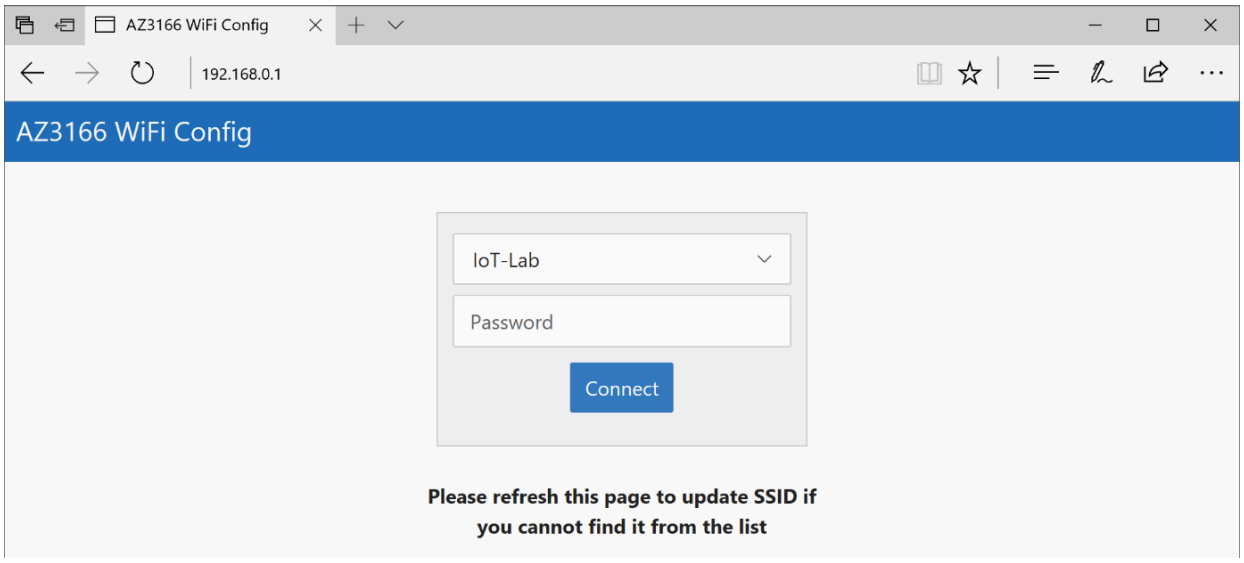
7. You've been redirected to your IoT devices list, click on your newly created device and inside the details, copy the **Connection string (primary key)**.
8. Go back into Visual Studio Code and press F1 to open the command palette, type and select **Azure IoT Device Workbench: Configure Device Settings...**, then select **Config Device Connection String > Select IoT Hub Device Connection String**. When prompted, paste the connection string inside the field and press **enter**.
9. On DevKit, hold down **button A**, push and release the **reset** button, and then release **button A**. Your DevKit enters configuration mode and saves the connection string.
10. Press F1 again, type and select **Azure IoT Device Workbench: Upload Device Code**. The code will be compiled and then uploaded to the DevKit.
11. The last step for the configuration of your device is to enable the Wi-Fi connection. Hold down button B, push and release the reset button, and then release button B. Your DevKit enters AP mode for configuring Wi-Fi. The screen displays the service set identifier (SSID) of the DevKit and the configuration portal IP address.



12. Now, use another Wi-Fi enabled device (computer or mobile phone) to connect to the DevKit SSID (highlighted in the previous image). Leave the password empty.



13. Open the IP address shown on the DevKit screen on your computer or mobile phone browser, select the Wi-Fi network that you want the DevKit to connect to, and then type the password. Select **Connect**.



14. Finally, you need to link your device ID to an existing user account in your Azure AD tenant, see section § CREATING A USER IN YOUR AZURE AD TENANT. You are going to use the user iot-device-2@<your tenant>.onmicrosoft.com that you've created before. Go on **Resource groups** in your Azure administration panel, then the resource group where your Azure Blockchain Workbench is, and search for the database.
15. Click on it, and inside it, click on **Query Editor**.
16. Sign in using your credential defined when you created your Workbench.

    Inside it, run the following command:

```
Update [User] Set ExternalDeviceId = 'MXChipDevice' where EmailAddress = 'iot-device-2@<your
tenant>.onmicrosoft.com'
```

17. Check that the command has been correctly executed by looking into the messages tab below the Query Editor. It should be *Query succeeded: Affected rows: 1.*

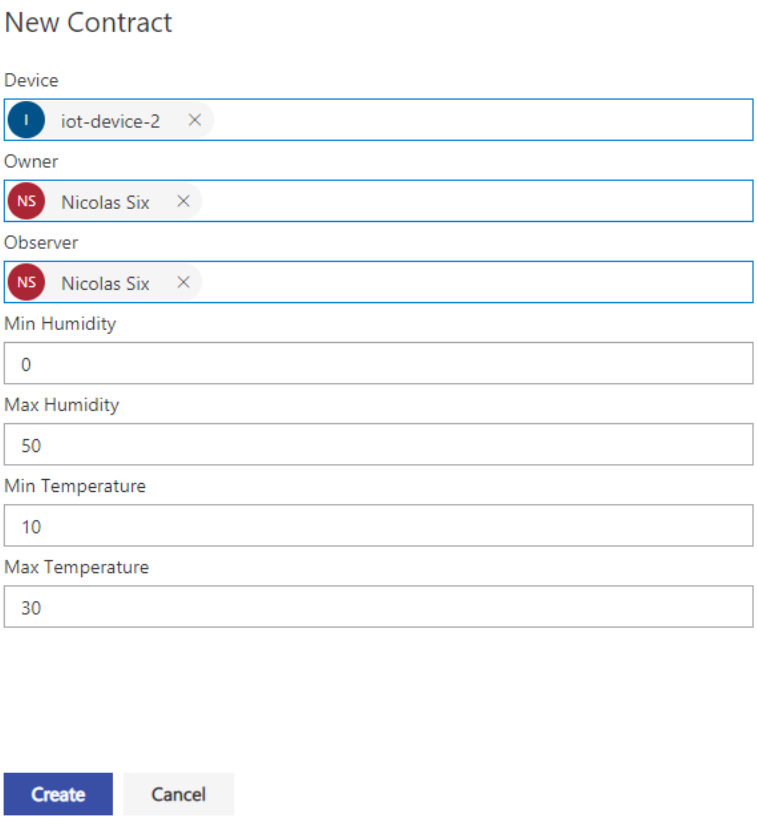
Everything is now set up, let's try your device!

Testing the device

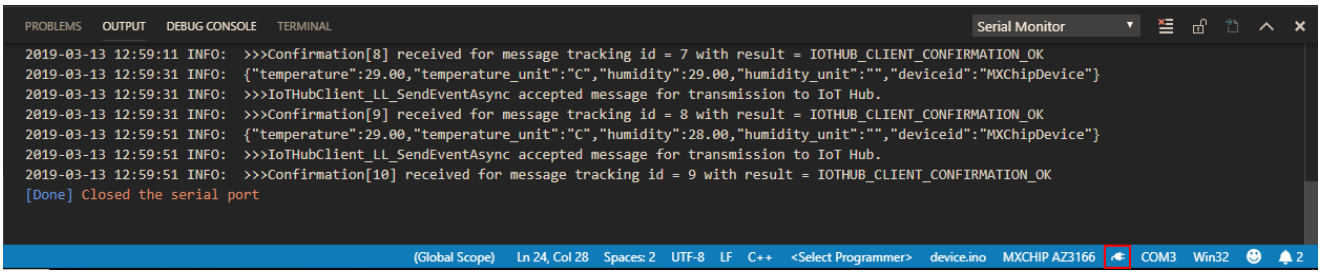
Perform the following steps:

1. Go inside your Workbench application on your browser and click on your deployed **Refrigerated Transportation** application.
2. On the main application tab, click on **+ New** to create a new contract. Fill all the persona fields with your identity except for the device that you will fill with your IoT Device. Set the temperature between 10 and 30 and the humidity between 0 and 50 for testing. Finally, click on **Create** to deploy the Contract.

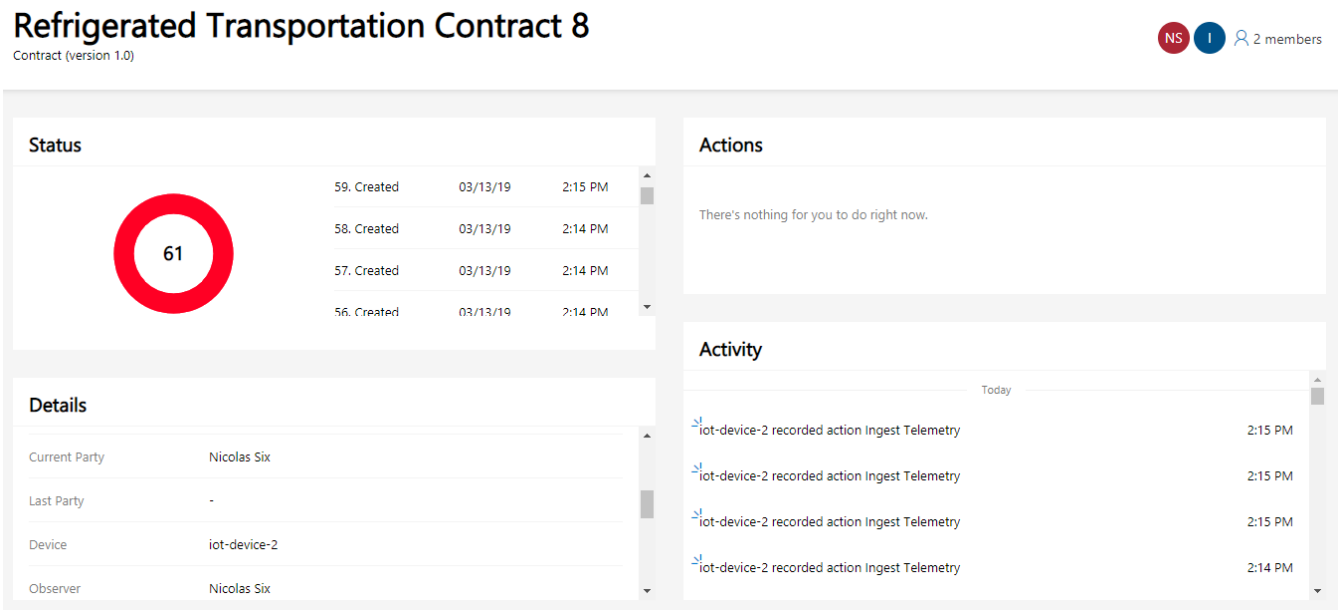
Note Of course, those values aren't realistic for refrigerated transportation, but one should guess that you will test your device to room temperature!



3. Once the contract is deployed, you are ready to start your device. If you let it plugged, the device is already sending telemetry to the Workbench, using the same routing method as your simulated device. You can see sent messages into the Visual Studio Code console, by clicking on the **plug icon** at the bottom of the screen.



4. Check in your Workbench contract if everything works fine. The IoT Device should appear in it, invoking **Ingest Telemetry** method. If you heat the temperature sensor on your device up to 30 degrees Clesius, you will see that the contract state will automatically change to *"Out-of-compliance"*.



The MXChip is now up and running, and as such, it greatly illustrate what could be achived with a sensor for supply chain monitoring.

Indeed, the sensor is sending values every 20 seconds, and everything is recorded inside the blockchain. If the temperature or the humidity reaches an "out-of-compliance" value, the contract state changes automatically and the participants inside the supply chain will know which counterparty is responsible for the perished goods.

Using an Azure Sphere MT3620 device (optional)

Overview and prerequisites

The second device that you will use is the Azure Sphere MT3620. Azure Sphere is a secured, high-level application platform with built-in communication and security features for internet-connected devices. It comprises an Azure Sphere microcontroller unit (MCU), tools and an SDK for developing applications, and the Azure Sphere Security Service, through which applications can securely connect to the cloud and web.

All the documentation about Azure Sphere is available at this link[45]. You can also find more samples on Seeed Studio website[46].

[45] AZURE SPHERE DOCUMENTATION: https://docs.microsoft.com/en-us/azure-sphere/
[46] Azure Sphere MT3620 Development Kit: http://wiki.seeedstudio.com/Azure_Sphere_MT3620_Development_Kit/

To setup this solution, you will need to:

- Get an Azure Sphere device (available for order here[47]), a MT3620 Grove Shield (available for order here[48]) and a Temperature & Humidity Sensor (available for order here[49]) if you don't have them yet.

Note An alternative is to pick a Development Kit which contains the Shield, the Temp&Humi Sensor and other types of sensors here[50].

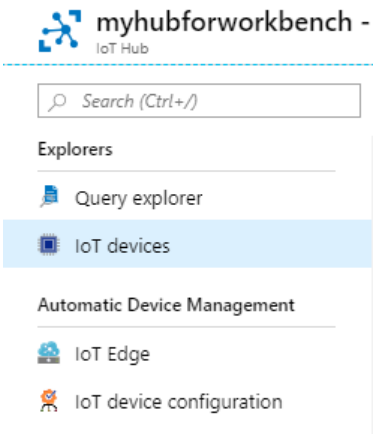
- Download Visual Studio for free if you choose Community Edition (at this link[51]).
- Download the Azure Sphere SDK Preview for Visual Studio, available here[52].

Setting up the device

The first step will be to create to your Azure Sphere an identity on your IoT Hub and link it with an Azure AD user for your Workbench.

Perform the following steps:

1. Open your **Azure administration panel** on your browser, and search for your **IoT Hub**. Once inside it, search for **IoT devices** and click on it.



2. Click on **+ Add**. A tab will open, fill the **Device ID field** with "*AzureSphereDevice*" and click on **Save**.

[47] Azure Sphere MT3620 Development Kit: https://www.seeedstudio.com/Azure-Sphere-MT3620-Development-Kit-EU-Version-p-3134.html

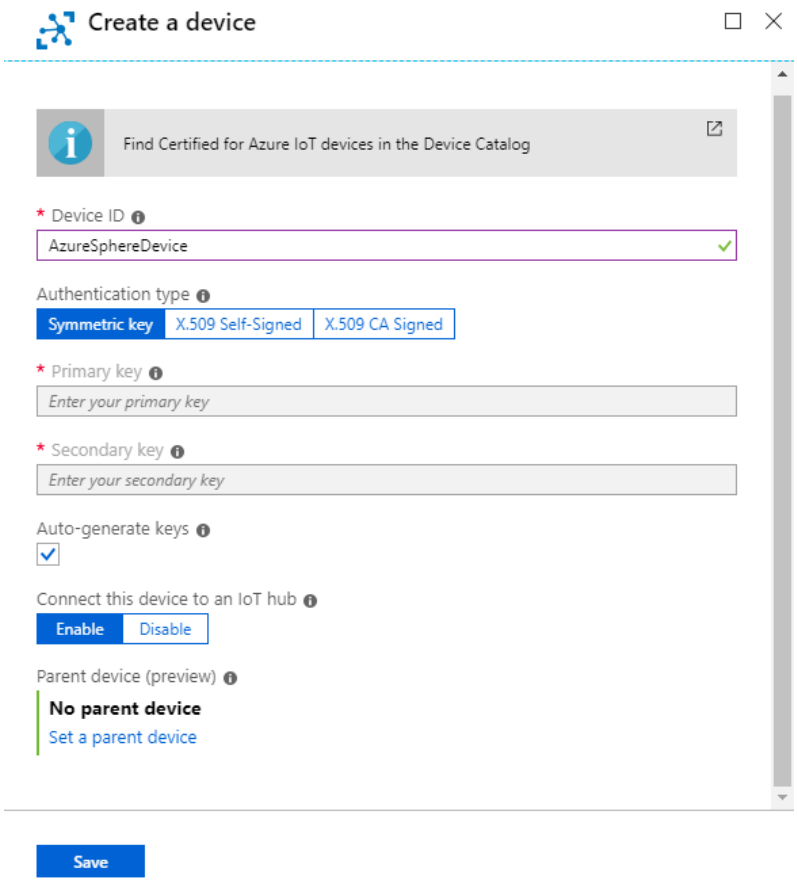
[48] Grove Starter Kit for Azure Sphere MT3620 Development Kit: https://www.seeedstudio.com/MT3620-Grove-Shield-p-3145.html

[49] Grove – Temperature & Humidity Sensor (SHT31): https://store.arduino.cc/grove-temperature-humidity-sensor-sht31

[50] Grove Starter Kit for Azure Sphere MT3620 Development Kit: http://wiki.seeedstudio.com/Grove_Starter_Kit_for_Azure_Sphere_MT3620_Development_Kit/

[51] Visual Studio Community Edition: https://visualstudio.microsoft.com/

[52] Azure Sphere SDK Preview for Visual Studio: https://aka.ms/AzureSphereSDKDownload



Note The Device ID is set to AzureSphereDevice here for convenience because you will find the same ID inside the code of the device, but you can change it if you want. Simply don't forget to also change it inside the code and the SQL request that you will see below.

3. Finally, you need to link your device ID to an existing user account in your Azure AD tenant, see section § CREATING A USER IN YOUR AZURE AD TENANT. You are going to use the user iot-device-3@<your tenant>.onmicrosoft.com that you've created before. Go on **Resource groups** in your Azure administration panel, then the resource group where your Azure Blockchain Workbench is, and search for the database.
4. Click on it, and inside it, click on **Query Editor**.
5. Sign in using your credential defined when you created your Workbench.

