Microsoft

Implementing an end-to-end solution using Azure Blockchain Workbench

For a real-life scenario: A supply chain of refrigerated products

Table of Contents

Overview

Introduction

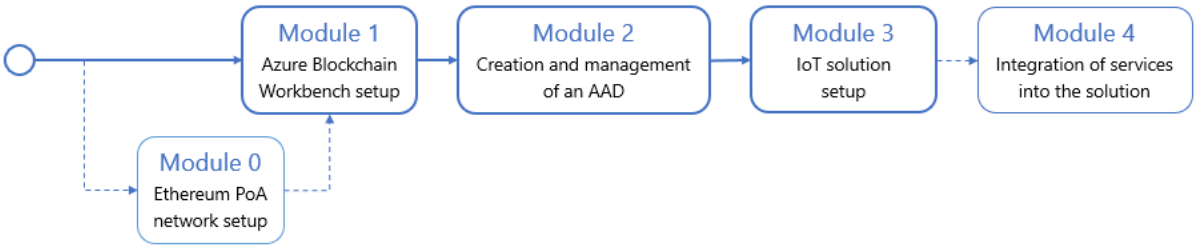
Since May 2018, Azure Blockchain Workbench is publicly available. The main objective of this service is to allow users to deploy a blockchain network with just a few clicks, in order to focus on DApps (Decentralized Applications) development instead of the main blockchain problematics.

Thanks to its integration in Azure, the workbench can be used with a lot of others Azure services, like Azure AD to manage identities and access policies to the DApps and the workbench, or Azure Logic Apps to create powerful apps around Azure Blockchain Workbench through Azure Event Grids.

The main objective of this guide is to show you how to implement an end-to-end solution based on a real scenario, which will capitalize on Azure Blockchain Workbench to reveal its full potential: refrigerated transportation of products.

You can find more details about the scenario and its Solidity/JSON code at this link. Indeed, a lot of samples and technology demos are already available on Github, but our objective here is to show the implementation of a complete solution rather than bricks.

In this guide, you will follow several modules, and each module within the course is built on the previous one. However, as the solution is really customizable, some modules or module contents are optional.



At the end, you will learn to:

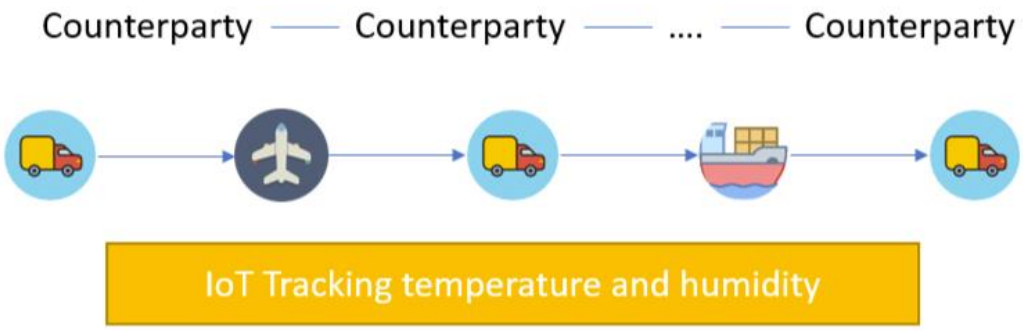
- Create an Ethereum PoA network within Azure.
- Deploy a Azure Blockchain Workbench on this Ethereum PoA network or on a fresh blockchain network.
- Create and manage an Azure Active Directory to manage members of the Workbench and applications participants.
- Build an IoT solution by deploying a IoT Hub, a IoT Service Bus/Queue and a Logic App, then by programming and running connected devices.
- Integrate multiple Azure services into the solution (Different types of Logic Apps, Dynamics 365 services, …).

Scenario

Think about a supply chain of refrigerated products, like meat, frozen prepared dishes, pharmaceuticals products …
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 IoT (Internet of Things) sensor can be used. Attached to the goods during all the transport process, its role is to report the measures in real time

and check if one of those 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 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 is here to keep track of the truth during all the transport process.

To do that, when a shipment order is created, a smart contract is created at the same time inside the blockchain. It specifies the initiating counterparty, the device which is responsible for measuring the temperature inside the shipment and the owner of the product 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.

Requirements

To set up this solution, you will need:

- A Microsoft account
- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin.
- The project which contains the code for every part of this guide, you can grab it here from Github. Save it where you can easily find it.

Optionally, you may need additional things for each tutorial in this guide. Check out the tab below if you want to know requirements needed for each part.

Note: Some requirements can be redundant depending of parts choosed.

Ethereum PoA Consortium Setup	➢ *Mozilla Firefox* (or any *MetaMask* compatible browser). You can download it here.
IoT solution setup > Simulated device	➢ Node.js v4 or later. If you don't installed it yet, you can download it here. ➢ A code editor (like Visual Studio or Visual Studio Code).
IoT solution setup > MXChip IoT DevKit	➢ A physical MXChip IoT DevKit. ➢ ST-Link drivers, available here for Windows, *(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. ➢ Visual Studio Code.
IoT solution setup > Azure Sphere	➢ A physical Azure Sphere. ➢ MT3620 Grove Shield (available here) ➢ Visual Studio. ➢ Azure Sphere SDK Preview for Visual Studio, available here. ➢ Grove SHT31 Humidity & Temperature sensor (available here)
IoT solution setup > Raspberry Pi	➢ A physical Raspberry Pi. ➢ 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. ➢ A software which can flash disk images, like Etcher.

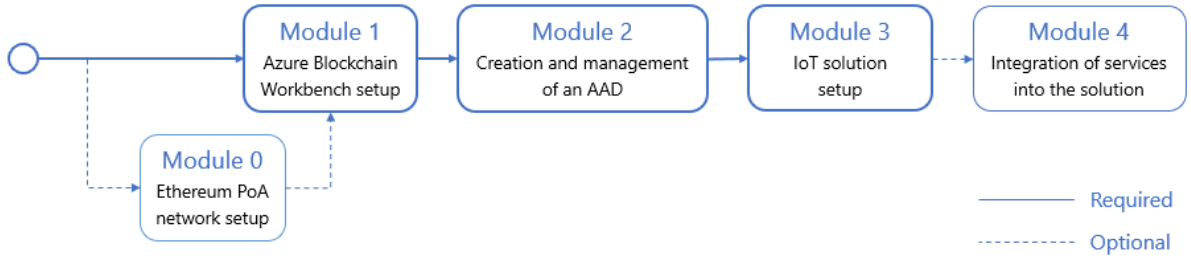
	➢ Grove SHT31 Humidity & Temperature sensor (available here) ➢ 4 male-to-female breadboard cables.
Integration of services > Connected Field Services	➢ An *Office 365* subscription ➢ A *Dynamics 365* subscription *Note: if you don't have those licences, there is a Office 365 & Dynamics 365 30-days trial available, follow the guide to find out how to setup it.*
Integration of services > React web app	➢ Node.js v4 or later. If you don't installed it yet, you can download it here. ➢ A code editor (like Visual Studio or Visual Studio Code).