   Inside it, run the following command:

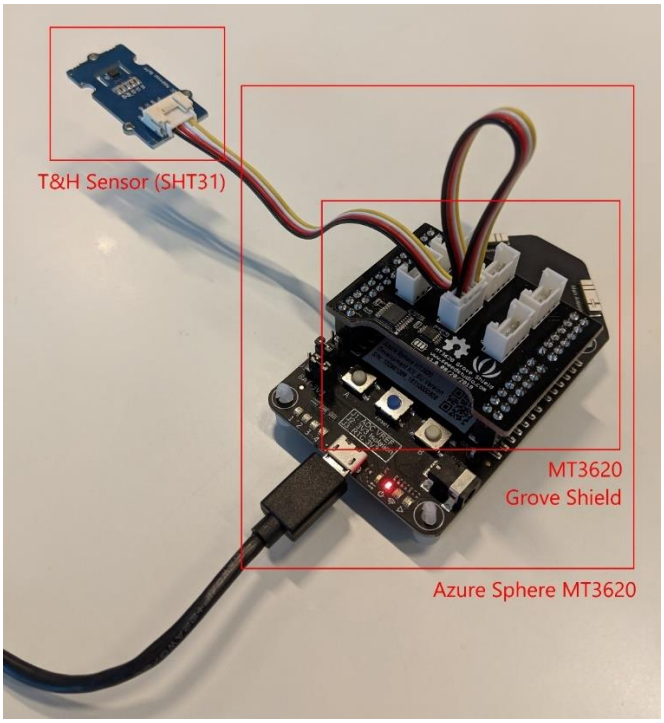
```
Update [User] Set ExternalDeviceId = 'AzureSphereDevice' where EmailAddress = 'iot-device-3@<your tenant>.onmicrosoft.com'
```

6. Check that the command has been correctly executed by looking into the messages tab below the Query Editor. It should be *Query succeeded: Affected rows: 1*.

Now, you're ready to assemble and configure the device remotely, using a terminal.

Perform the following steps:

1. Assemble the device by plugging the shield into the Azure Sphere board and the sensor into an I2C bus plug on the shield. The picture below shows how it should looks like.



2. Search for "*Azure Sphere Developer Command Prompt*" in your computer, and open it. It has been installed with the Azure Sphere SDK.
3. Type the command into the terminal to update the Azure Sphere OS:

```
azsphere device recover
```

The result should be like the screenshot below.

```
Starting device recovery. Please note that this may take up to 10 minutes.
Board found. Sending recovery bootloader.
Erasing flash.
Sending images.
Sending image 1 of 16.
Sending image 2 of 16.
. . .
Sending image 16 of 16.
Finished writing images; rebooting board.
Device ID: <GUID>
Device recovered successfully.
Command completed successfully in 00:02:37.3011134.
```

4. To work, the Azure Sphere needs to be claimed by an Azure AD tenant. Type the following command into the terminal console.

```
azsphere login
```

   You will have to signin with your (Microsoft) account. Do it.

5. You will get a list of available Azure AD tenants (you should already have the default one), choose the one you want to use by typing:

```
azsphere tenant select --tenantid <Tenant ID>
```

6. Type the following command to link your device with the selected Azure AD tenant.

```
azsphere device claim
```

   Be aware, however, that the Azure Sphere Security Service currently enables all members of the organization to manage all devices in an Azure Sphere tenant. If you want greater control over access to your Azure Sphere devices, you or your IT administrator can limit access to your tenant[53].

   WARNING : Claiming is a one-time operation that you cannot undo even if the device is sold or transferred to another person or organization. A device can be claimed only once. Once claimed, the device is permanently associated with the Azure Sphere tenant.

```
Claiming device.
Claiming attached device ID 'ABCDE082513B529C45098884F882B2CA6D832587CAAE1A90B1CEC4A376EA2F22A96C4E7E
Successfully claimed device ID 'ABCDE082513B529C45098884F882B2CA6D832587CAAE1A90B1CEC4A376EA2F22A96C4
Command completed successfully in 00:00:05.5459143.
```

7. To complete this setup, you need to enable Wi-Fi connection on the Azure Sphere. Register the device's MAC address if your network environment requires it. Use the following command to get the MAC address:

```
azsphere device wifi show-status
```

8. Add your Wi-Fi network to the device by using the `azsphere device wifi add` command as follows:

```
azsphere device wifi add --ssid <yourSSID> --key <yourNetworkKey>
```

[53] Limit access to your tenant: https://docs.microsoft.com/en-us/azure-sphere/install/limit-tenant-access

Replace <yourSSID> with the name of your network and <yourNetworkKey> with your WPA/WPA2 key. Azure Sphere devices do not support WEP. Network SSIDs are case-sensitive. For example:

```
azsphere device wifi add --ssid My5GNetwork --key secretnetworkkey
```

To add an open network, omit the `--key` flag.

If your network SSID or key has embedded spaces, enclose the SSID or key in quotation marks. If the SSID or key includes a quotation mark, use a backslash to escape the quotation mark. Backslashes do not require escape if they are part of a value. For example:

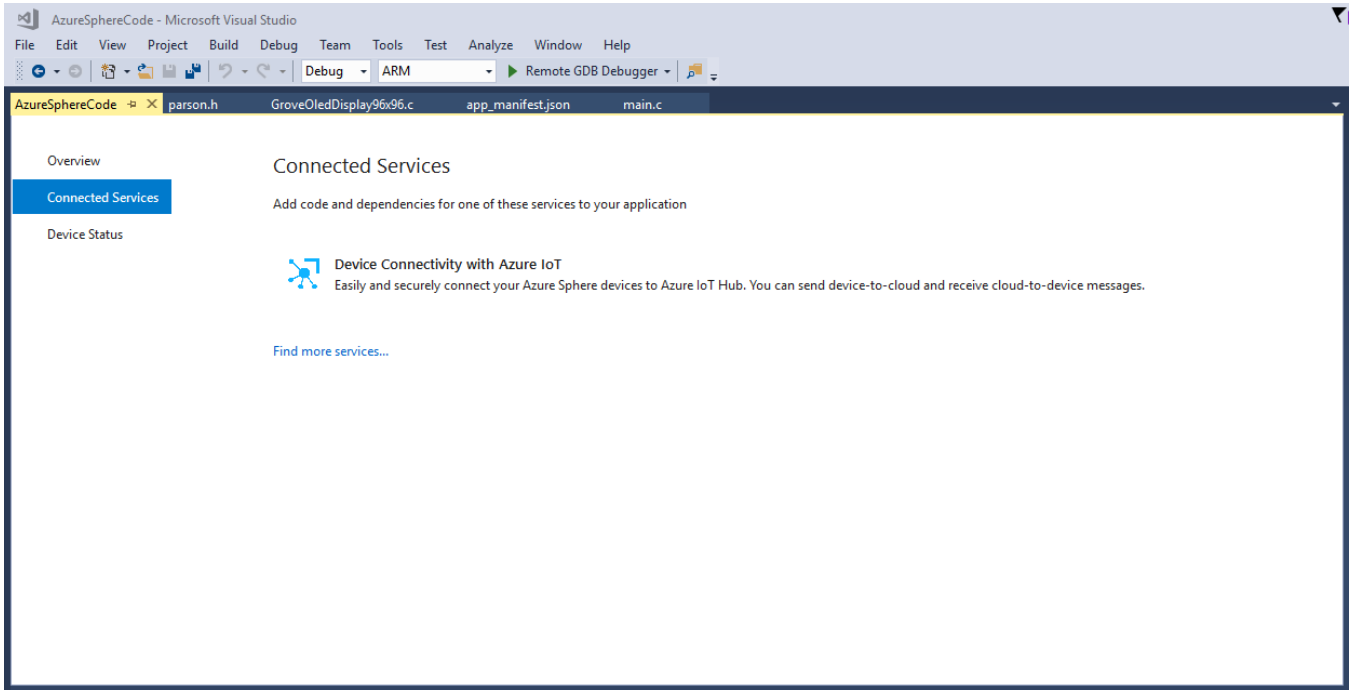
```
azsphere device wifi add --ssid "New SSID" --key "key \"value\" with quotes"
```

It typically takes several seconds for networking to be ready on the board, but might take longer, depending on your network environment.

Setting up the project

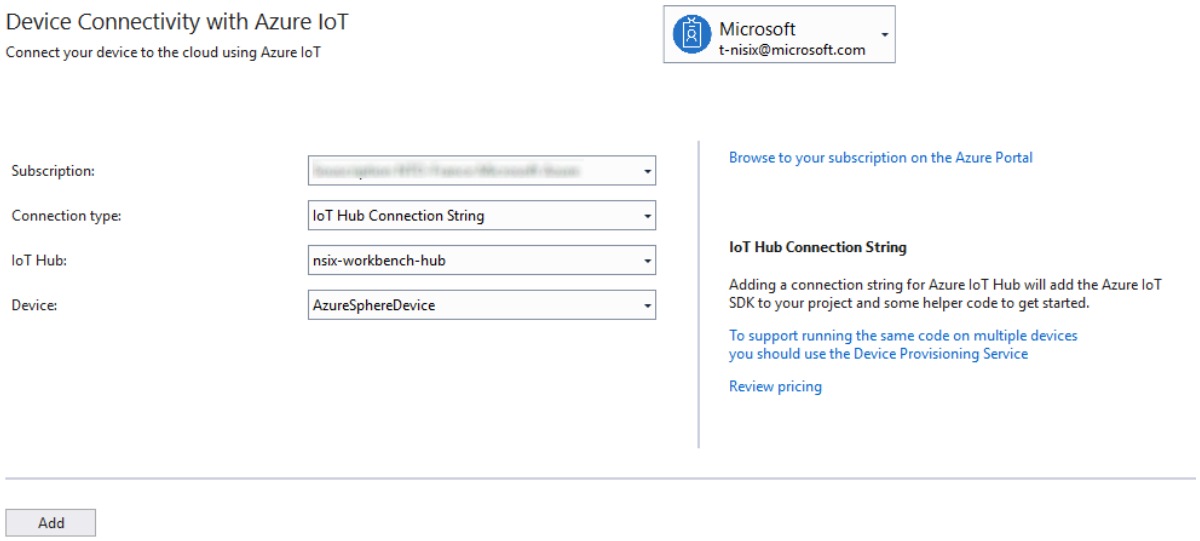
Now that the board is ready, your next steps will consist in opening the Visual Studio project, compiling the project code and sending it to your Azure Sphere device.

1. In Visual Studio Code, open the *AzureSphereCode.sln* which is inside *refrigerated-sc-sample* > *IoT setup* > *azure-sphere-device* directory.
2. Inside your IDE, in the **Solution Explorer** tab, in the *AzureSphereCode* project right-click on **References**, then on **Add Connected Service ...**. You should arrive on this screen.



3. Click on **Device Connectivity with Azure IoT**. Choose (or add) your (Microsoft) Account, then fill the form, and then click on **Add**.

Setting	Description
Subscription	Select your Azure Subscription.
Connection type	Select **IoT Hub Connection String**.
IoT Hub	Select your IoT Hub.
Device	Select the device ID of your Azure Sphere.



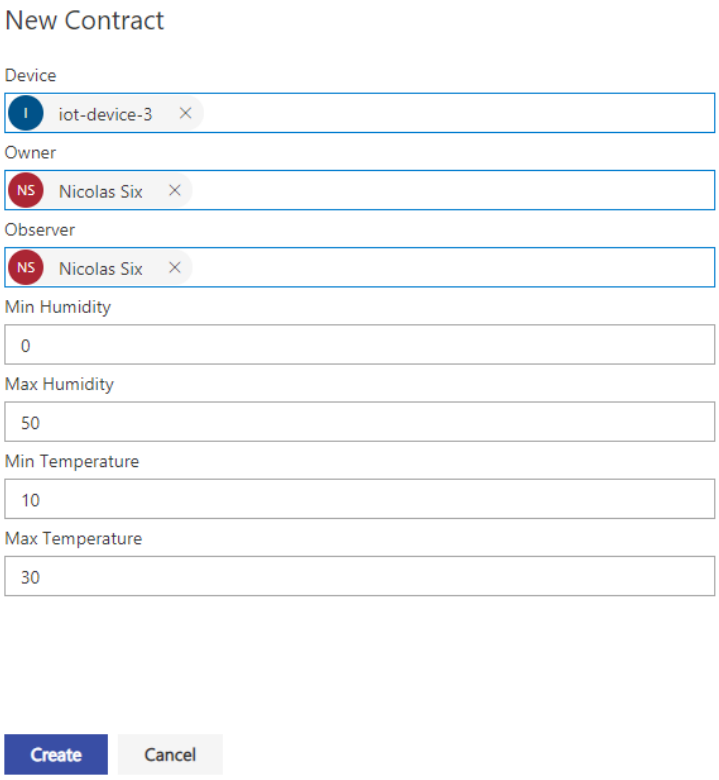
4. Right-click on **References** again, and click on **Add Reference ...**. Check the box linked to *MT3620_Grove_Shield_Library* and click on **OK**.

Testing the device

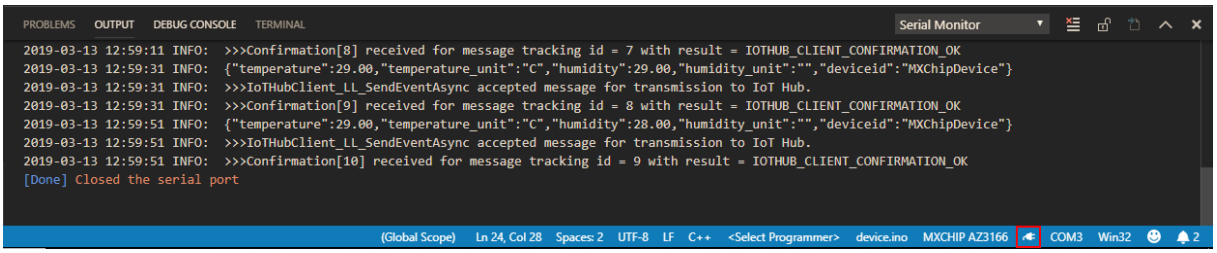
Perform the following steps:

1. Go inside your Workbench application on your browser and click on your deployed **Refrigerated Transportation** application.
2. On the main application tab, click on **+ New** to create a new contract. Fill all the persona fields with your identity except for the device that you will fill with your IoT Device. Set the temperature between 10 and 30 and the humidity between 0 and 50 for testing. Finally, click on **Create** to deploy the Contract.

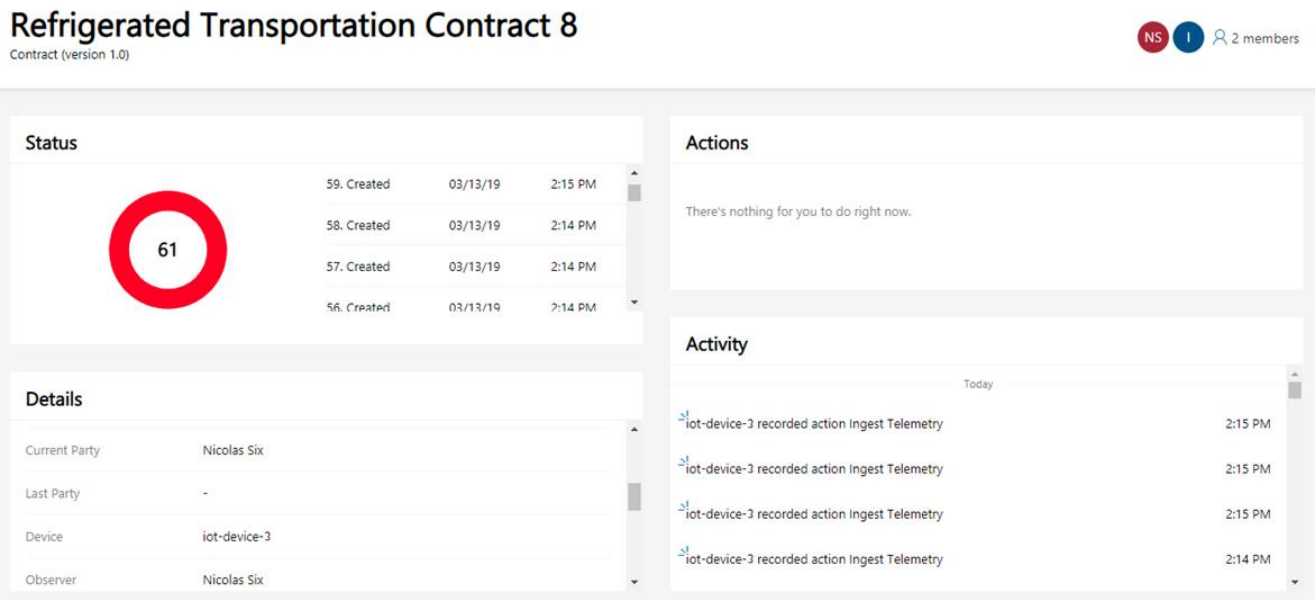
Note Of course, those values aren't realistic for refrigerated transportation, but one should guess that you will test your device to room temperature!.



3. Once the contract is deployed, you are ready to start your device. If you let it plugged, the device is already sending telemetry to the Workbench, using the same routing method as your simulated device. You can see sent messages into the Visual Studio Code console, by clicking on the **plug icon** at the bottom of the screen.



4. Check in your Workbench contract if everything works fine. The IoT Device should appear in it, invoking **Ingest Telemetry** method. If you heat the temperature sensor on your device up to 30 degrees Celsius, you will see that the contract state will automatically change to *"Out-of-compliance"*.



The Azure Sphere is now up and running, and as such, it greatly illustrate what could be achived with a sensor for supply chain monitoring.

Indeed, the sensor is sending values every 20 seconds, and everything is recorded inside the blockchain. If the temperature or the humidity reaches an "out-of-compliance" value, the contract state changes automatically and the participants inside the supply chain will know which counterparty is responsible for the perished goods.

Using a Raspberry Pi device (optional)

Overview and prerequisites

The last device which will be introduced here is one of the most known device in the IoT world and beyond, i.e. the Raspberry Pi.

This device is different from the 2 previous devices introduced here. While both the Azure Sphere and the MXChip IoT DevKit are microcontrollers which can only run one program at a time, the Raspberry Pi is a fully-plegded working microcomputer.

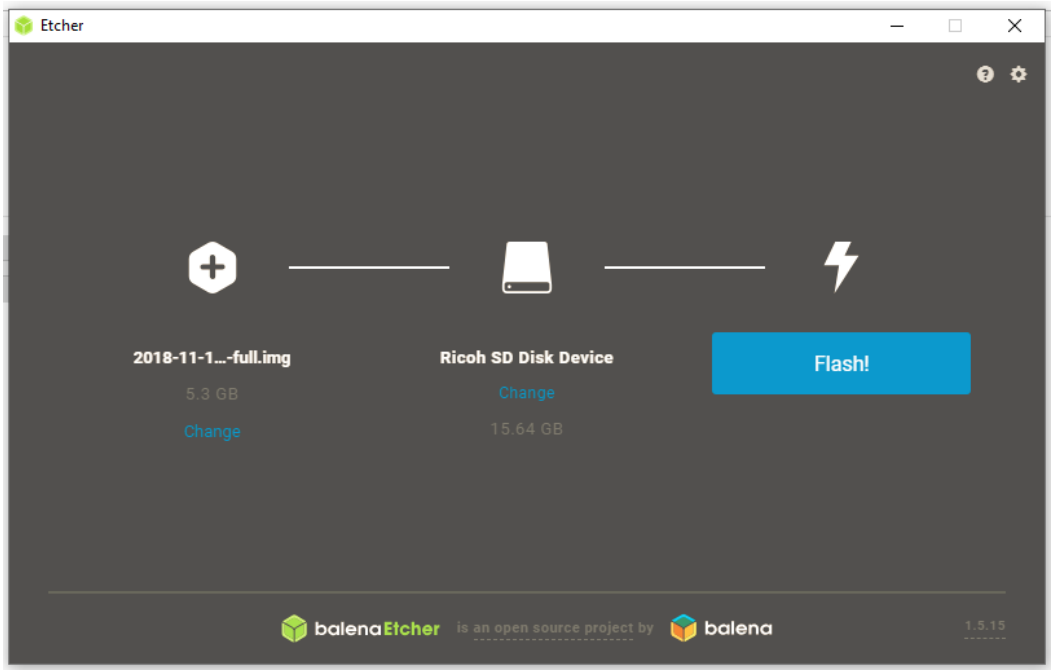
For the sake of this guide, you will be using the Raspberry Pi 2B, with an OS named Raspbian. This is a Debian-based computer OS specially optimized for the Raspberry Pi. Although the walkthrough covers this version of Raspberry Pi here, you're free to use any other Raspberry Pi to follow this guide.

First, you have to create a bootable Micro SD card for your Raspberry Pi. If you didn't have downloaded it yet, go to the Raspbian website[54] and download the disk image of Raspbian. You will also need a flash software to write the disk image to your Micro SD card. For that purpose, you're going to use Etcher, available here[55].

[54] Raspbian: https://www.raspberrypi.org/downloads/raspbian/

[55] balenaEchter: https://www.balena.io/etcher/

Insert an empty Micro SD card into your computer, open the software, click on **Select image** and choose the downloaded Raspbian disk image, then click on **Select drive** / **Change** (if the wrong device is selected by default, for example a USB stick instead of your SD Card), then click on **Flash**.



Wait for the program to complete flashing, and remove your Micro SD card from your computer when everything is done.

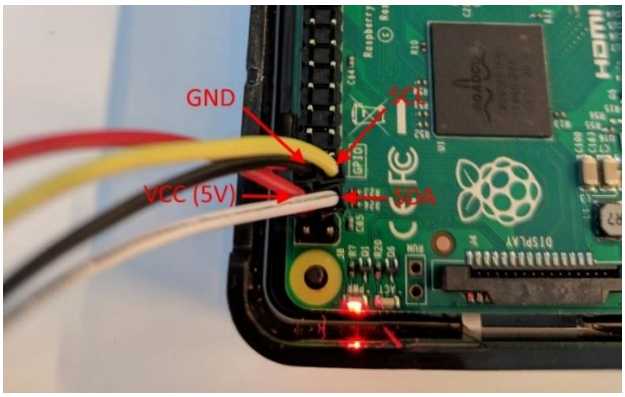
Before launching the Raspberry Pi for the first time, you need to plug the sensor to the card. This guide opts for the Grove SHT31 Temperature & Humidity sensor, but you may use the Adafruit BME280 sensor instead.

> **Note** For more information, see article CONNECT RASPBERRY PI TO AZURE IOT HUB (NODE.JS)[56].

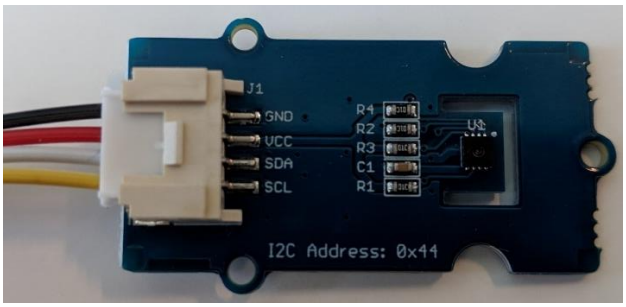
> **Note** A lot of sensors can be used with the Raspberry Pi, so if you are comfortable with sensors and Node.js, feel free to modify the Raspberry Pi code that you will deploy on your device later, as the solution introduced in this guide is only one of the many possible solutions. You could have also leveraged the Windows 10 IoT Core[57] as the Raspberry OS for example.

The SHT31 sensor can be plug on a board compatible with I2C, such as the Raspberry Pi. To connect the sensor to the board, plug the four male-to-female cables to the four pins of the SHT31 sensor, then plug the other side of the cables to the board as illustrated in the picture below.

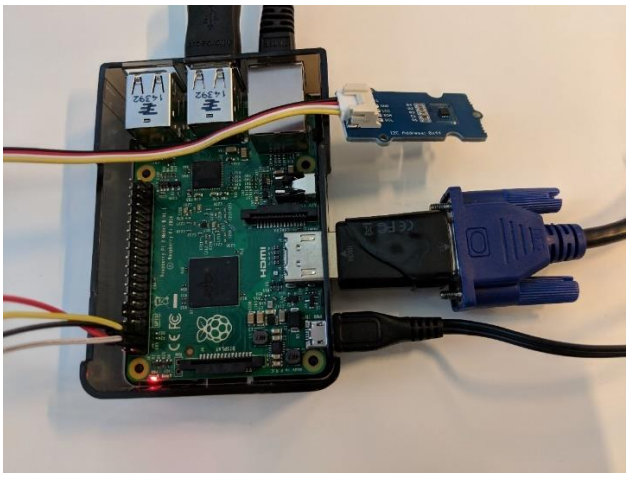
[56] CONNECT RASPBERRY PI TO AZURE IOT HUB (NODE.JS): https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-raspberry-pi-kit-node-get-started
[57] HOW TO INSTALL WINDOWS 10 IOT CORE ON RASPBERRY PI 3: https://www.windowscentral.com/how-install-windows-10-iot-raspberry-pi-3



If you need, you can see directly on the sensor which cables correspond to which plug on the board:



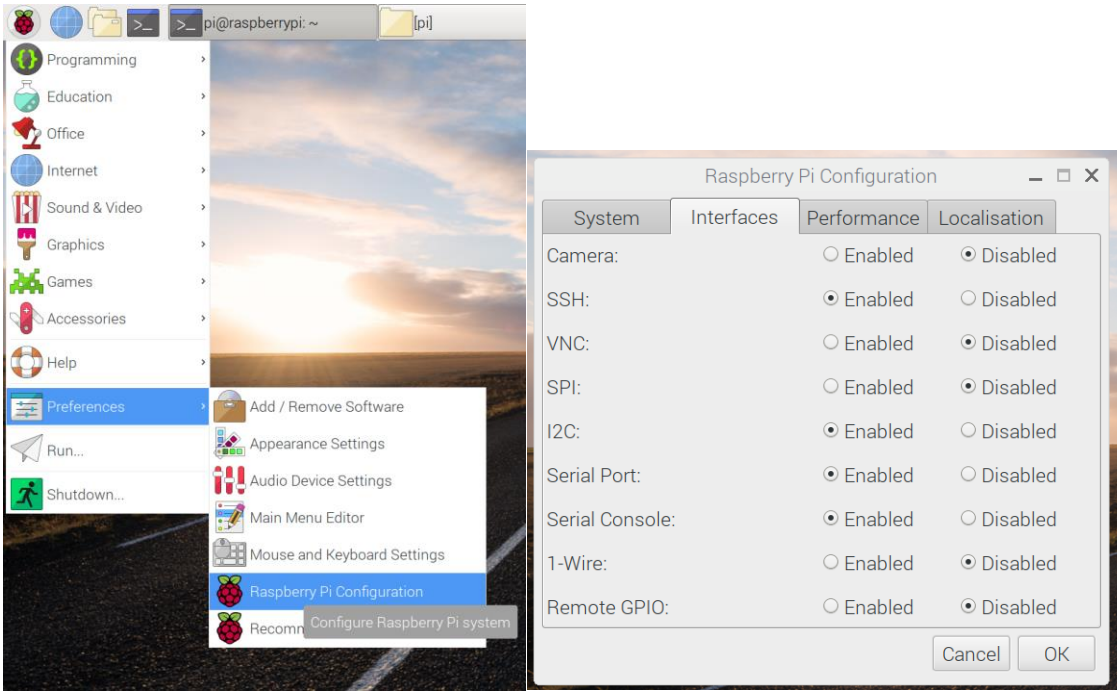
You're nearly ready to boot up the Raspberry Pi. Insert the Micro SD card into the SD slot under the card, your computer peripherics (keyboard, mouse, ethernet cable/wifi dongle and screen), and power on your Raspberry Pi with a Micro USB power source. You should see your Raspberry turn on by looking on the screen: wait for the Raspberry Pi to finish and you're ready to configure it in order to run your solution.



Setting up the device

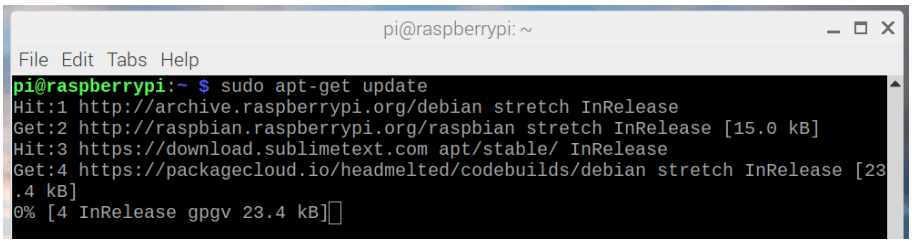
Now, you can start configuring the Raspberry Pi. Perform the following steps:

1. Click on the raspberry at the top of the screen, then **Preferences** > **Raspberry Pi Configuration**.
2. Go in the **Interfaces** tab, and enable the I2C communication by clicking on **Enabled** radio button.



3. Open a terminal console, and type the following command to update your system.

```
sudo apt-get update
```

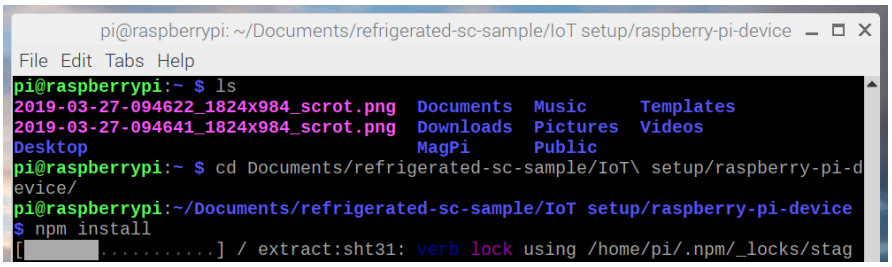


4. Go with the terminal inside the refrigerated-sc-sample code file (which can be downloaded or cloned from here[58]), then *IoT Devices* > *raspberry-pi-device*.

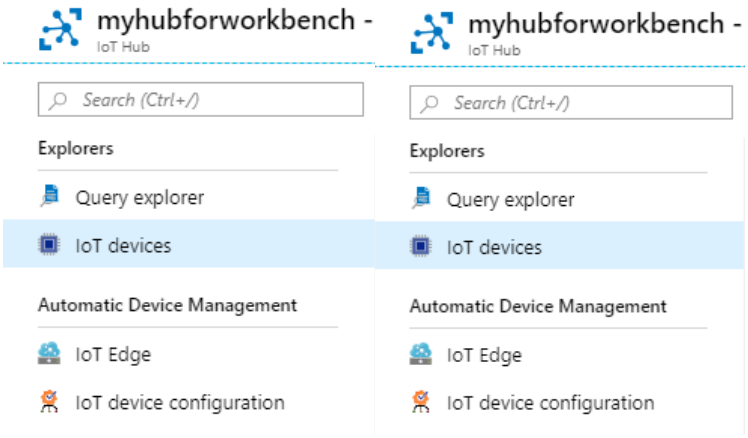
5. To install required packages for your application, type the following command:

```
npm install
```

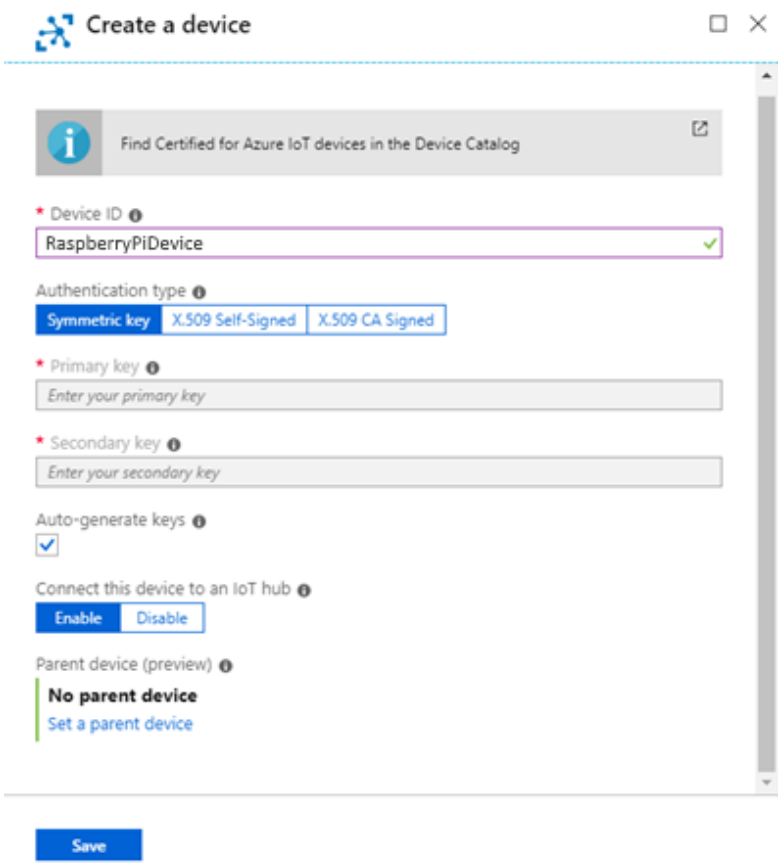
[58] Refrigerated Supply Chain samples: https://aka.ms/ABWDevGuideSamples



6. To test the program, you need to provide a connection string which can be get from the IoT Hub. Thus, you will create an identity for your Raspberry Pi in the devices registry of your IoT Hub. Open your Azure administration panel on your browser, and search for your IoT Hub. Once inside it, search for "*IoT devices*" and click on it.



7. Click on **+ Add**. A tab will open, fill the **Device ID** field with "*RaspberryPiDevice*", and click on **Save**.



Note The Device ID is set to "RaspberryPiDevice" here for convenience because you will find the same ID inside the code of the device, but you can change it if you want. Simply don't forget to also change it inside the code and the SQL request that you will see below.

8. You've been redirected to your IoT devices list, click on your newly created device and inside the details, and save the **Connection string (primary key)**. You will need it to enable the connection between your Raspberry Pi and your IoT Hub.
9. Finally, you need to link your device ID to an existing user account in your Azure AD tenant, see section § CREATING A USER IN YOUR AZURE AD TENANT. You are going to use the user iot-device-4@<your tenant>.onmicrosoft.com that you've created before. Go on **Resource groups** in your Azure administration panel, then the resource group where your Azure Blockchain Workbench is, and search for the database.
10. Click on it, and inside it, click on **Query Editor**.
11. Sign in using your credential defined when you created your Workbench.

Inside it, run the following command:

```
Update [User] Set ExternalDeviceId = 'RaspberryPiDevice' where EmailAddress = 'iot-device-4@<your tenant>.onmicrosoft.com'
```

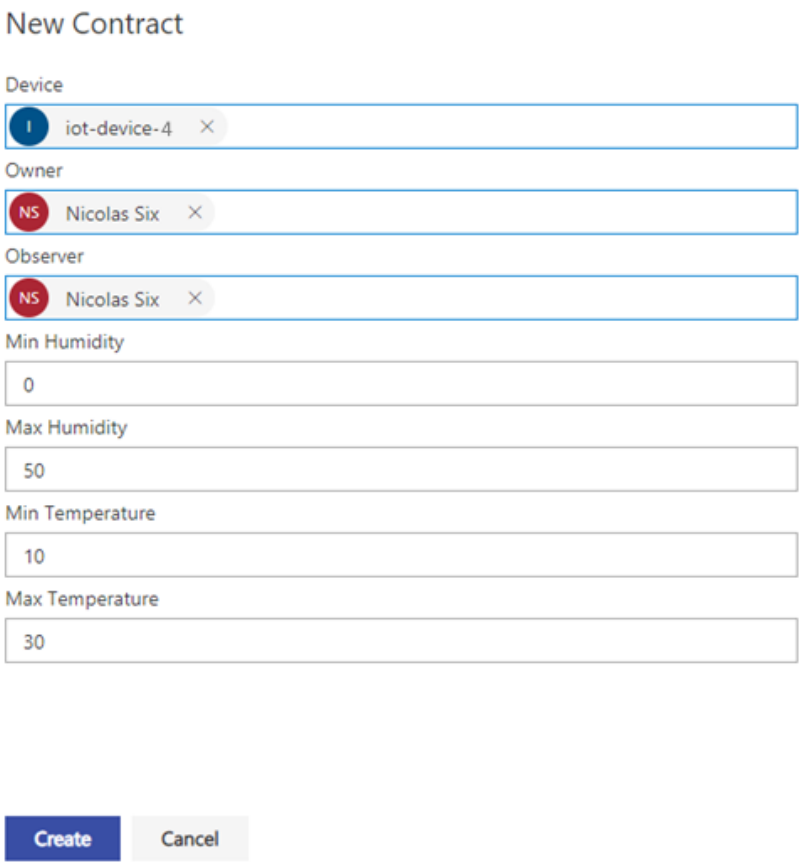
12. Check that the command has been correctly executed by looking into the messages tab below the Query Editor. It should be *Query succeeded: Affected rows: 1.*

Testing the device

Perform the following steps:

1. Go inside your Workbench application on your browser and click on your deployed **Refrigerated Transportation** application.
2. On the main application tab, click on **+ New** to create a new contract. Fill all the persona fields with your identity except for the device that you will fill with your IoT Device. Set the temperature between 10 and 30 and the humidity between 0 and 50 for testing. Finally, click on **Create** to deploy the Contract.