Module 0 : Ethereum Proof-of-Authority Consortium network setup

Overview

When you create an *Azure Blockchain Workbench*, you can choose to set up a new network which will be a set of *Ethereum Proof-of-Authority (PoA)* nodes. 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 you already have a working Ethereum PoA blockchain) in order to understand how does it work or custom the network, you can read this section in order to set it up and then, get the *Endpoint URL* which will be used 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.

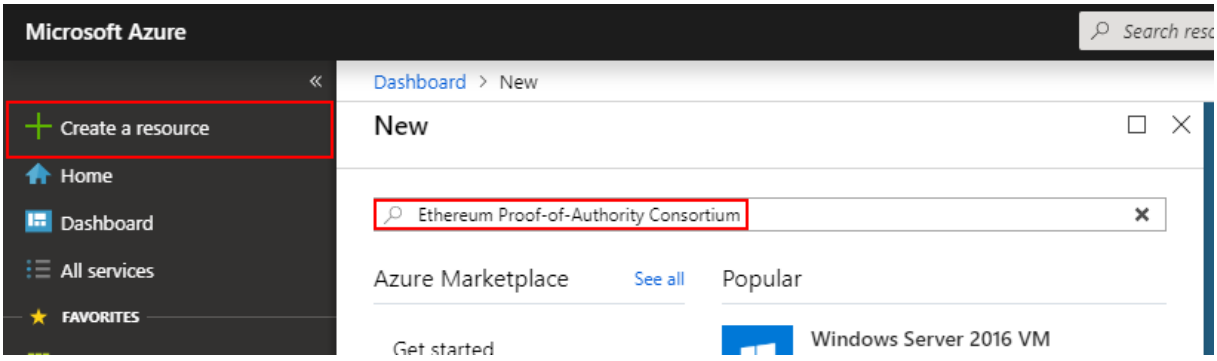
Network deployment

First of all, you need to log into the Azure portal in order to create an *Ethereum Proof-of-Authority Consortium*.

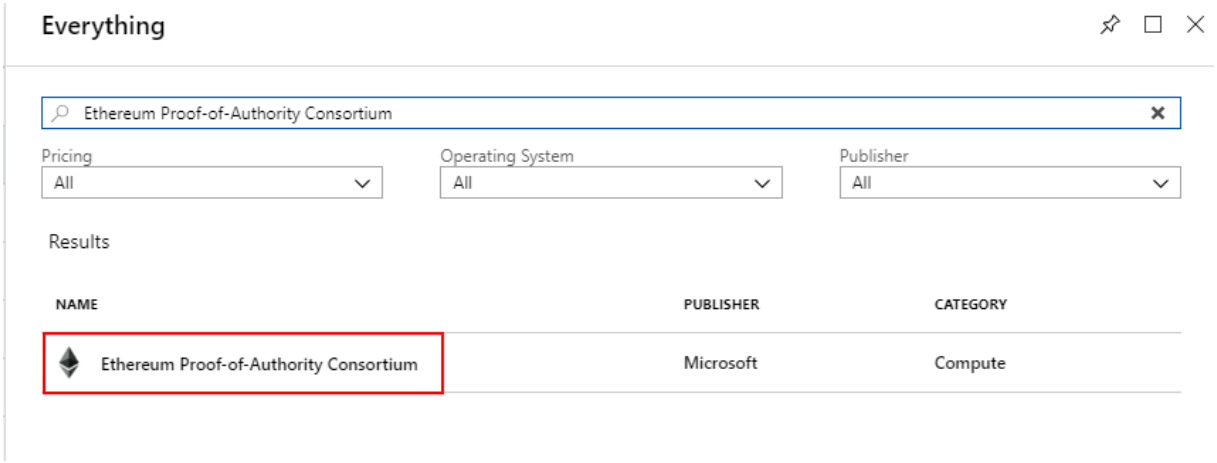
1. Sign in to the Azure portal.

2. Select your account in the top-right corner, and switch to the desired Azure AD tenant where you want to deploy your *Ethereum Proof-of-Authority Consortium*.

3. In the left pane, select **Create a resource**. Search for *Ethereum Proof-of-Authority Consortium* in the **Search the Marketplace** search bar.

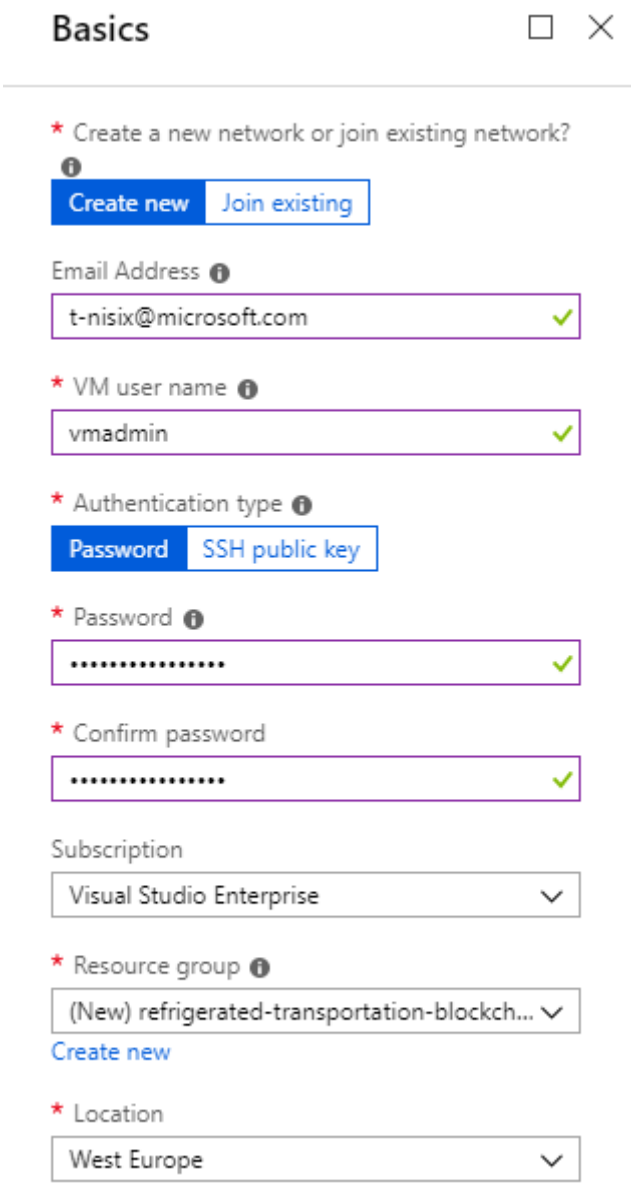


4. Select **Ethereum Proof-of-Authority Consortium**



5. Select **Create**

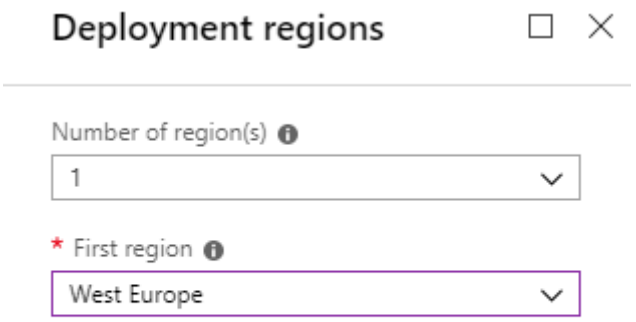
6. Complete the basic settings



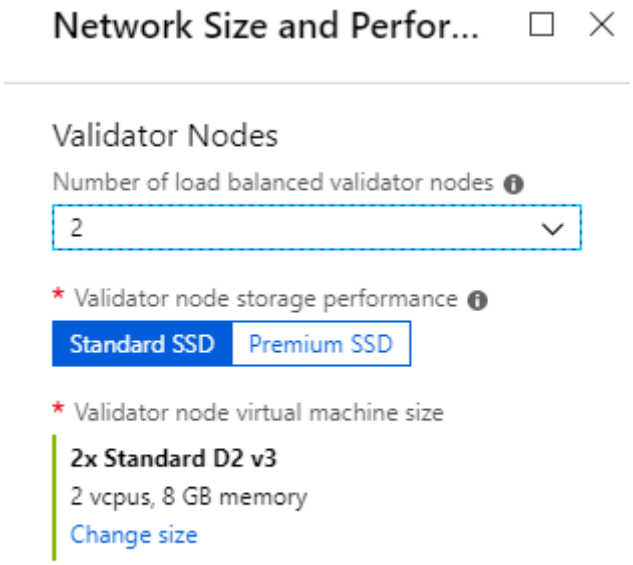
Setting	Description
Create new or Join existing	Choose if you want to create a new blockchain network or if you want to join an existing network.
Email Address	An Email Address which will receive a confirmation email when the blockchain deployment will end with additional information.
VM user name	The user name is used as an administrator for all virtual machines (VM).
Authentication type	Select if you want to use a password or key for connecting to VMs.
Password	The password is used for connecting to VMs.

Setting	Description
SSH	Use an RSA public key in the single-line format beginning with *ssh-rsa* or use the multi-line PEM format. You can generate SSH keys using ssh-keygen on Linux and OS X, or by using PuTTYGen on Windows. More information on SSH keys, see How to use SSH keys with Windows on Azure.
Database password / Confirm database password	Specify the password to use for access to the database created as part of the deployment.
Subscription	Specify the Azure Subscription you wish to use for your deployment.
Resource groups	Create a new Resource group by selecting **Create new** and specify a unique resource group name.
Location	Specify the region you wish to deploy the framework.

7. Complete the **Deployment Regions Settings** by choosing the number of regions to deploy VMs into and specify the regions below this choice. Keep the number of regions by default if you don't have specific needs.



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 specific requirements.

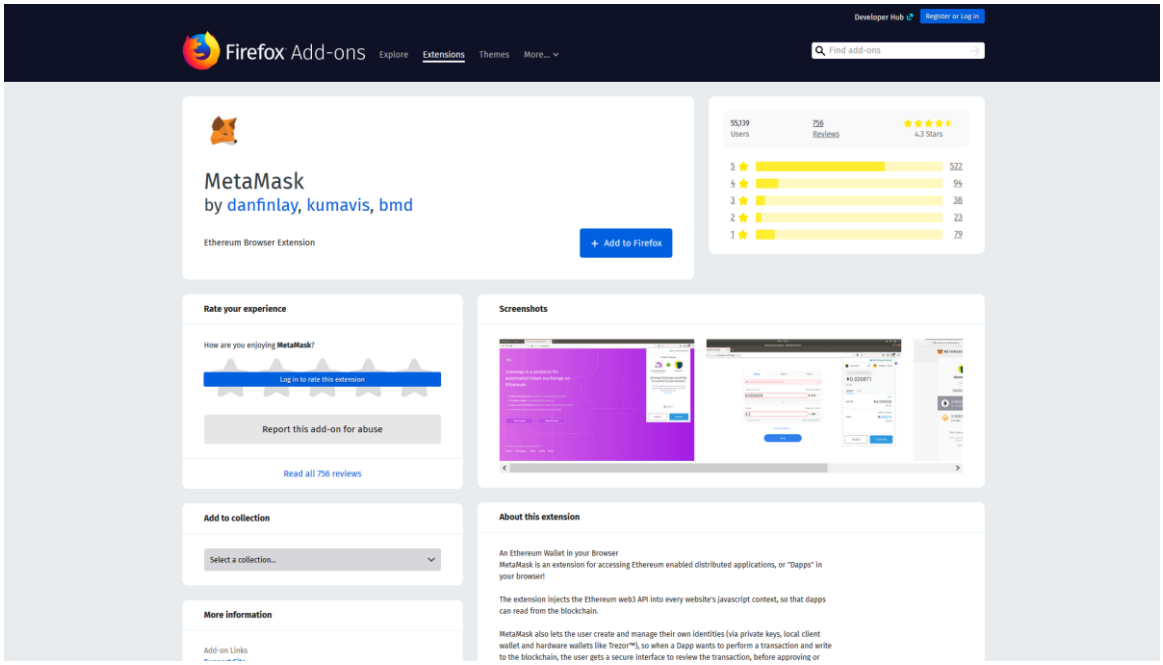


9. Ethereum Settings.

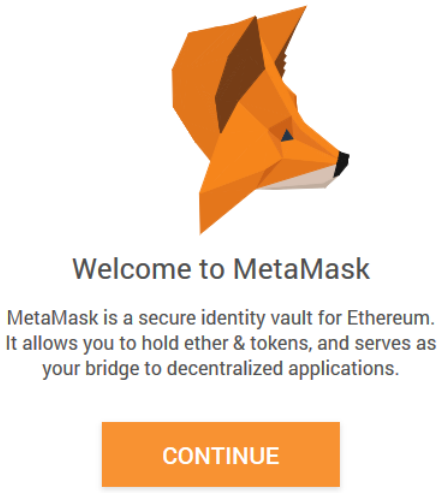
At this step, you will arrive on **Ethereum Settings**. This section will require you to have a public Ethereum address. This address is a 64-character hex string generated which will represent your identity on the blockchain.

In order to get it, you will need to install *MetaMask*. This is a browser extension compatible with *Chrome, Firefox, Opera and Brave*. For this guide, we will install it on Firefox, but the extension can work on any browser previously mentioned. Open a new tab if you are on a compatible browser or choose another one if not.

Go on the MetaMask website, 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**.