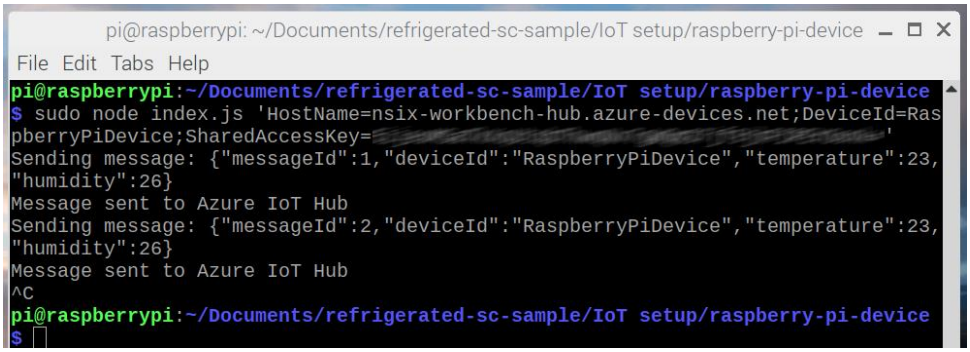
Note Of course, those values aren't realistic for refrigerated transportation, but I guess that you will test your device to room temperature!.



3. To test the device, type in your Raspberry Pi terminal the following command:

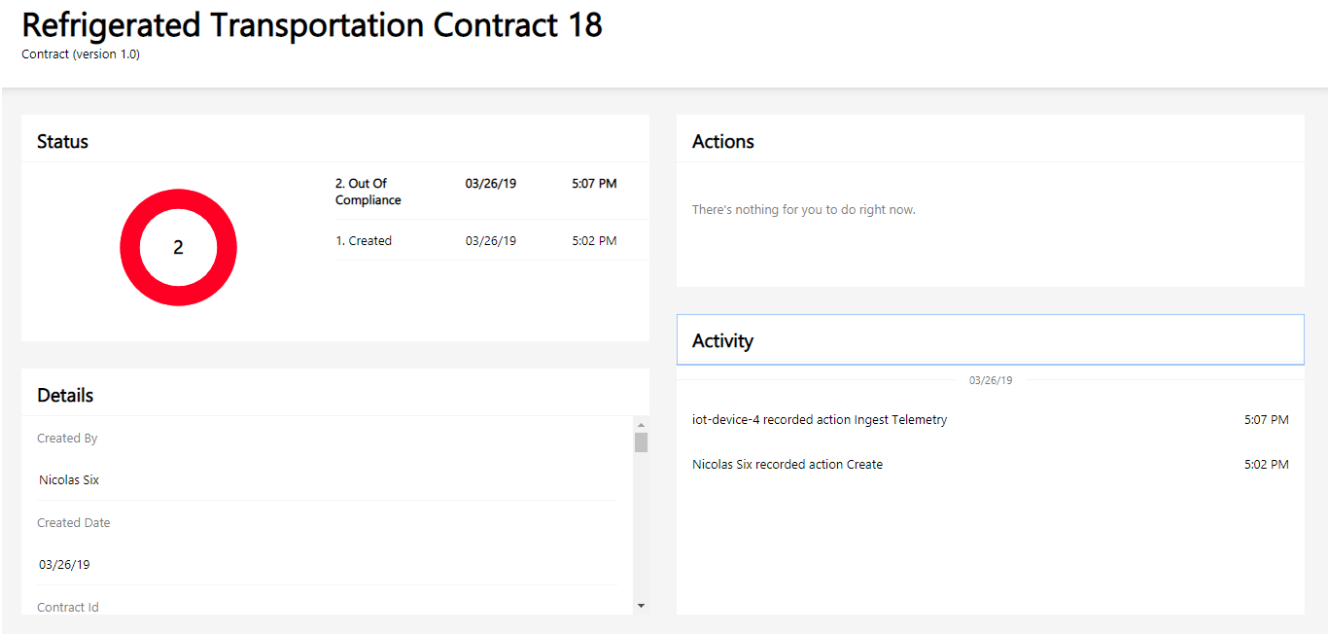
```
sudo node.js index.js '<Your connection string>'
```

by replacing the connection string with your own that you got from the Hub. If the connection is established, the program will send telemetry to your Hub and you will see the flow of sent packets, with the current temperature and humidity.



4. Check in your Workbench Contract if everything works, the IoT Device should appear in it, invoking **Ingest Telemetry** method.

If you heat the Temperature sensor on your device up to 30 degrees Celsius, you will see that the contract state will automatically change to *"Out-of-compliance"*.



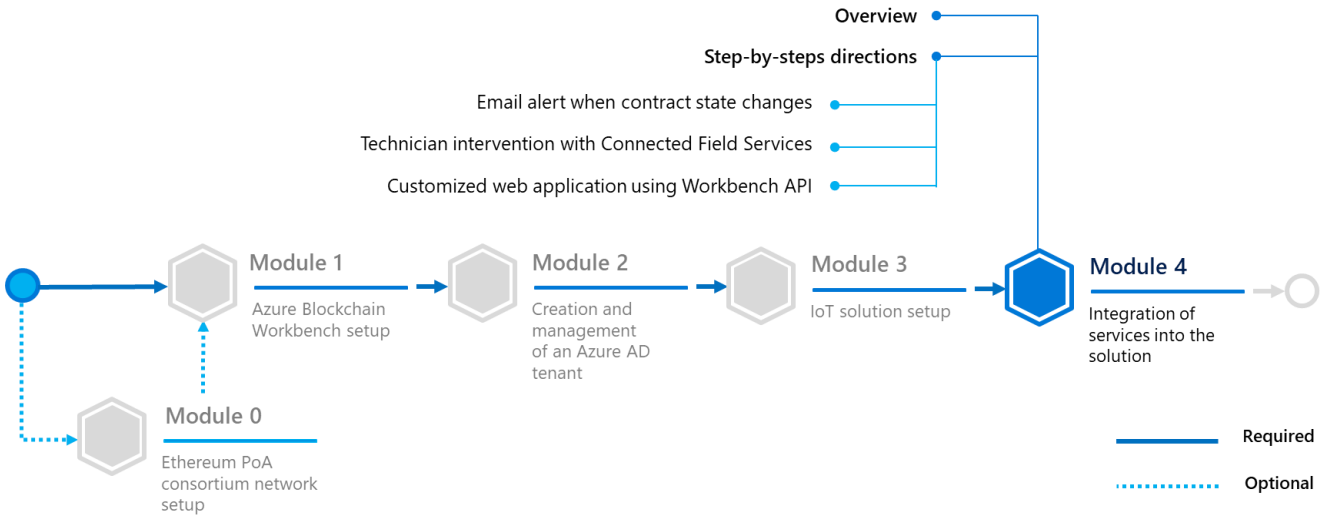
Module 4: Integration of services into the solution

Overview

Since the beginning and throughout the previous modules, you've built an up and running solution, with an fully functional blockchain and an additional layer on top of it which is the Workbench.

You've also connected multiple IoT devices, which demonstrates that any connected device can be used as a temperature sensor provided that it can establish a connection with your IoT Hub, and you also saw how to manage Workbench users and user groups.

But now, the objective will be to discover how Azure services can infuse your solution, adding a large panel of possibilities.



In this module, you will learn to:

- Connect an external web application to the Workbench using the API. That is how the Workbench can be integrated into any application.
- Trigger a technician intervention using Connected Field Services[59], a Dynamics 365 module.

[59] Connected Field Services – Overview: https://docs.microsoft.com/en-us/dynamics365/customer-engagement/field-service/connected-field-service

Step-by-step directions

This module covers the following activities:

1. Sending an alert by email when contract state changes.
2. Initiating a rechnician intervention with Connected Field Services.
3. Adding IoT devices to your solutionBuidlng a React Web application.

Each activity is described in order in the next sections.

Sending an alert by email when contract state changes

A good idea would be to alert counterparties of the contract whenever the state changes. For example, the Observer member of the contract should get informed if a shipment is marked as “out-of-compliance” to take action. Or, the Owner of the shipment should get a notification if his shipment is arrived.

Probably the best way to do so is to create a new Logic App. It will take Workbench messages as an input from its Egress Topic, and check if a state changes and if yes, which one. Then, the app will send an email to alert if something important happened.

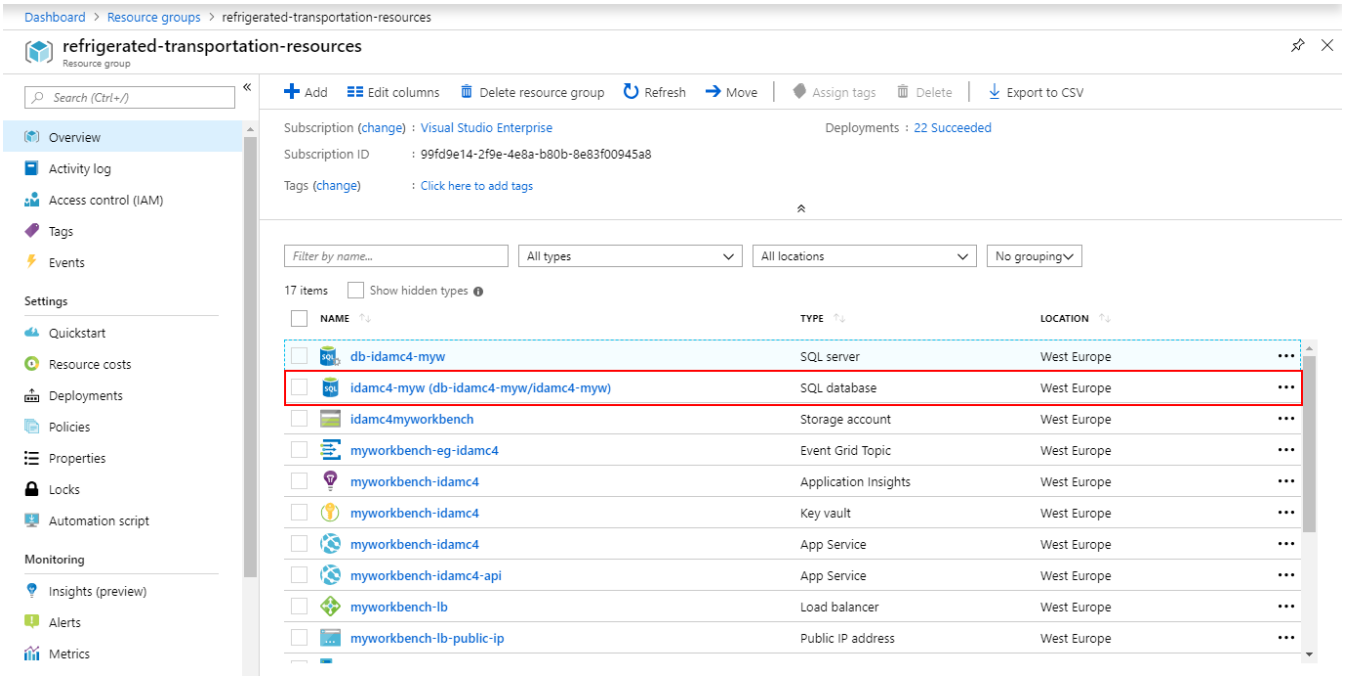
Also, sending an email to contract parties when the counterparty in charge of the shipment changes could be a great fit for the sake of this guide. Good news! You’re going to implement both of those alerts inside your Logic App.

Before doing that, you will need to create a request which is able to retrieve all contact parties emails.

Getting contract parties emails

Perform the following steps:

1. Go on **Resource groups**, then the resource group where your Azure Blockchain Workbench is, and search for the database.



2. Click on it, and inside it, click on **Query Editor**.
3. Sign in using your credential defined when you created your Workbench.
4. Open the *refrigerated-sc-sample* which is the code for this guide that you downloaded or cloned before (or grab it here[60] if you don't have done it yet), then open the *getContractParties.sql* file which is inside the directory *Integration*, paste its content into the editor and click on **Run**.
5. Verify the success of the request by checking the line below the editor. You should see *Query succeeded: Affected rows: 0*.

Setting up the Logic App

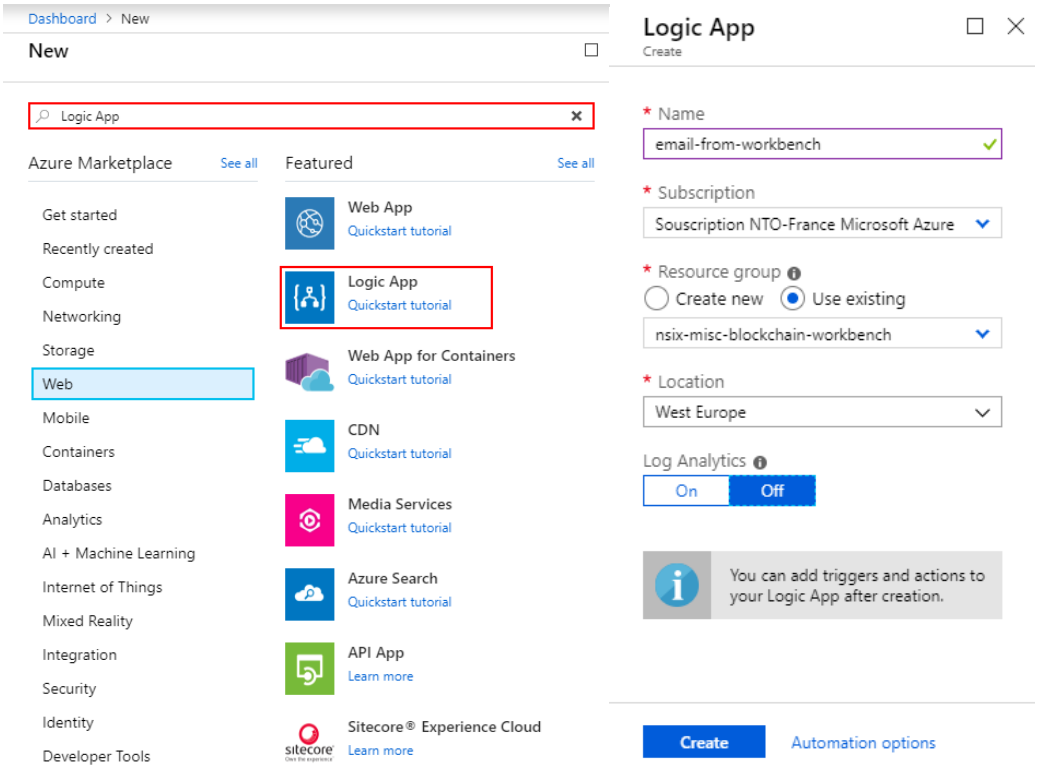
> **Note** When creating a Logic App, you have the possibility to rename each step of the App if you want. This is useful if you want to give a more explicit name, for example name a step "Get contract parties" instead of the default "Execute a stored procedure".
>
> If you want to do so, keep in mind that you should do it when creating the step, and not at the end of the Logic App creation, because it's impossible to rename a step if some data from the step is used somewhere else. In this guide, you should keep default step names.

Perform the following steps:

1. Sign in to the Azure portal.
2. Choose +**Create a resource**, then choose **Web**.
3. Click **Logic App** from the list on the right.