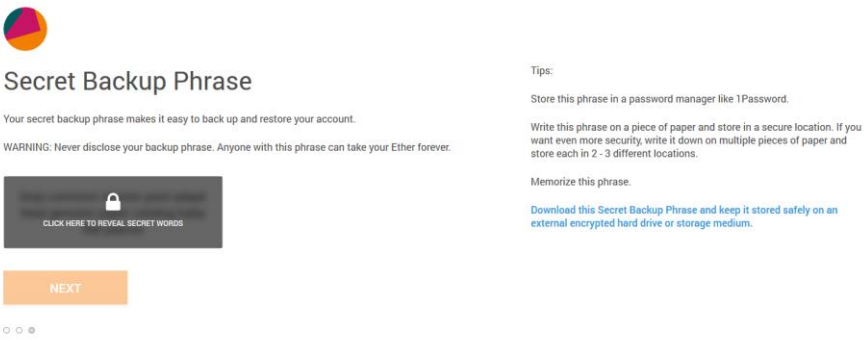
After that, the extension will open itself, and prompt you to begin the installation. Click on **Continue.**



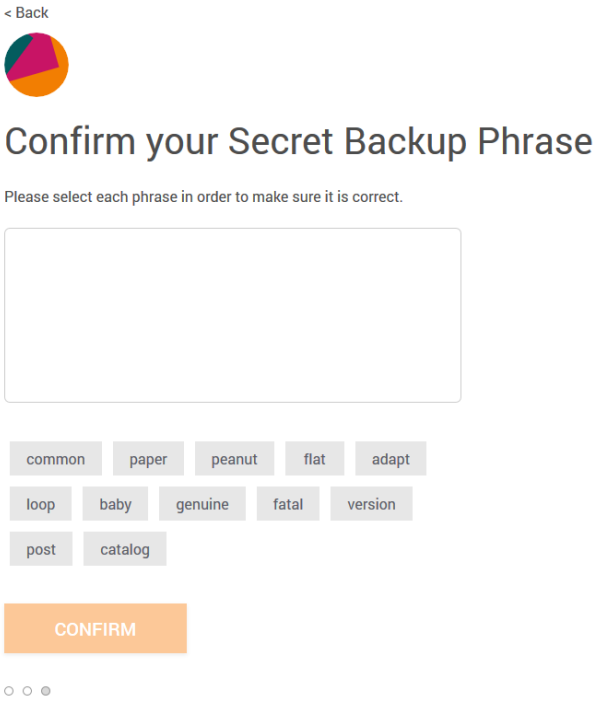
It will ask you to create a password. Type one and click on **Create,** then **Next** and accept the **Terms of Use, Privacy Notice & Phishing Warning**.

You will arrive on the Secret Backup Phrase page. Click on the lock to reveal the secret words. This phrase really important, because they represent your *Ethereum Account*. Indeed, thanks to a cryptographic algorithm, you can generate your *Ethereum Wallet* from this phrase.

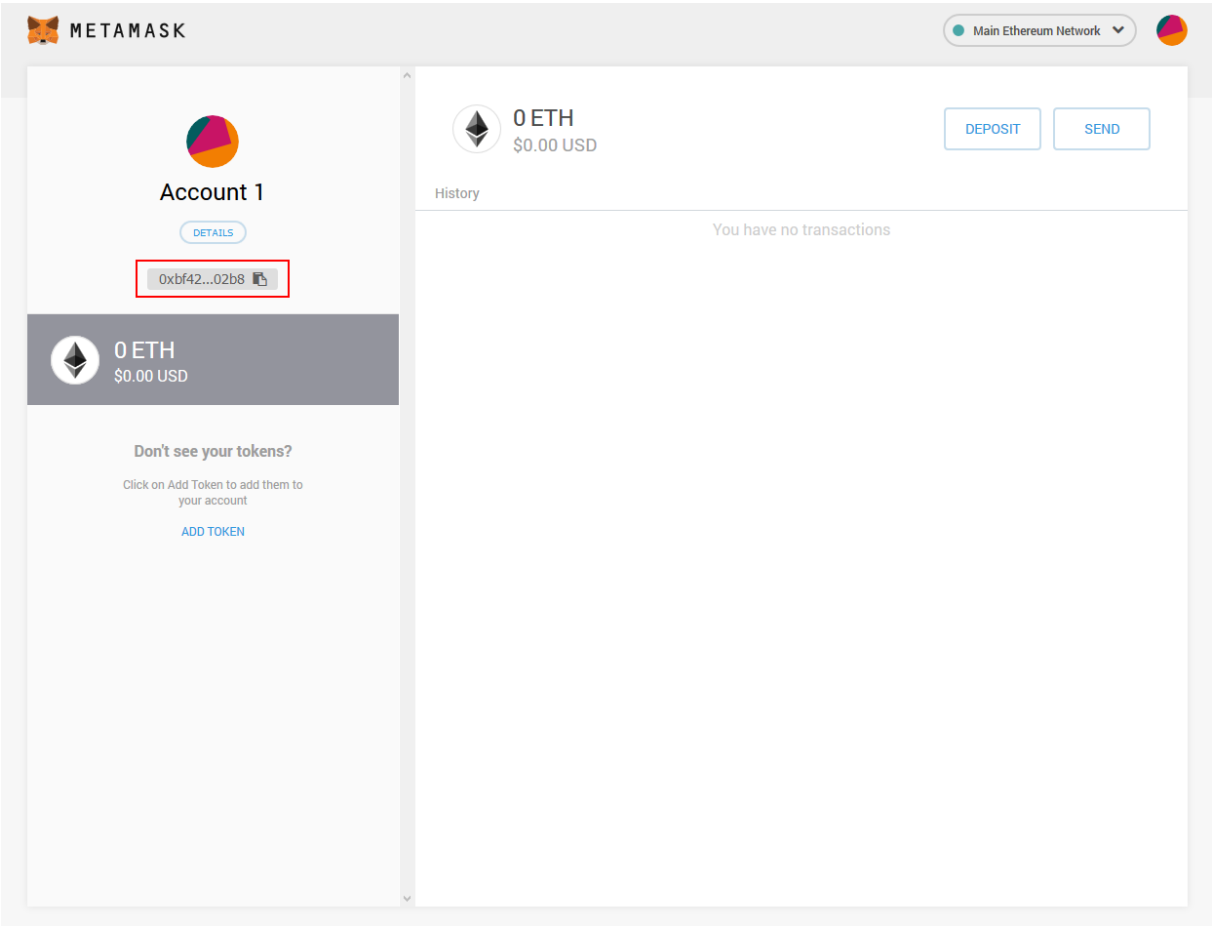
Note: if you want to use this wallet in the future and have the possibility to retrieve your account in any case, ***please keep this phrase secret*** *as anybody who has this phrase is able to connect to your account****, and save it somewhere*** *like on a paper or in a password manager.*



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**.