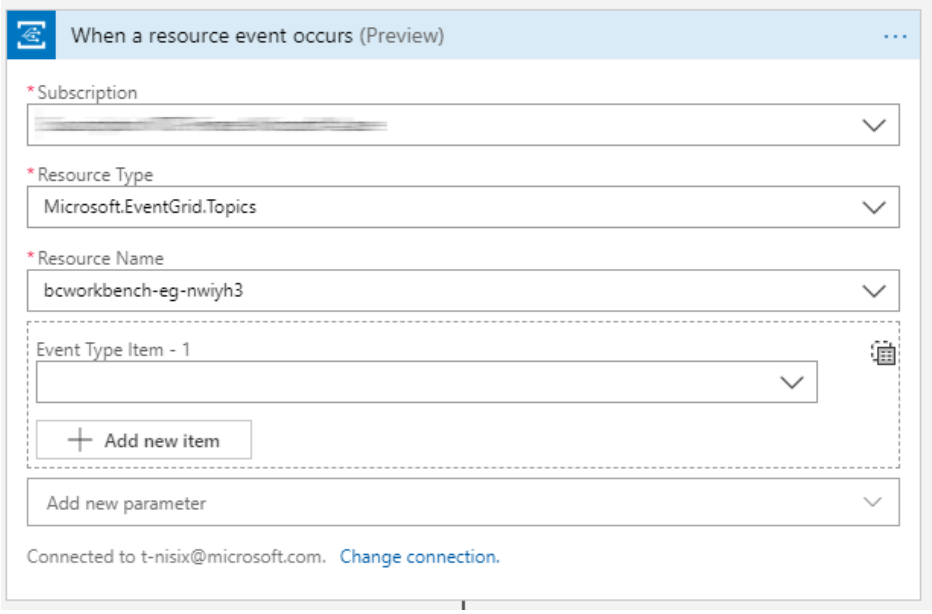
[60] Refrigerated Supply Chain samples: https://aka.ms/ABWDevGuideSamples

4. Fill the configuration tab.

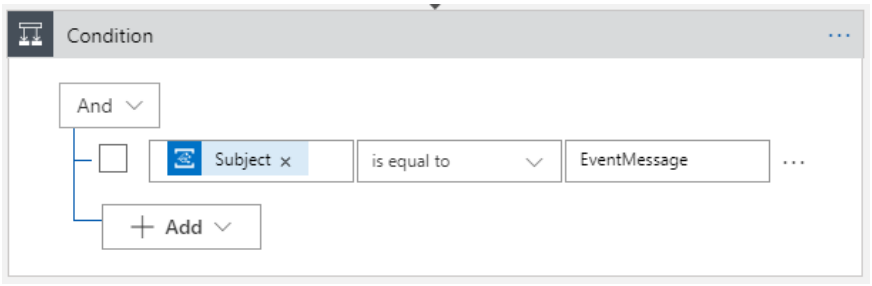


Setting	Description
Name	This is the name of your future app.
Subscription	Select the subscription to use for your Logic App.
Resource group	You can create a new resource group or use an existing one. To create a new one, click **Create** new and fill in the name you want to use. To use an existing resource group, click **Use existing** and select the resource group from the dropdown list.
Location	This is the region in which you want your Logic App to be located. Select the location closest to you from the dropdown list.
Log Analytics	Keep the **Off** parameter for diagnostic logging.

5. After the creation of the Logic App, go inside the **Logic App Designer**. Click on **Service Bus** and search for **When a resource event occurs,** then click on it.
6. Fill in the form by selecting your subscription, the resource type named **Microsoft.EventGrid.Topics**, and your Workbench egress queue which should be named *<workbench-name>-eg-<workbench-id>*.



7. Click on **+New Step**. In the **Action** tab, select **Control**, then **Condition**.
8. Click on the left value field, and select **Subject** from the egress queue, and in the right field type **EventMessage**. It means that you're going to check every message from the Workbench and if the subject is EventMessage, something happened inside the Workbench and you need to check if it requires an email.

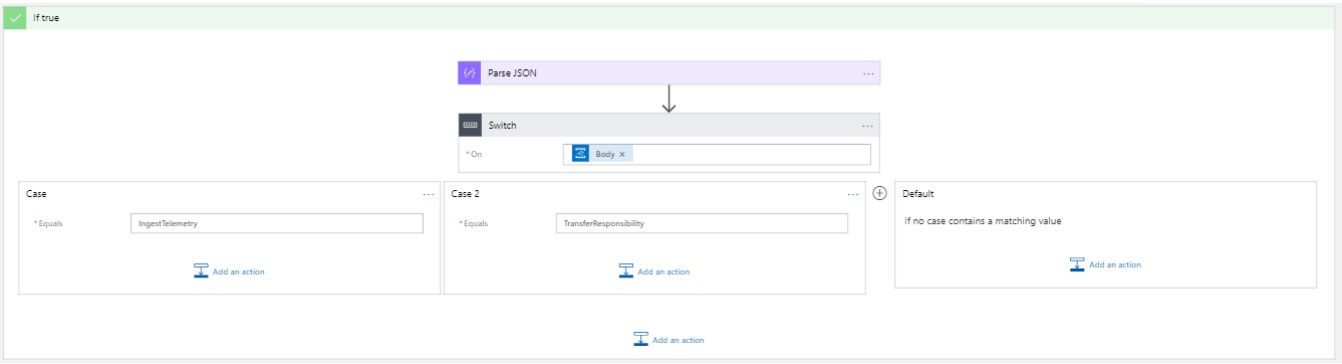


9. Two tabs should have appeared, True and False. Let the False one by default, as you're not going to take an action for messages which are not Event Messages. In the True tab, click on **Add an Action**. Click on **Data Operations** then **Parse JSON**. Click on the field **Content** and select **Body** from the egress queue, and paste the following code snippet into the schema.

```
{
    "properties": {
        "data": {
            "properties": {
                "AdditionalInformation": {},
                "Caller": {},
                "ConnectionId": {
                    "type": "integer"
                },
                "ContractId": {
                    "type": "integer"
                },
                "ContractLedgerIdentifier": {
                    "type": "string"
                },
                "EventName": {
                    "type": "string"
                },
                "FunctionName": {
                    "type": "string"
                },
                "InTransactionSequenceNumber": {
                    "type": "integer"
                },
                "MessageName": {
                    "type": "string"
                },
                "MessageSchemaVersion": {
                    "type": "string"
                },
                "Parameters": {
                    "items": {
                        "properties": {
                            "Name": {
                                "type": "string"
                            },
                            "Value": {
                                "type": "string"
                            }
                        },
                        "required": [
                            "Name",
                            "Value"
                        ],
                        "type": "object"
                    },
                    "type": "array"
                },
                "Transaction": {
                    "properties": {
                        "From": {
                            "type": "string"
                        },
                        "ProvisioningStatus": {
                            "type": "integer"
                        },
                        "To": {
                            "items": {},
                            "type": "array"
                        },
```

```
                        "TransactionHash": {
                            "type": "string"
                        },
                        "TransactionId": {
                            "type": "integer"
                        }
                    },
                    "type": "object"
                }
            },
            "type": "object"
        },
        "dataVersion": {
            "type": "string"
        },
        "eventTime": {
            "type": "string"
        },
        "eventType": {
            "type": "string"
        },
        "id": {
            "type": "string"
        },
        "metadataVersion": {
            "type": "string"
        },
        "subject": {
            "type": "string"
        },
        "topic": {
            "type": "string"
        }
    },
    "type": "object"
}
```

10. Click on **Add an Action**, then **Control** and **Switch**. In the **On value** field, click and select **Body** from the *egress queue*. Create a second case by clicking the **rounded + button**, and in the two equal fields type respectively "*IngestTelemetry*" and "*TransferResponsibility*".



Those two cases represent the two possible paths that your message will take, the first one if a IoT device send a message to the Workbench (which can possibly lead to a "Out-of-compliance" status change), the second one if a transfer of responsibility between two counterparties occurs.

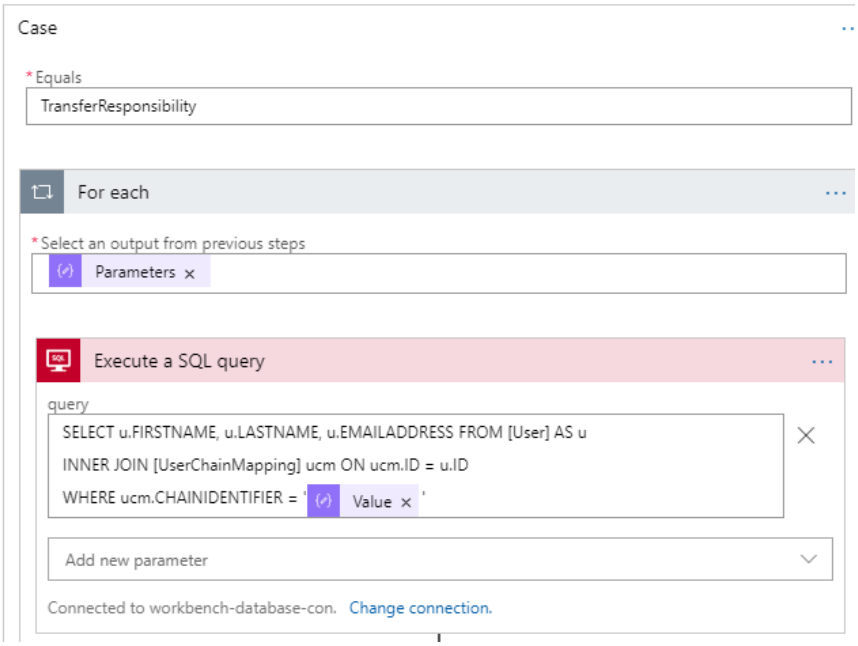
You will complete both of the cases, but for now you will start with the *TransferResponsibility* case.

11. In the TransferResponsibility case, click on **Add an Action**, then **Control** and **For each**. Click in the value field and choose **Parameters** from the previous **Parse JSON** operation.

12. Inside the `For each`, click on **Add an action**, choose **SQL Server** then **Execute a SQL query**. Add a new parameter **query** and inside the new query field, type this request:

```
SELECT u.FIRSTNAME, u.LASTNAME, u.EMAILADDRESS FROM [User] AS u INNER JOIN [UserChainMapping] ucm ON
ucm.ID = u.ID WHERE ucm.CHAINIDENTIFIER = ‘’;
```

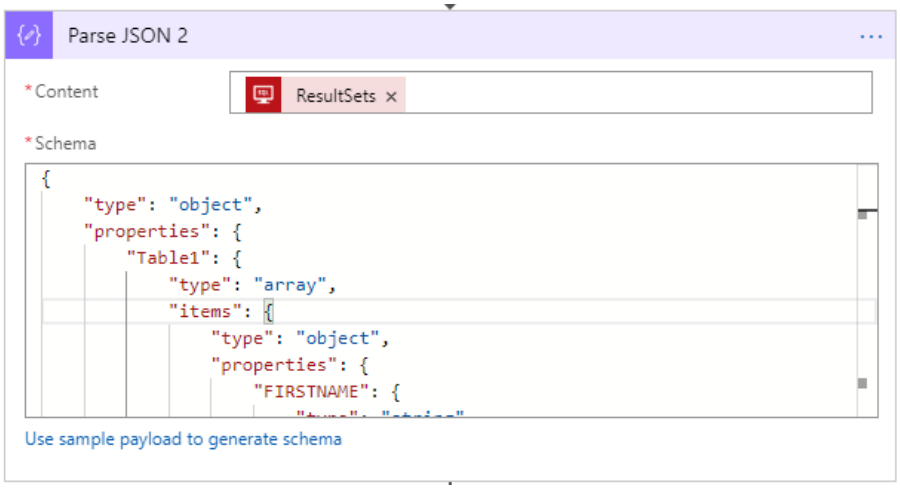
Click between empty semicolons inside the request next to the `CHAINIDENTIFIER`, and in the **Dynamic content** tab, click on **Value** in so far as you want to inject as a parameter the address of the new counterparty inside the query to get his information.



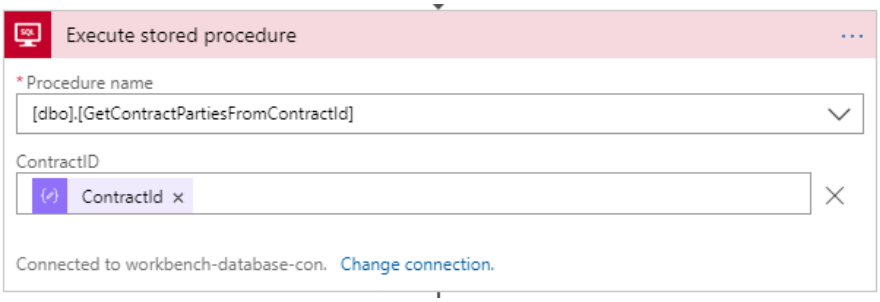
13. Click on **Add an Action**. Click on **Data Operation** then **Parse JSON**. Click on the **Content** field and select **ResultSets**. Paste the following content into Schema:

```
{
    "type": "object",
    "properties": {
        "Table1": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "FIRSTNAME": {
                        "type": "string"
                    },
                    "LASTNAME": {
                        "type": "string"
                    },
                    "EMAILADDRESS": {
```

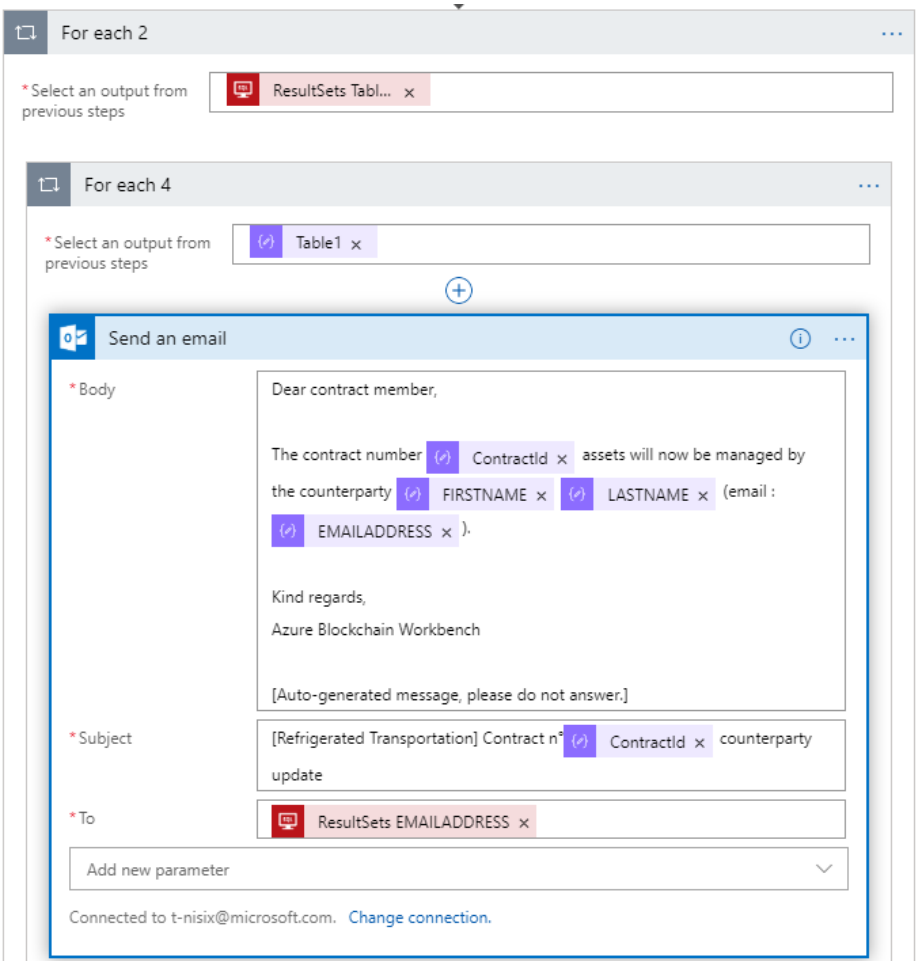
```
                    "type": "string"
                }
            },
            "required": [
                "FIRSTNAME",
                "LASTNAME",
                "EMAILADDRESS"
            ]
        }
    }
}
```



14. Click on **Add an action**. Choose **SQL Server** then **Execute stored procedure**. Select the procedure **[dbo].[GetContractPartiesFromContractId]** previously created, then add a parameter named **ContractID**, click on the field and select the **ContractID** dynamic value.



15. For the final step of this first case, click on **Add an action**, click on **Office 365 Outlook** then **Send an email**. Fill the form as the screenshot below, and keep in mind that adding the dynamic values will automatically create two `For each`, one which will go through the contract parties, and the other will go through the SQL request which returns the new counterparty information.

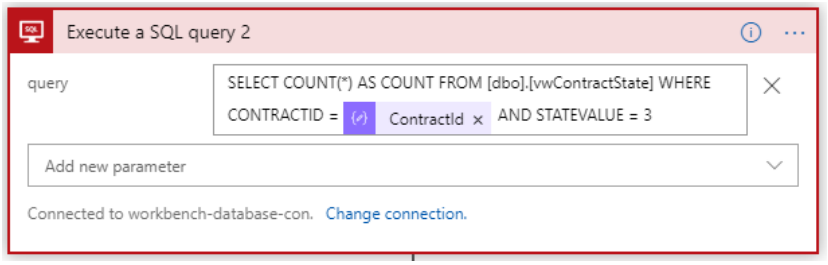


16. Go back to the **IngestTelemetry** case. Click on **Add an action**, choose **SQL Server** then **Execute a SQL query**. Add a new parameter **query** and inside the new query field, type this request:

```
SELECT COUNT(*) AS COUNT FROM [dbo].[vwContractState] WHERE CONTRACTID = <> AND STATEVALUE = 3;
```

Replace the `<>` by the dynamic value `ContractId` as you want to check if a contract defined by his ContractId is "Out-of-Compliance". The table `[dbo].[vwContractState]` returns every state change of the contract, so if an "Out-of-Compliance" state is found (represented by the `STATEVALUE 3`), you're sure that the contract is "Out-of-Compliance" as his state can't change anymore.