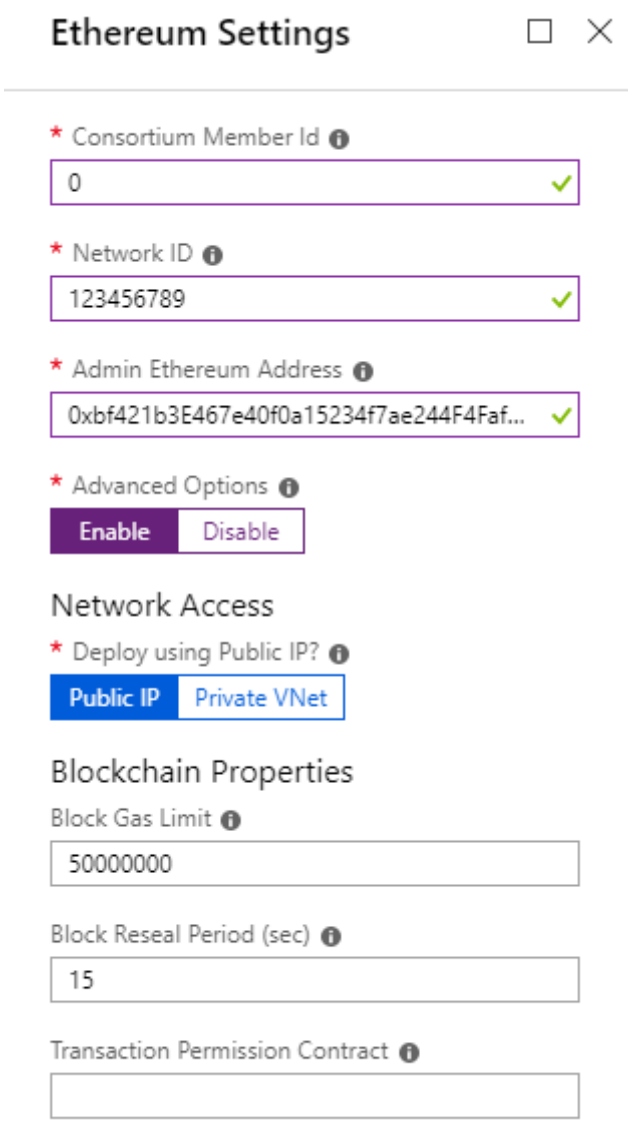


Now that the installation is finished, you just have to copy the generated address by clicking on it inside the application. This will be your identity on *Azure Ethereum PoA Consortium* network.



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 logging in with MetaMask.

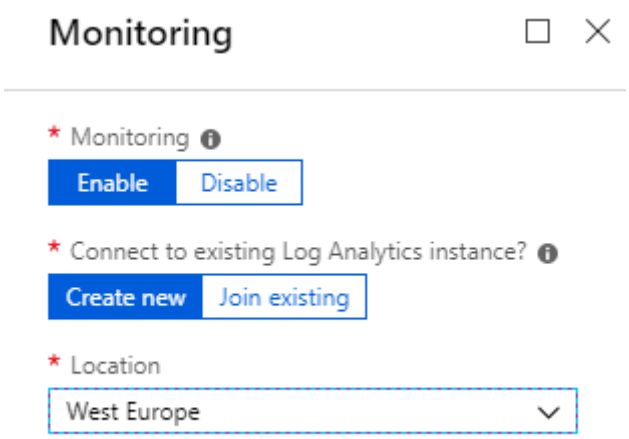
10. Go back on your **Ethereum Settings** tab and complete it.



Parameter name	**Description**	**Allowed values**
Consortium Member ID	The ID associated with each member participating in the consortium network used to configure IP address spaces to avoid collision. In the case of a private network, Member ID should be unique across different organizations in the same network. A unique member ID is needed even when the same organization deploys to multiple regions. Make note of the value of this parameter since you will need to share it with other joining members to ensure there's no collision.	0-255
Network ID	The network ID for the consortium Ethereum network being deployed. Each Ethereum network has its own Network ID, with 1 being the ID for the public network.	5 - 999,999,999
Admin Ethereum Address	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	Advanced options for *Ethereum* settings	Enable or Disable
Public IP (Advanced Options = Enable)	Deploys the network behind a VNet Gateway and removes peering access. If this option is selected, all members must use a VNet Gateway for the connection to be compatible.	Public IP Private VNet
Block Gas Limit (Advanced Options = Enable)	The starting block gas limit of the network	Any numeric
Block Reseal Period (sec)	The frequency at which empty blocks will be created when there are no transactions on the network. A higher frequency will have faster finality but increased storage costs.	Any numeric
Transaction Permission Contract	Bytecode for the Transaction Permissioning contract. Restricts smart contract deployment and execution to a permissioned list of Ethereum accounts.	Contract bytecode

(Advanced Options = Enable)		

11. 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	Option to enable 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)	The region where the new Azure Monitor logs instance will be deployed	All Azure Monitor logs regions
Existing Log Analytics Workspace Id (Connect to existing Azure Monitor logs = Join Existing)	Workspace ID of the existing Azure Monitor logs instance	

Parameter name	Description	Allowed values
Existing Log Analytics Primary Key (Connect to existing Log Analytics = Join Existing)	The primary key used to connect to the existing Log Analytics instance	

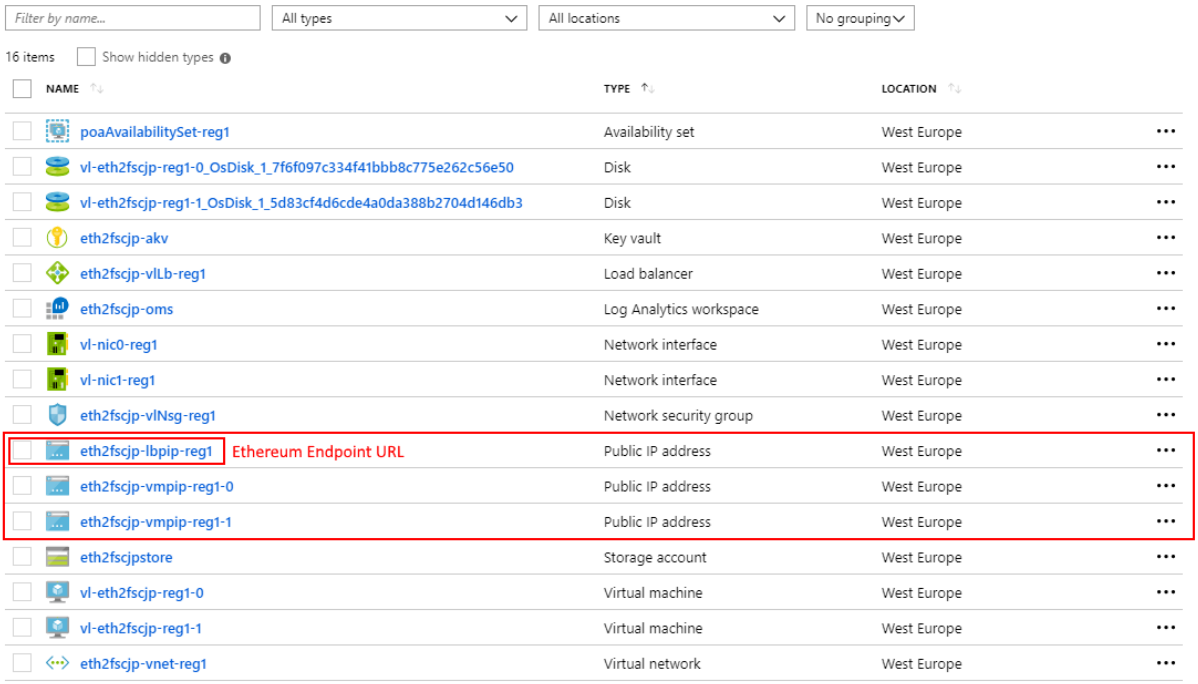
12. The summary of your choices should appear now. Wait for the automatic validation, then click on **OK** if everything is correct. On the next window, click on **Create**.