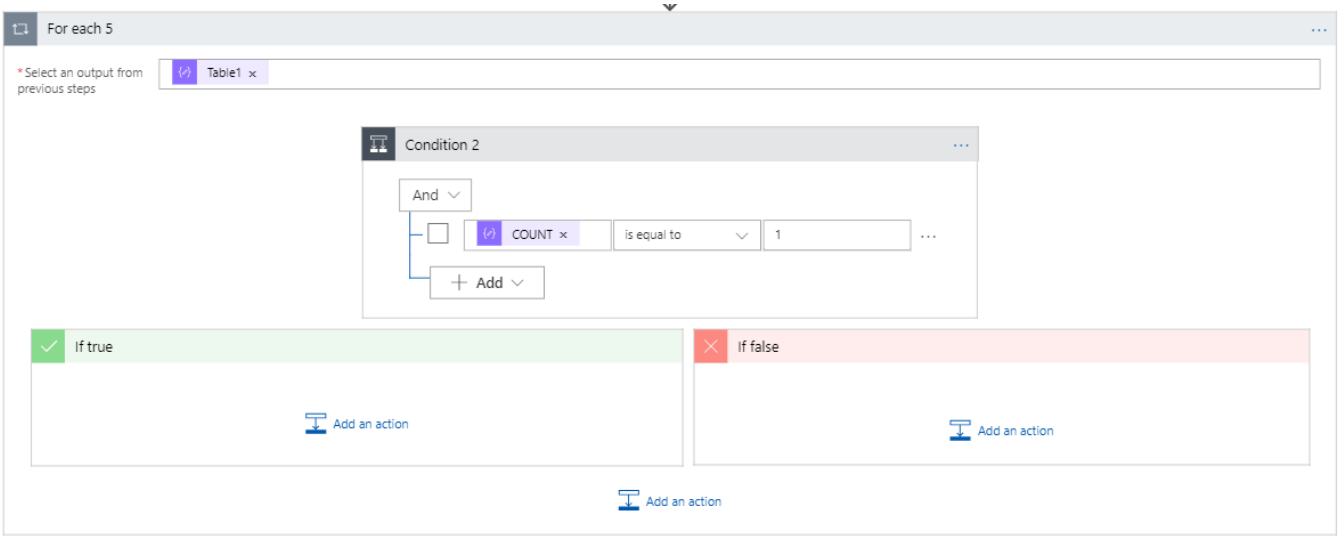
So, the request returns 0 (if the contract is not "Out-of-Compliance" or 1.



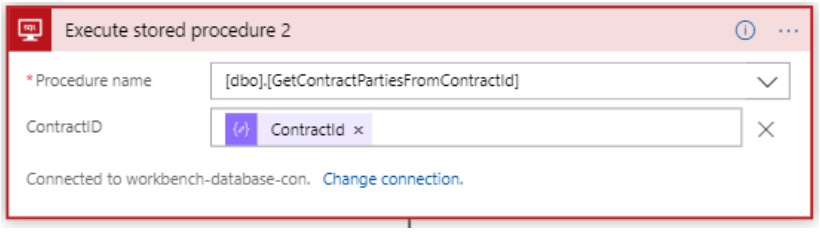
17. Click on **Add an Action**. Click on **Data Operation** then **Parse JSON**. Click on the **Content** field and select **ResultSets**. Paste the following content into **Schema**:

```
{
    "properties": {
        "Table1": {
            "items": {
                "properties": {
                    "COUNT": {
                        "type": "integer"
                    }
                },
                "required": [
                    "COUNT"
                ],
                "type": "object"
            },
            "type": "array"
        }
    },
    "type": "object"
}
```

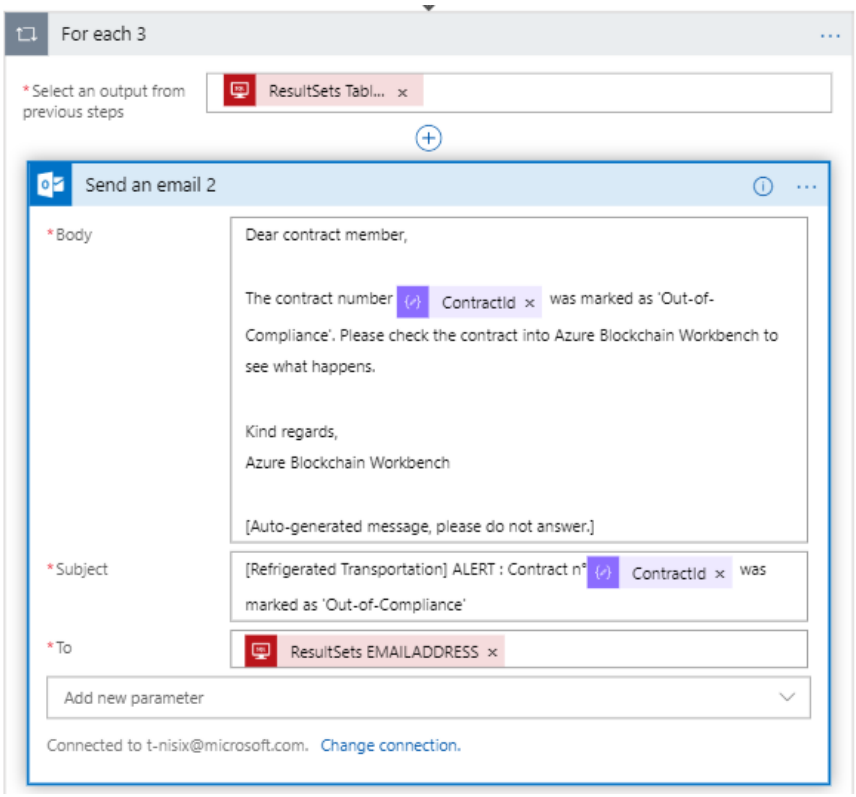
18. Click on **Add an Action**, then on **Control** and **Condition**. Click on the left value field and add the dynamic value `COUNT` from the previous Parse JSON operation, and on the right value field the number 1 (a `For Each` will be automatically added).



19. In the True block, click on **Add an Action** and choose **SQL Server** then **Execute a stored procedure**. Select the procedure **[dbo].[GetContractPartiesFromContractId]** previously created, then add a parameter named **ContractID**, click on the field and select the **ContractID** dynamic value.

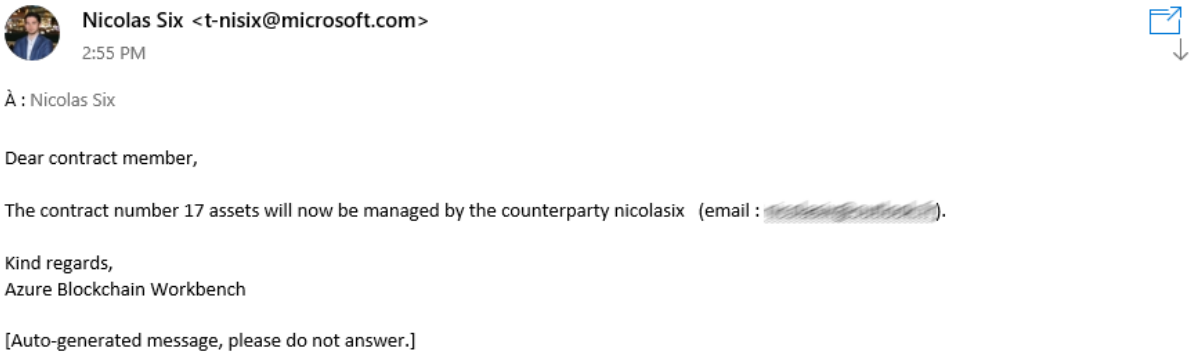


20. The final block will be the email sending. Click on **Add an Action, Office 365 Outlook** then **Send an Email**. Fill the form as the screenshot below, and keep in mind that adding the dynamic values will automatically create a `For each` which will go through the contract parties to send emails.

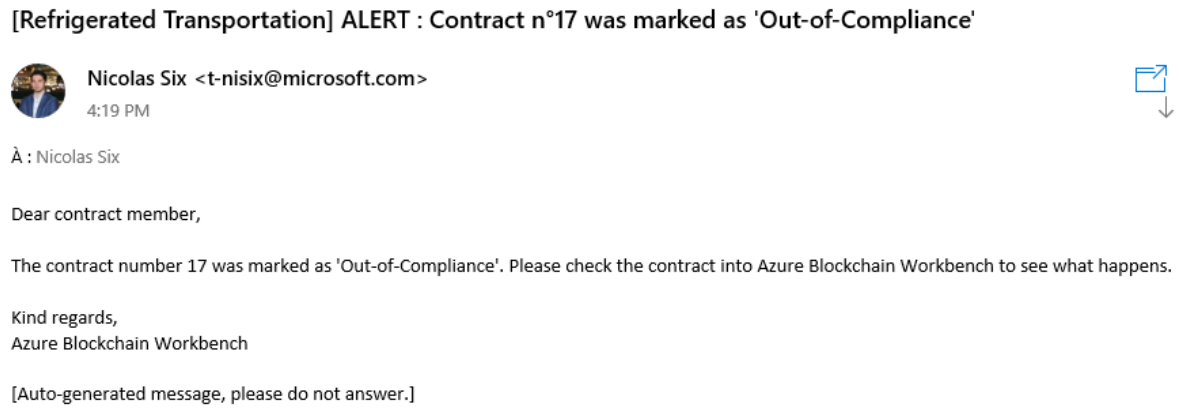


Testing the application

Congrats! Your Logic app, is complete. You can now test it with your Azure Blockchain Workbench. Go inside your Refrigerated Transportation application, then create a contract and change his counterparty to another person. You will receive an email like this one below.



And if you send out-of-range temperature or humidity, the contract will be set on "Out-of-Compliance" and you will get this email:



Your Logic App is now fully functional. Those two functionnalities work really well with the Refrigerated Transportation scenario, but you're free to go even depeer through the Logic App documentation (available here[61]) to develop your own Logic Apps around the Workbench (or not)!

Initiating a rechnician intervention with Connected Field Services

Sometimes, a device can shutdown unexpectedly, for example if its battery is fully discharged, or if a part of the device broke. In such a situation, this is an main issue because a broken device is not able to send telemetry and therefore, insure that the shipment stay at correct temperature and humidity.

A solution to this issue could be a technical intervention, by checking if supply chain devices are correctly sending telemetry using Workbench database, and if not send a technician to repair/change the device.

Setting up Dynamics 365

To begin, you need to have an Dynamics 365 subscription.

[61] Azure Logic Apps Documentation: https://docs.microsoft.com/en-us/azure/logic-apps/

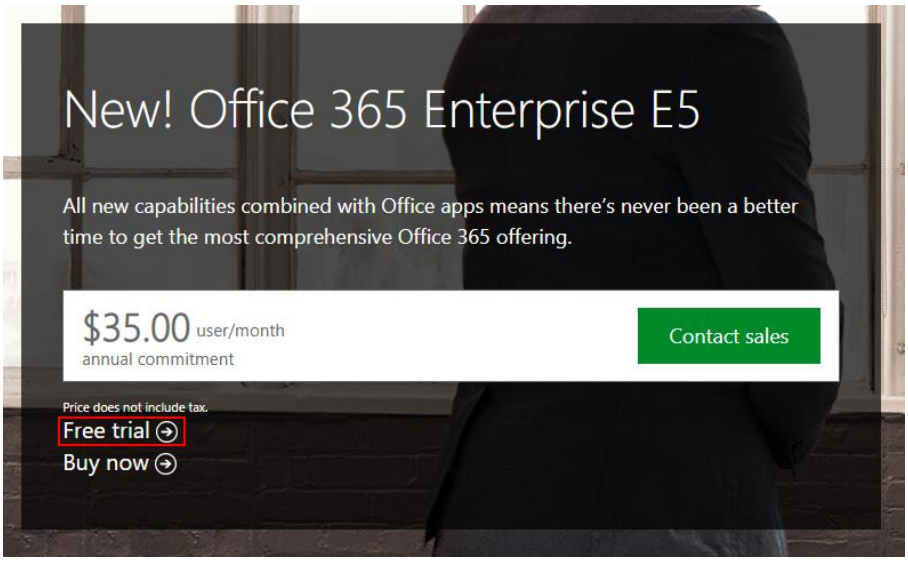
If you already have a running Dynamics 365 with Connected Field Services, you can skip this part and directly start to implement the Logic App.

The objective of this section is to create an Office 365 tenant using the trial version, and do the same for Dynamics 365 and Connected Field Services.

Perform the following steps:

1. Go on Microsoft Office website at this link[62], and click on **Free Trial**.

Nothing you do in this guide should result in any billing to your Azure subscription.



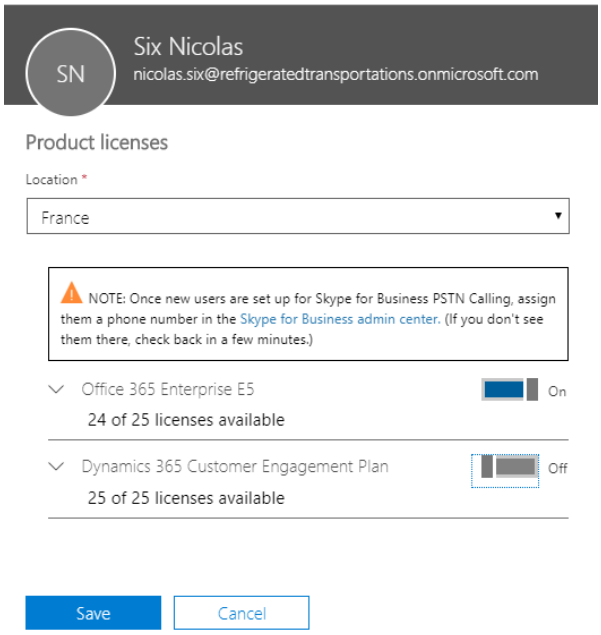
2. Fill the form to create your tenant and your admin user.
3. Go on the admin panel[63], and on the left navbar click on **Billing** > **Subscriptions**, then click on **+Add subscriptions**.
4. Search for **Dynamics 365 Customer Engagement Plan**, put your cursor on the tile and click on **Start free trial**.

Note If you're using the Preview version of the admin panel, you need to click on Billing > Purchase services instead, then click on Dynamics 365 Customer Engagement Plan and click on Get free trial.

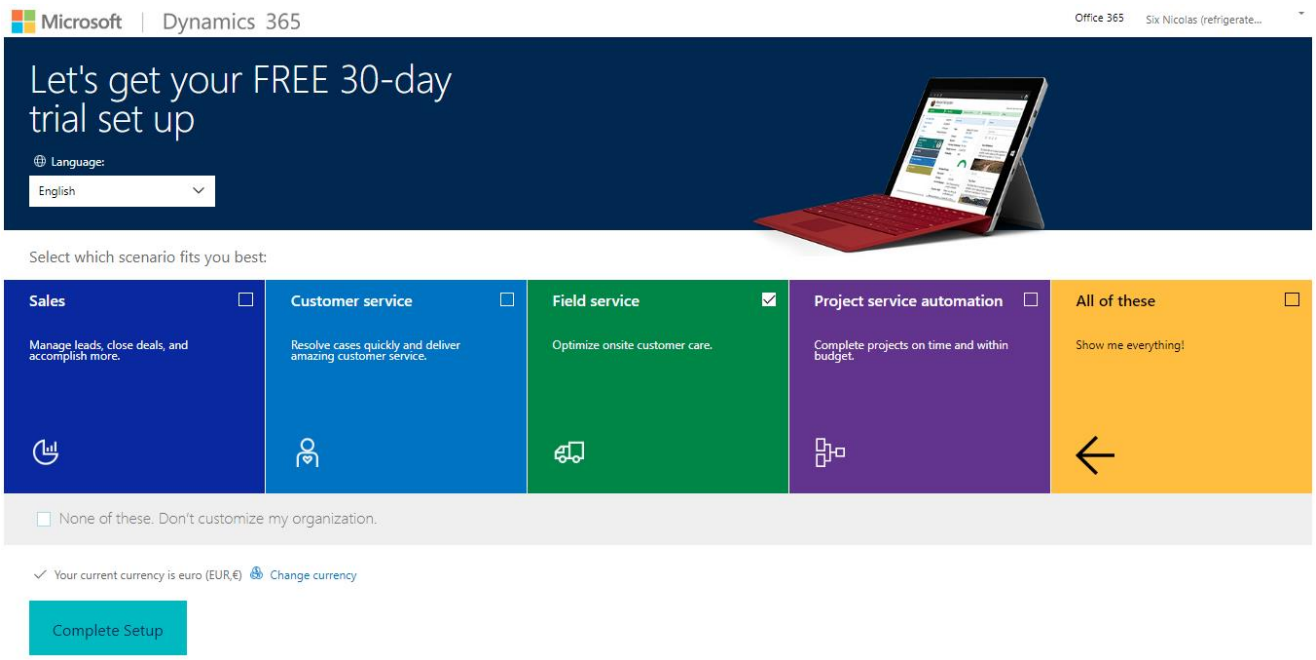
5. Click on **Users** > **Active users**, then click on yourself and in your profile, on **Edit** next to the License section.
6. Turn on **Dynamics 365** for your account and click on **Save**.