The deployment will take up to 50 minutes. When completed, you will receive an email on the address you specified on the first step, describing the parameters of your blockchain.

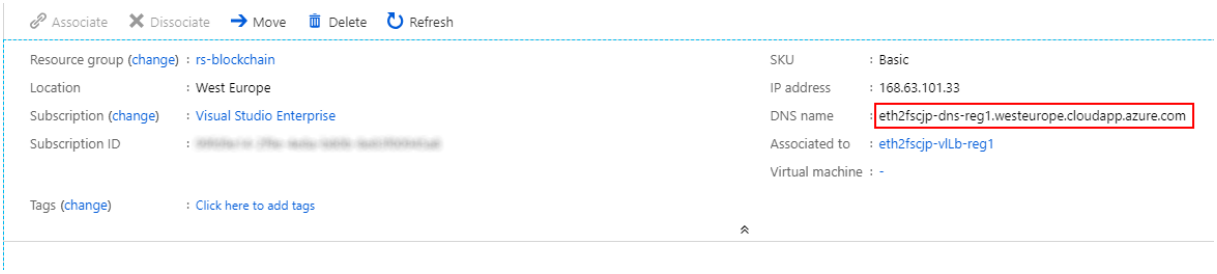
Now that your *Ethereum network* is online, you can get the *Ethereum Endpoint URL* which will be used later by the workbench.

On Azure, 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 we are searching is the one which its 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.



Module 1 : Azure Blockchain Workbench setup

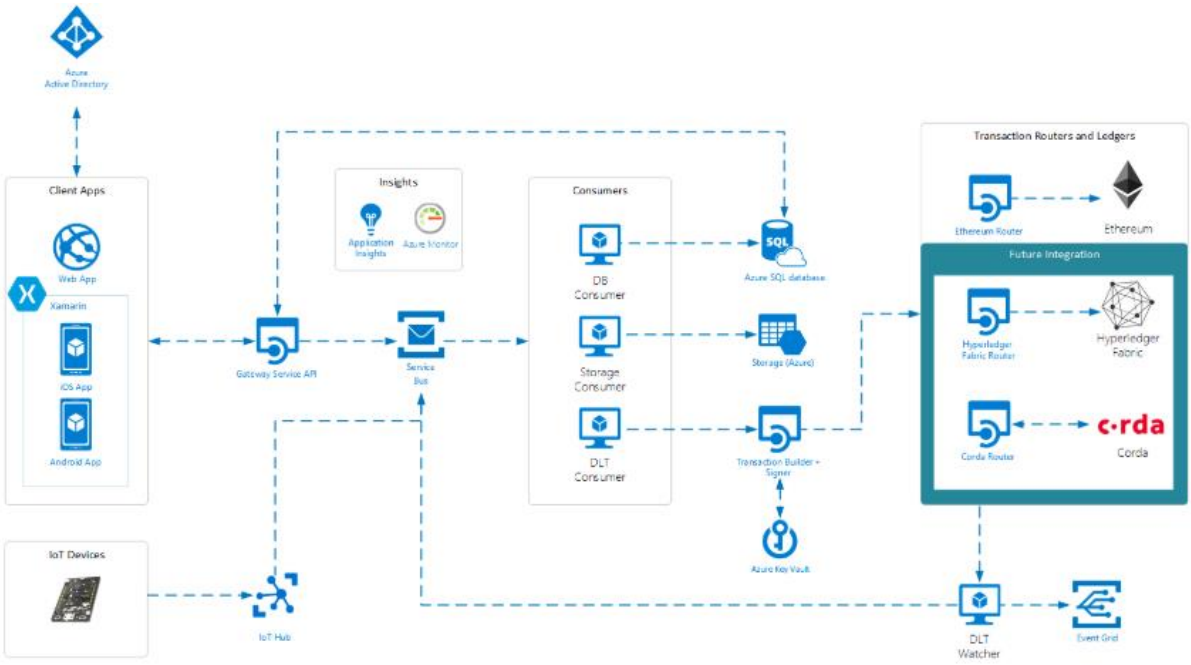
Overview

Whether you already created an *Ethereum PoA consortium* network or not, we're ready to deploy and set up the cornerstone of our application: *Azure Blockchain Workbench*.

This Azure solution is able to plug itself to an existing *Ethereum PoA network* or to create a new one, and it adds a high-level layer to the blockchain. Indeed, with *Azure Blockchain Workbench* you get a graphical interface which enables you to quickly deploy new decentralized applications, or use the existing ones.

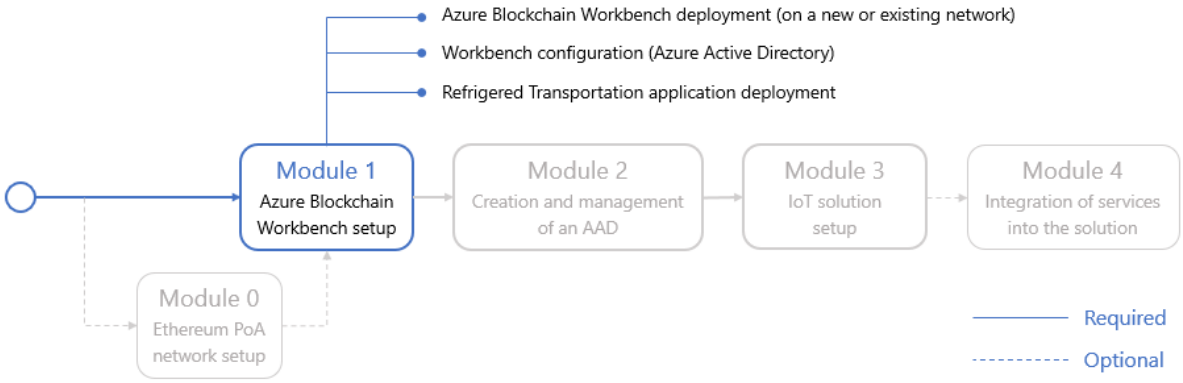
You can also connect existing applications to *Workbench API*, *Workbench SQL Database* or *Egress Queue*, which allows your applications to request *Workbench* data.

The following architecture scheme represents *Azure Blockchain Workbench*, which is 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 infrastructure, key management, …

You can focus on creating new decentralized applications.



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.

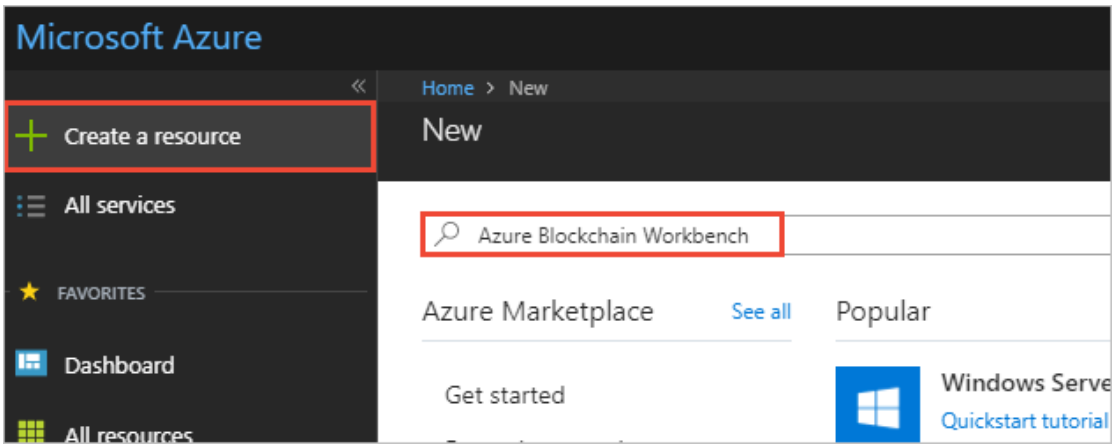
Creating an Azure Blockchain Workbench

First step

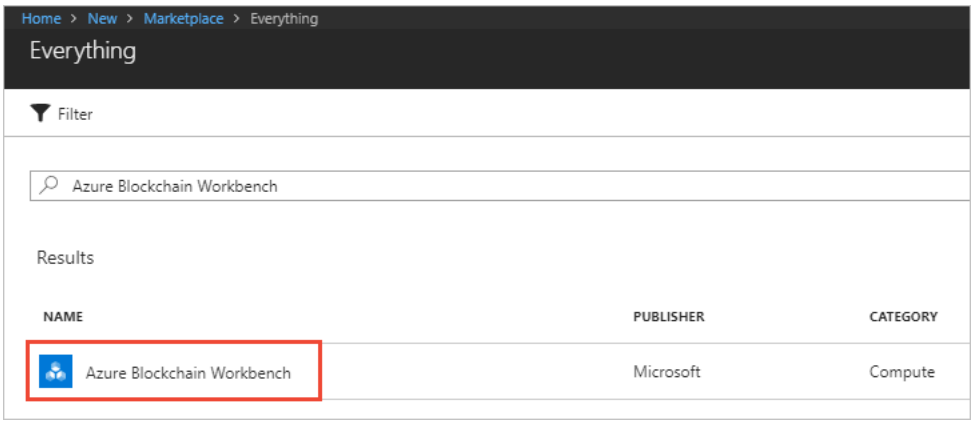
The first step is to log into the Azure portal to create an *Azure Blockchain Workbench*.

Note: The whole documentation about Azure Blockchain Workbench is available [at this link](#) if needed. This guide is inspired by this documentation and summarizes it.

1. Sign in to the [Azure portal](#).
2. Select your account in the top-right corner, and switch to the desired Azure AD tenant where you want to deploy *Azure Blockchain Workbench*.
3. In the left pane, select **Create a resource**. Search for *Azure Blockchain Workbench* in the **Search the Marketplace** search bar.

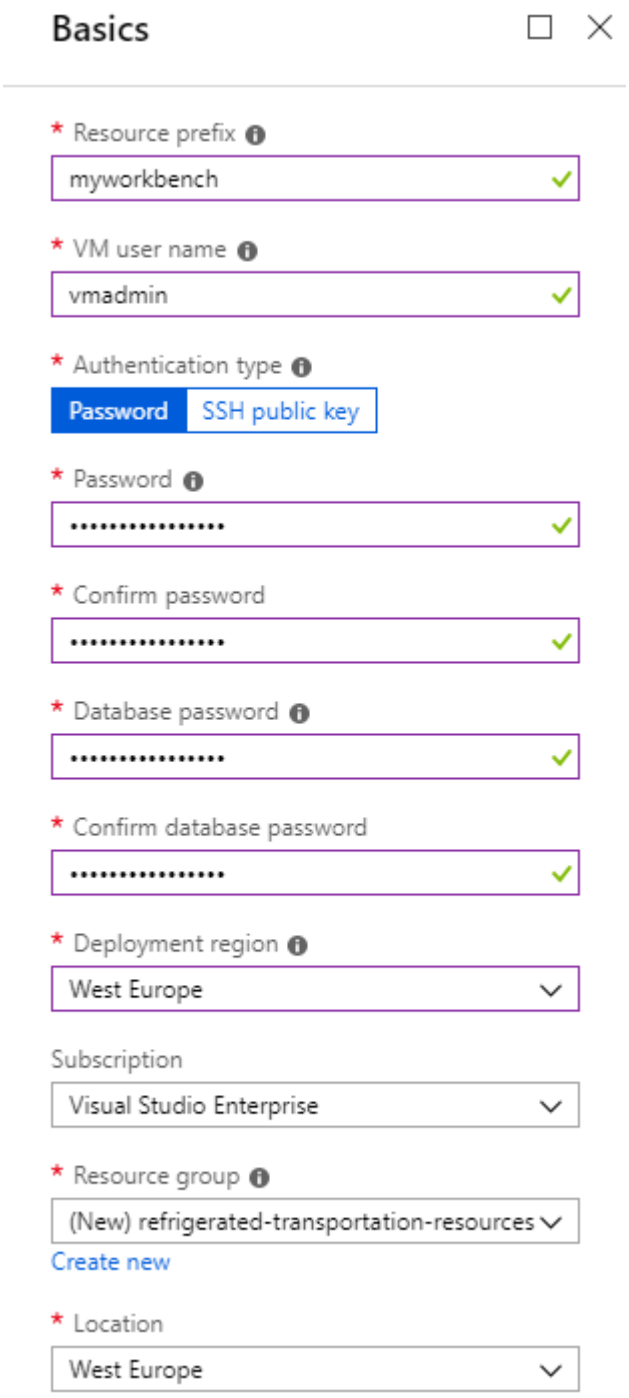


4. Select **Azure Blockchain Workbench**.



5. Select **Create**.

6. Complete the basic settings.



Setting	Description
Resource prefix	Short unique identifier for your deployment. This value is used as a base for naming resources.
VM user name	The user name is used as an administrator for all virtual machines (VM).
Authentication type	Select if you want to use a password or key for connecting to VMs.

Setting	Description
Password	The password is used for connecting to VMs.
SSH	Use an RSA public key in the single-line format beginning with **ssh-rsa** or use the multi-line PEM format. You can generate SSH keys using ssh-keygen on Linux and OS X, or by using PuTTYGen on Windows. More information on SSH keys, see How to use SSH keys with Windows on Azure.
Database password / Confirm database password	Specify the password to use for access to the database created as part of the 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 you wish to deploy the framework.