[62] Office 365 E5: https://products.office.com/en-us/business/office-365-enterprise-e5-business-software

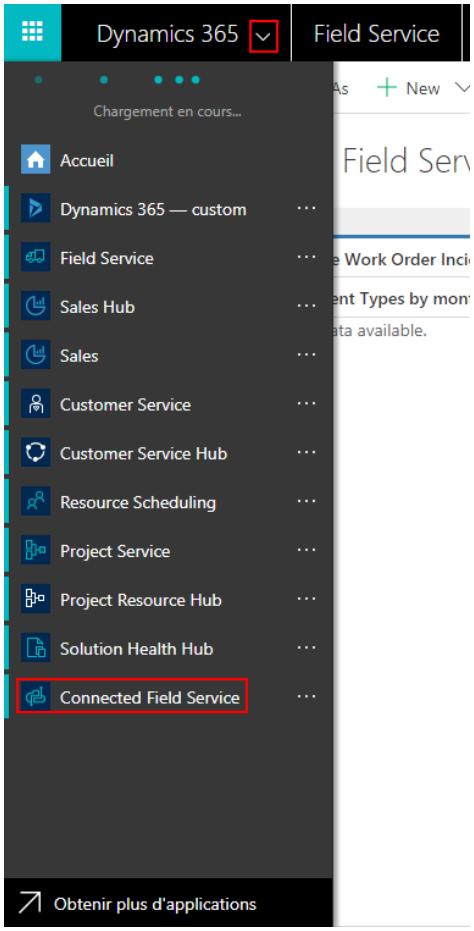
[63] Admin panel: https://admin.microsoft.com/AdminPortal



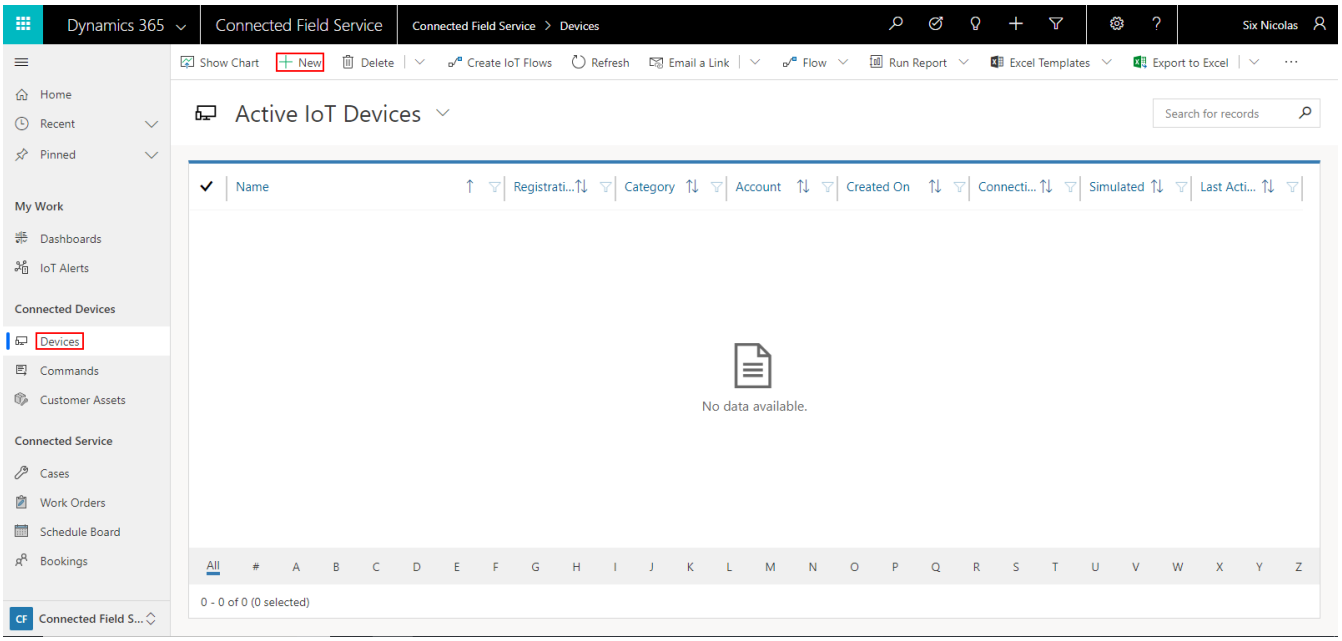
7. Go into **Admin centers** *section*, click on **All admin centers** and choose **Dynamics 365**.
8. You will arrive on a page which invites you to choose your wanted Dynamics 365 scenario. Check at least **Field service** (if you want to try other services, feel free to check other boxes), and click on **Complete Setup**.



9. Wait for setup completion, then click on the down arrow inside the navbar and in the menu, click on **Connected Field Service**.



10. (Optional) The setup is complete and you have now a fully functional Connected Field Service, and even if it's not mandatory, you can create devices to improve IoT alerts reports later (because the alert will be linked with a device profile on Connected Field Service). Click on **Devices**, then **+New.**



Enter its name, device ID and other information if you want to, then click on **Save**. Your device is now created and appears in **Devices** list.

Now that Connected Field Service is set up, you are able to send IoT alerts, and it will be implemented in the next part of this guide.

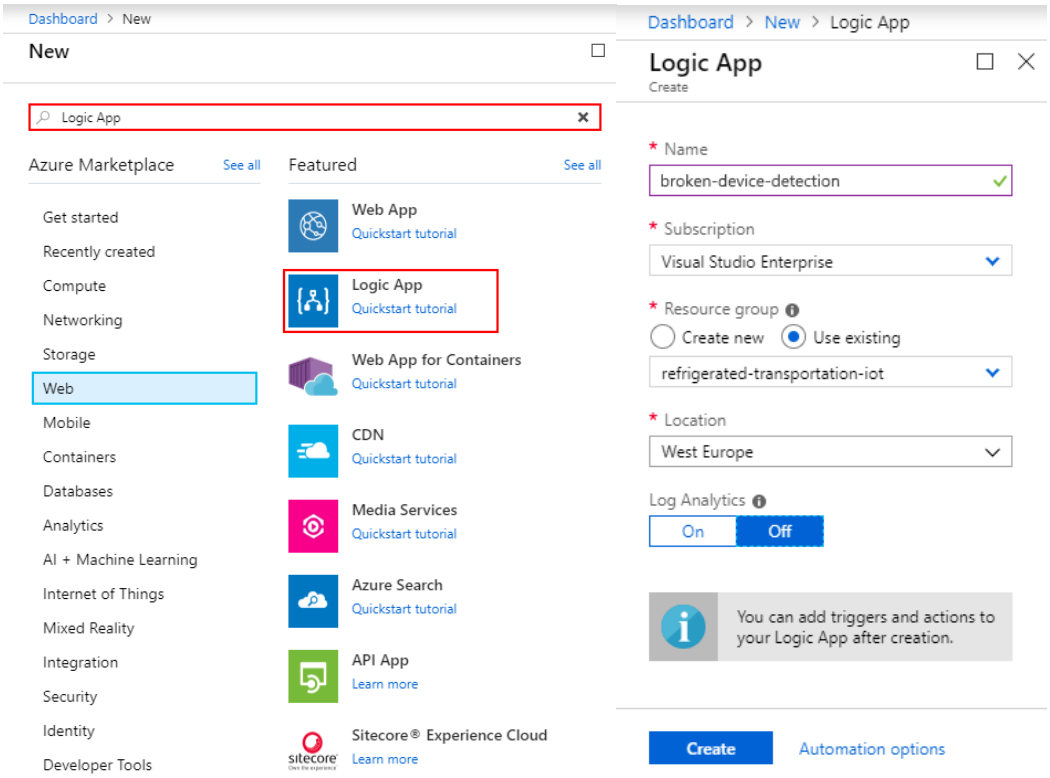
Setting up the Logic App

To create a link between your Workbench and Dynamics 365, you're going to use a Logic App. The objective is to execute a request which will retrieve all inactive devices from existing open contracts, and send an IoT alert into Connected Field Services.

Note When creating a Logic App, you have the possibility to rename each step of the App if you want. This is useful if you want to give a more explicit name, for example name a step "Get contract parties" instead of the default "Execute a stored procedure".

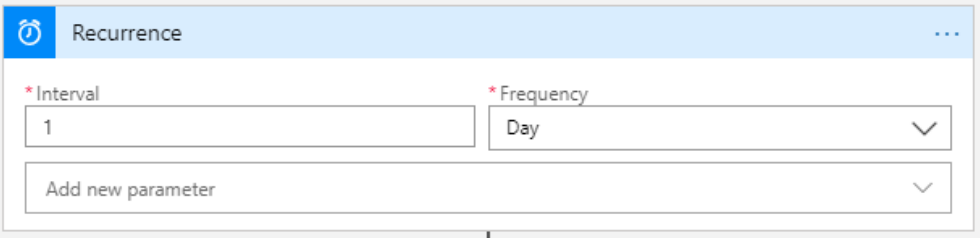
If you want to do so, keep in mind that you should do it when creating the step, and not at the end of the Logic App creation, because it's impossible to rename a step if some data from the step is used somewhere else. In this guide, you should keep default step names.

1. Sign in to the Azure portal.
2. Choose +**Create a resource**, then choose **Web**.
3. Click **Logic App** from the list on the right.
4. Fill the configuration tab.



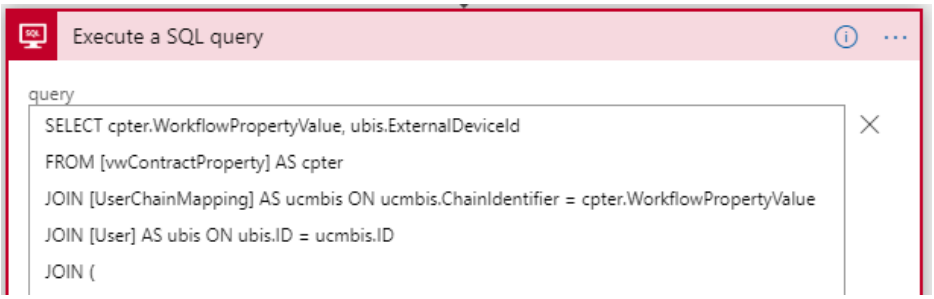
Setting	Description
Name	This is the name of your future app.
Subscription	Select the subscription to use for your Logic App.
Resource group	You can create a new resource group or use an existing one. To create a new one, click **Create** new and fill in the name you want to use. To use an existing resource group, click **Use existing** and select the resource group from the dropdown list.
Location	This is the region in which you want your Logic App to be located. Select the location closest to you from the dropdown list.
Log Analytics	Keep the **Off** parameter for diagnostic logging.

5. After the creation of the Logic App, go inside the **Logic App Designer**. Click on **Schedule** and search for **Recurrence,** then click on it. Fill **Interval** and **Frequency** to enable Logic App triggering every day.



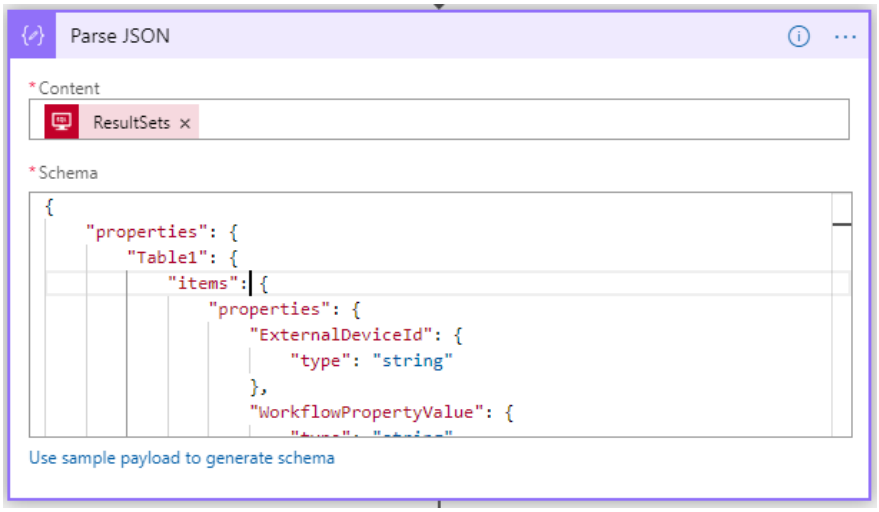
6. Click on **+New Step**. Choose **SQL Server > Execute a SQL query**, and paste the following query inside the **Query** field:

```
SELECT cpter.WorkflowPropertyValue, ubis.ExternalDeviceId
FROM [vwContractProperty] AS cpter
JOIN [UserChainMapping] AS ucmbis ON ucmbis.ChainIdentifier = cpter.WorkflowPropertyValue
JOIN [User] AS ubis ON ubis.ID = ucmbis.ID
JOIN (
         SELECT cp.ContractId, cp.StateValue, cp.WorkflowPropertyName
         FROM [vwContractProperty] cp
         WHERE cp.WorkflowPropertyName = 'State'
         AND StateValue = (
                  SELECT max(cpbis.StateValue)
                  FROM [vwContractProperty] cpbis
                  WHERE cp.ContractId=cpbis.ContractId
         )
    AND StateValue < 2
) AS a ON a.ContractId = cpter.ContractId
WHERE cpter.WorkflowPropertyName = 'Device'
AND NOT EXISTS (
         SELECT u.ExternalDeviceId, ucm.ChainIdentifier
         FROM [Transaction] AS t
         JOIN [ContractEvent] AS ce ON t.TransactionHash = ce.TransactionHash
         JOIN [UserChainMapping] AS ucm ON ucm.ChainIdentifier = t.[FROM]
         JOIN [User] AS u ON u.ID = ucm.ID
         WHERE ce.WorkflowFunctionName = 'IngestTelemetry'
         AND ce.Timestamp >= DATEADD(DAY, -1, GETDATE())
         AND cpter.WorkflowPropertyValue = ucm.ChainIdentifier
)
GROUP BY cpter.WorkflowPropertyValue, ubis.ExternalDeviceId
```

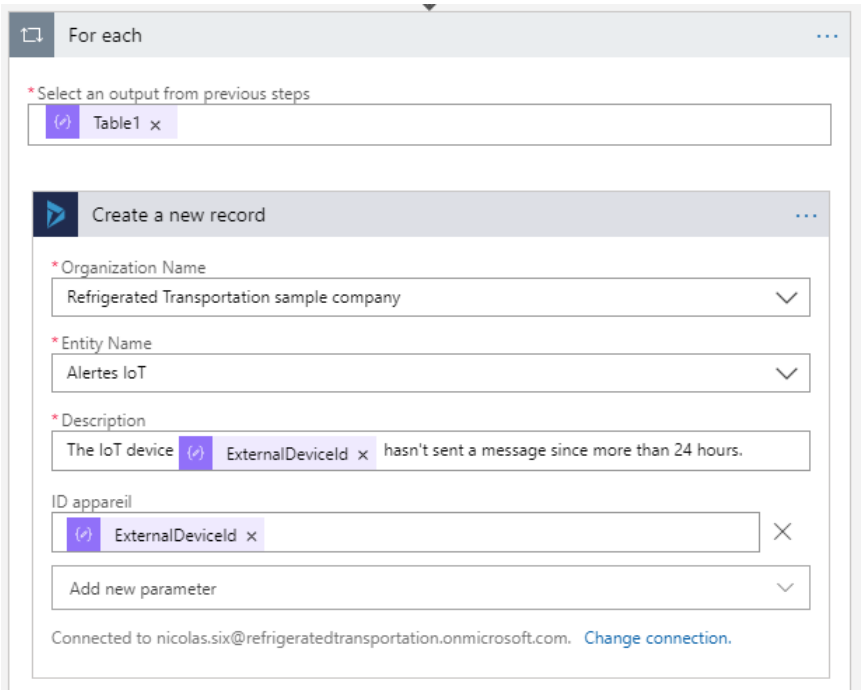


7. Click on **+New Step**. Choose **Data Operations** > **Parse JSON**. Click on the **Content** field, and choose **ResultSet**. Paste this following JSON into the **Schema** field:

```
{
    "properties": {
        "Table1": {
            "items": {
                "properties": {
                    "ExternalDeviceId": {
                        "type": "string"
                    },
                    "WorkflowPropertyValue": {
                        "type": "string"
                    }
                },
                "required": [
                    "WorkflowPropertyValue",
                    "ExternalDeviceId"
                ],
                "type": "object"
            },
            "type": "array"
        }
    },
    "type": "object"
}
```

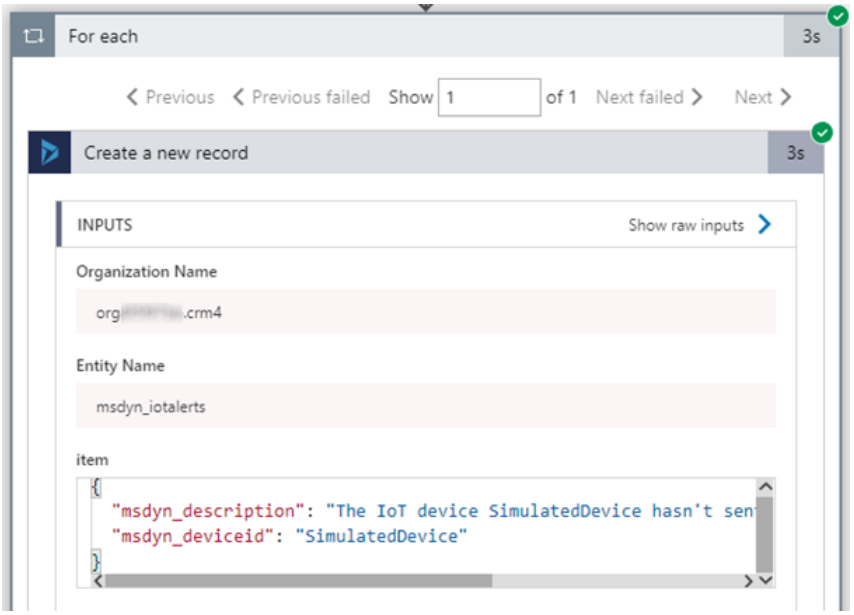


8. Click on **+New Step**. Select **Dynamics 365 > Create a new record**. You will have to log yourself on Dynamics 365, use the account where Dynamics is set up. Fill the form as below:

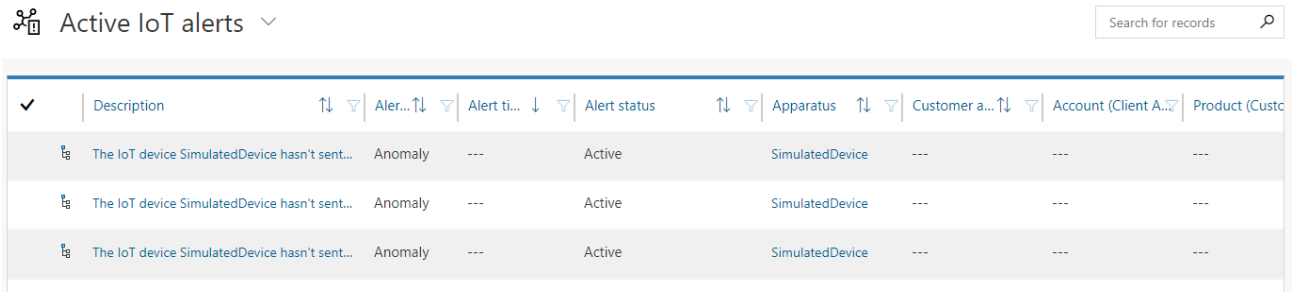


Testing the application

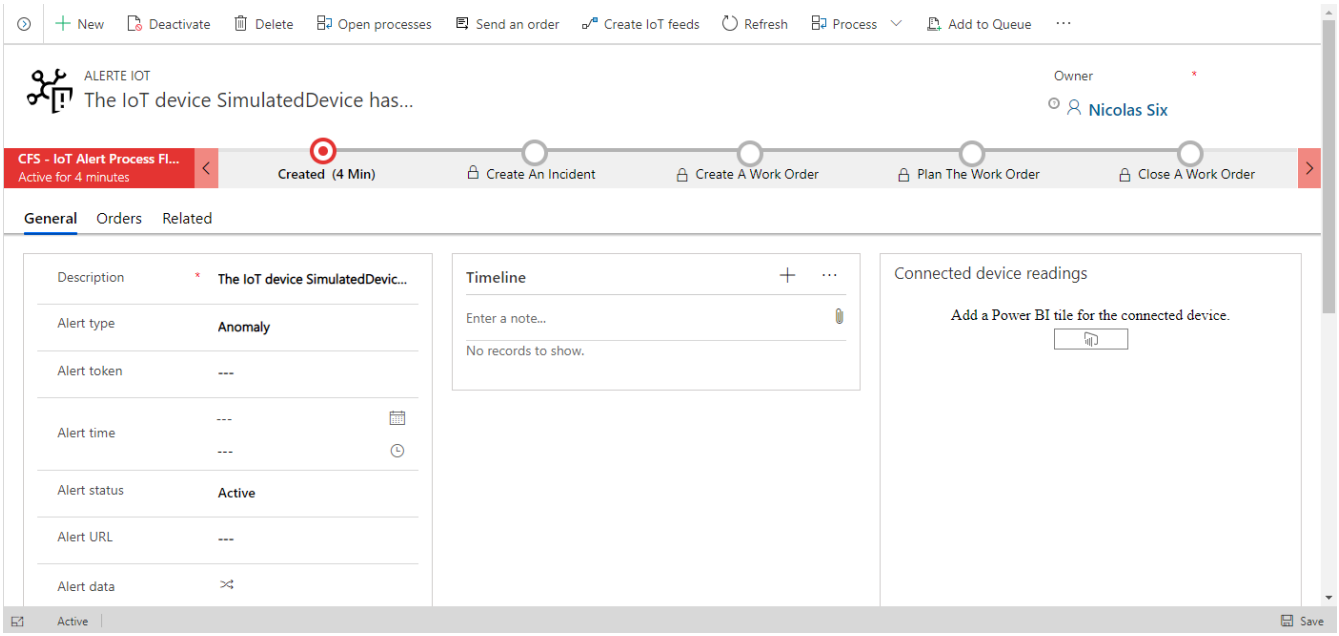
To test the Logic App, just click on the **Run** button to trigger the starting action. If it worked, you should see all steps marked as "*Validated*" and, by clicking on **Create a new record** step, the message sent to Connected Field Services.



Now, if you go on **Connected Field Services** tab > **IoT Alerts**, you should see all the alerts emaning from Azure.



You can click on any alert and see the details like the workflow to resolve the issue generated by Dynamics, or additional details and options linked to the issue.



This was a sample of how Azure Blockchain Workbench can be connected with CRM solutions like Dynamics 365 and its Connected Field Services application.

Buidlng a React Web application

Overview

In this guide, you already saw the default Azure Blockchain Workbench UI, which is great because you can put any kind of application into Workbench and it will render thanks to this UI. But it leads to a downside: the UI can be, sometimes, too generic.

This is fine for little applications or proof-of-concepts (PoCs), but not really suitable for business applications, where every party must have a specific version of the interface based on their job.

Fortunately, the Blockchain Workbench UI is not the only way to interact with the blockchain and execute transactions, because every Workbench is linked with an API, which makes the bridge between you and the blockchain. This is the solution to the problem mentioned before : you can use the API to integrate the blockchain inside an external solution, which can be a web application, a software, etc.

You are not bound to Workbench anymore, you can use your own solutions whatever the langage used to develop it or the plateform, and integrate the blockchain in the part where you really need it.

In this last part, you will see how to build and run a React application which is able to communicate with the Workbench. In this context, the application will replicate the features of the generic interface for the **Refrigerated Transportation** application but in a real world case, you could add more functionalities to the application.

Prerequisites

Before starting this part, check that Node.js is correctly installed on your machine, and also that its version is above v4.x.x. You can enter this command into a console to verify it.

```
node -v
```

Also, open the *refrigerated-sc-sample* that you downloaded before with your console and once inside open go through *Integration* > *webapp*, then type:

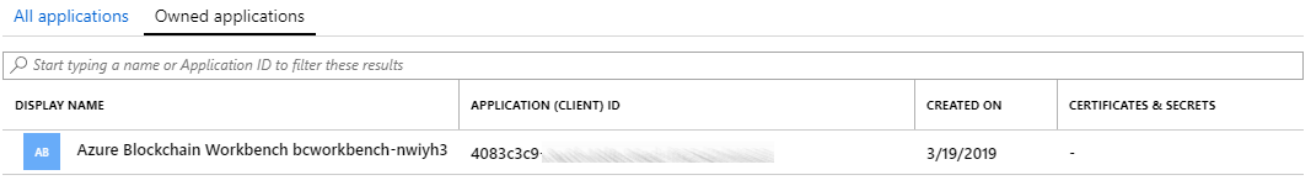
```
npm install
```

Now that you installed the required packages, you need to do one more thing before running the app: fill the config file with your Workbench information.

Open the *webapp* directory with your favorite editor, and open *src* > *js* > *adalConfig.js*.

```
1  import { AuthenticationContext, adalFetch, withAdalLogin } from 'react-adal';
2
3  // App Registration ID
4  const workbenchApiID = '4083c3c9-[redacted]';
5
6  // API Configuration
7  export const adalConfigApi = {
8    cacheLocation: 'localStorage',
9    clientId: workbenchApiID,
10   endpoints: {
11       api: workbenchApiID
12   },
13   tenant: 'microsoft.onmicrosoft.com',
14   redirectUri: window.location.origin + '/',
15   postLogoutRedirectUri: window.location.origin + '/',
16 };
17
18 // Authentication Context definition (contains cookies, tokens ...)
19 export const authContextApi = new AuthenticationContext(adalConfigApi);
20
21 // API Fetch function definition
22 export const adalApiFetch = (fetch, url, options) =>
23   adalFetch(authContextApi, adalConfigApi.endpoints.api, fetch, url, options);
24
25 // HOC Login if needed when accessing to a restricted page
26 export const withAdalLoginApi = withAdalLogin(new AuthenticationContext(adalConfigApi), workbenchApiID);
```

Modify the value of the variable `workbenchApiID` by your own Workbench API ID, which can be found in **Azure AD** > **App registration**:



Also, modify the **tenant** variable by your own tenant, the one that you're using with Azure Blockchain Workbench.

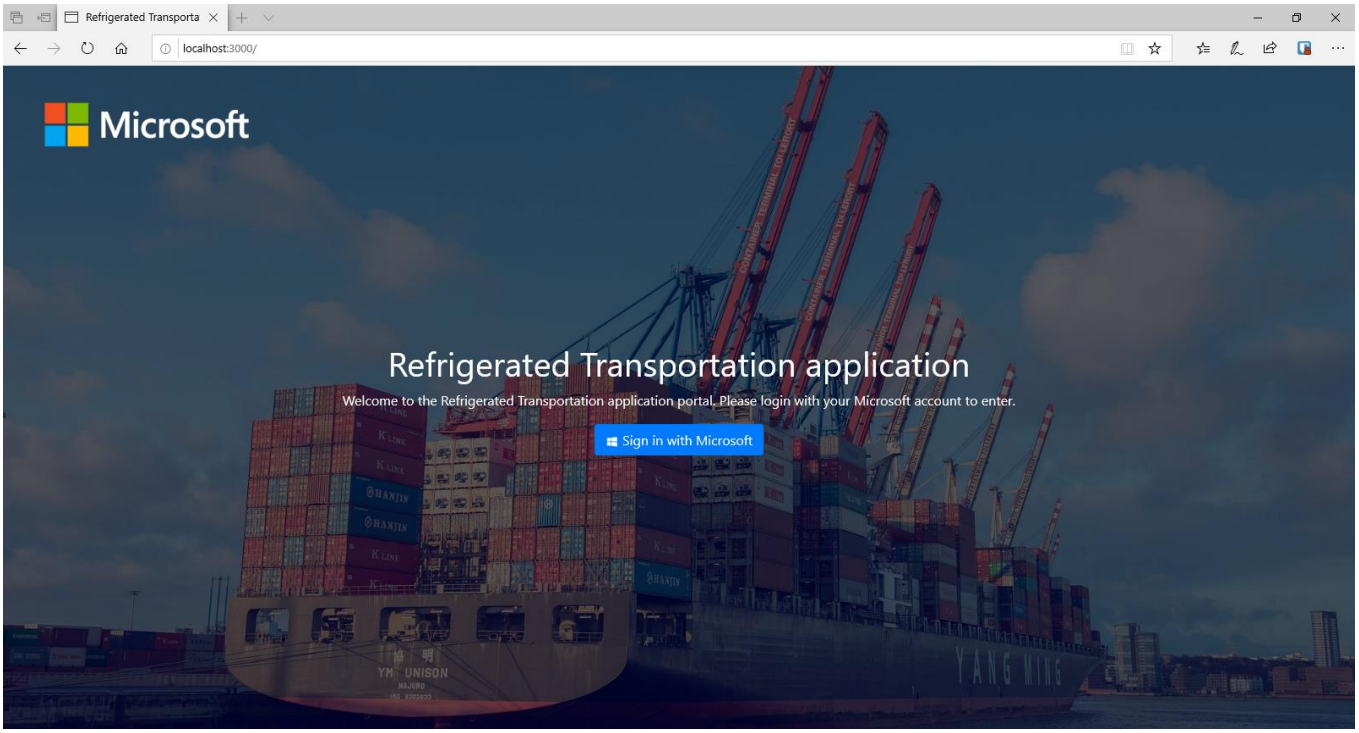
You're now ready to start the development server by typing this command into your terminal:

```
npm start
```

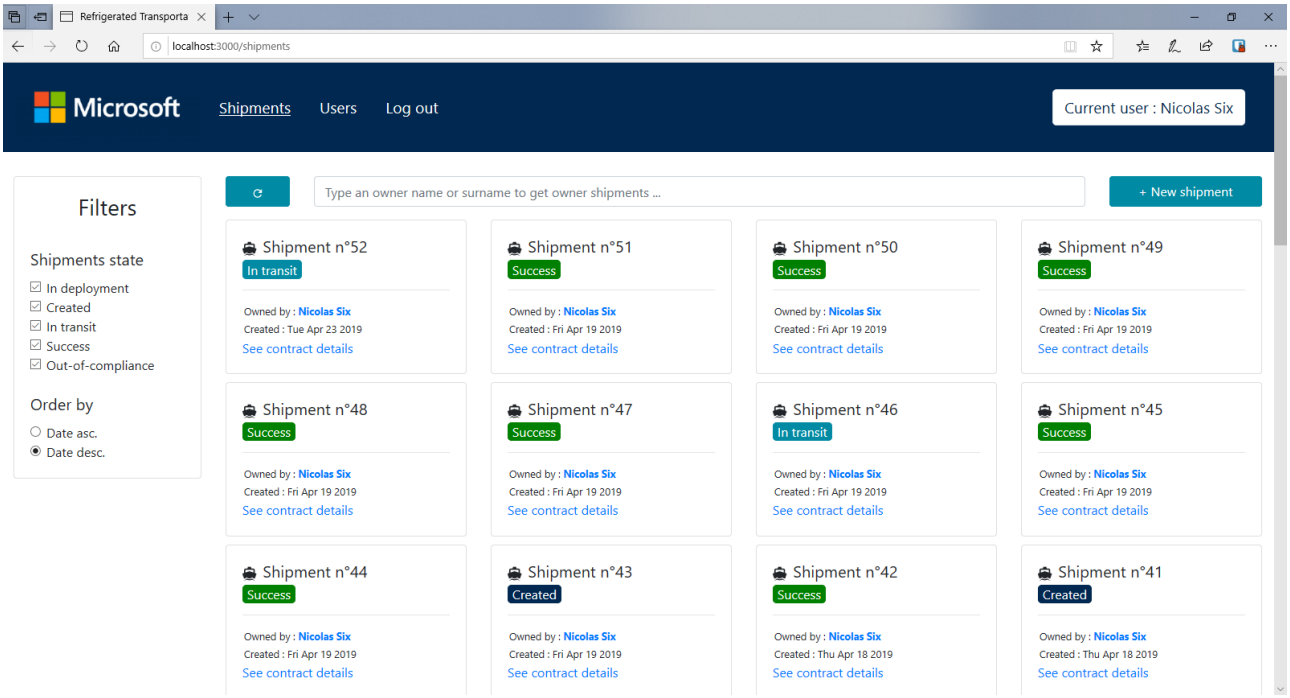
Testing the application

Your application will start and be automatically displayed into your default browser. If not, type "*localhost:3000*" into your URL search bar and press ENTER.

You will arrive on the **Login** screen and you sign in with your (Microsoft) account by clicking on the corresponding button.



Once inside the application, you'll be able to see your application contracts displayed as shipments. The UI is really tailored on the Refrigerated Transportation scenario, so you will have the same view as a supply chain party could have.

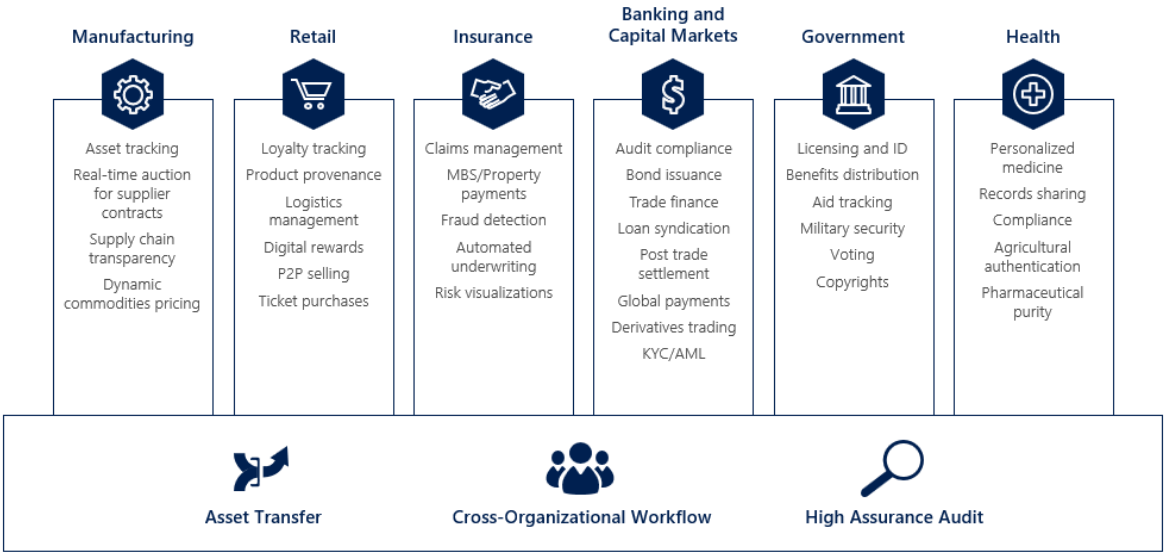

Refrigerated Transporta
localhost:3000/shipments
Microsoft
Shipments
Users
Log out
Current user : Nicolas Six
Filters
Shipments state
In deployment
Created
In transit
Success
Out-of-compliance
Order by
Date asc.
Date desc.
Type an owner name or surname to get owner shipments ...
+ New shipment
Shipment n°52
In transit
Owned by : Nicolas Six
Created : Tue Apr 23 2019
See contract details
Shipment n°51
Success
Owned by : Nicolas Six
Created : Fri Apr 19 2019
See contract details
Shipment n°50
Success
Owned by : Nicolas Six
Created : Fri Apr 19 2019
See contract details
Shipment n°49
Success
Owned by : Nicolas Six
Created : Fri Apr 19 2019
See contract details
Shipment n°48
Success
Owned by : Nicolas Six
Created : Fri Apr 19 2019
See contract details
Shipment n°47
Success
Owned by : Nicolas Six
Created : Fri Apr 19 2019
See contract details
Shipment n°46
In transit
Owned by : Nicolas Six
Created : Fri Apr 19 2019
See contract details
Shipment n°45
Success
Owned by : Nicolas Six
Created : Fri Apr 19 2019
See contract details
Shipment n°44
Success
Owned by : Nicolas Six
Created : Fri Apr 19 2019
See contract details
Shipment n°43
Created
Owned by : Nicolas Six
Created : Fri Apr 19 2019
See contract details
Shipment n°42
Success
Owned by : Nicolas Six
Created : Thu Apr 18 2019
See contract details
Shipment n°41
Created
Owned by : Nicolas Six
Created : Thu Apr 18 2019
See contract details

As a conclusion

By following this guide, you should have learned how to implement a lot of solutions within Azure and its Workbench, or outside the Azure ecosystem (as the IoT part, the React App, etc.).

Now, you should have a great overview of what you can achieve with Azure Blockchain Workbench, and how the Workbench can be applied to a wide range of applications in multiple domains.

You saw that a supply chain is a perfect use case for blockchain but there are a lot of other kind of applications which can be developed.



Now, you are not only in a position to rebuild the solution introduced in this guide, but you also can project yourself on how Azure Blockchain Workbench could be integrated into your own business workflows to improve them and save additional costs!

This concludes this guide for developers.

Microsoft France
39 Quai du Président Roosevelt
92130 Issy-Les-Moulineaux