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

8. Advanced Settings

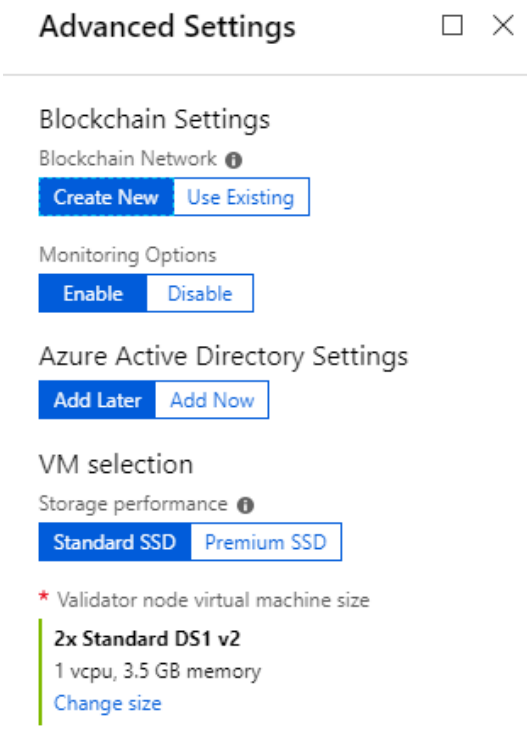
   To work, an *Azure Blockchain Workbench* need to rely on an *Ethereum PoA (Proof-of-Authority)* network. If you don't already have one, you can choose to create a Workbench on a new blockchain network by choosing the option **Create new** in *Advanced Settings* and follow the next point *Deploy on a new blockchain network*. It will create a set of Ethereum PoA nodes within a single member's subscription.

   But if you already created a network, you can choose **Use Existing** and skip the next point to read *Deploy over a pre-existing Ethereum PoA blockchain* 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.

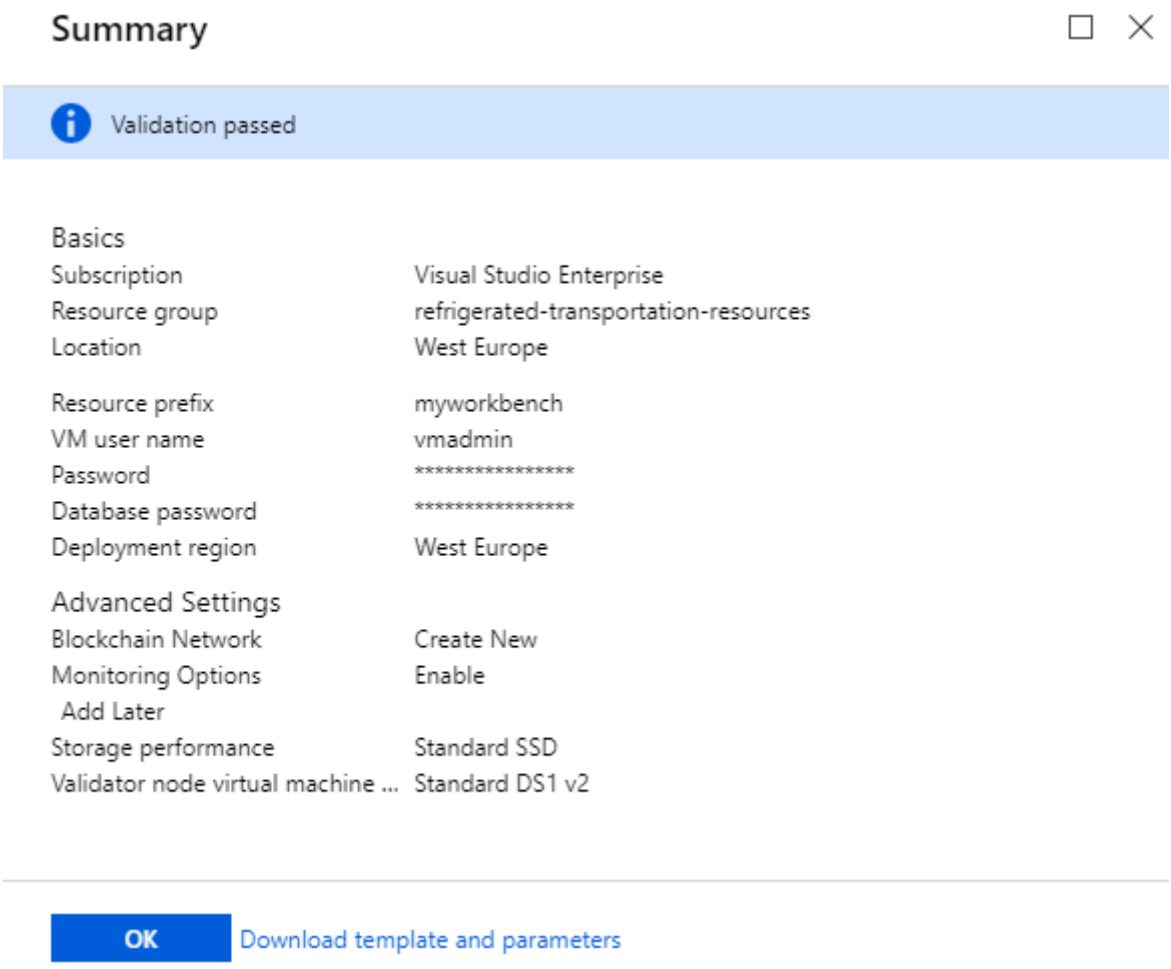
Deploy on a new blockchain network

The *Create New* option creates a set of Ethereum PoA nodes within a single member's subscription.



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 Azure free tier.

Select OK to finish *Advanced Settings*, then review the summary to verify your parameters are accurate and click OK to launch the deployment.

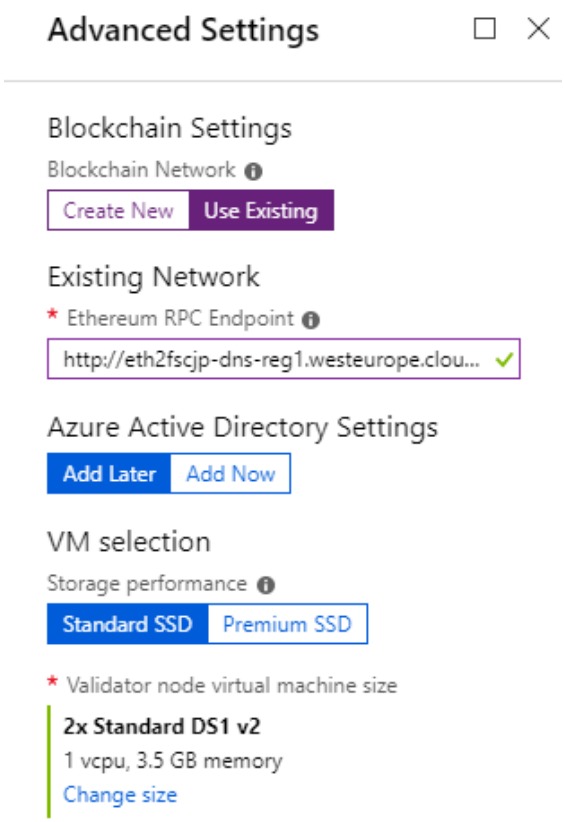


Deploy over a pre-existing Ethereum PoA blockchain

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

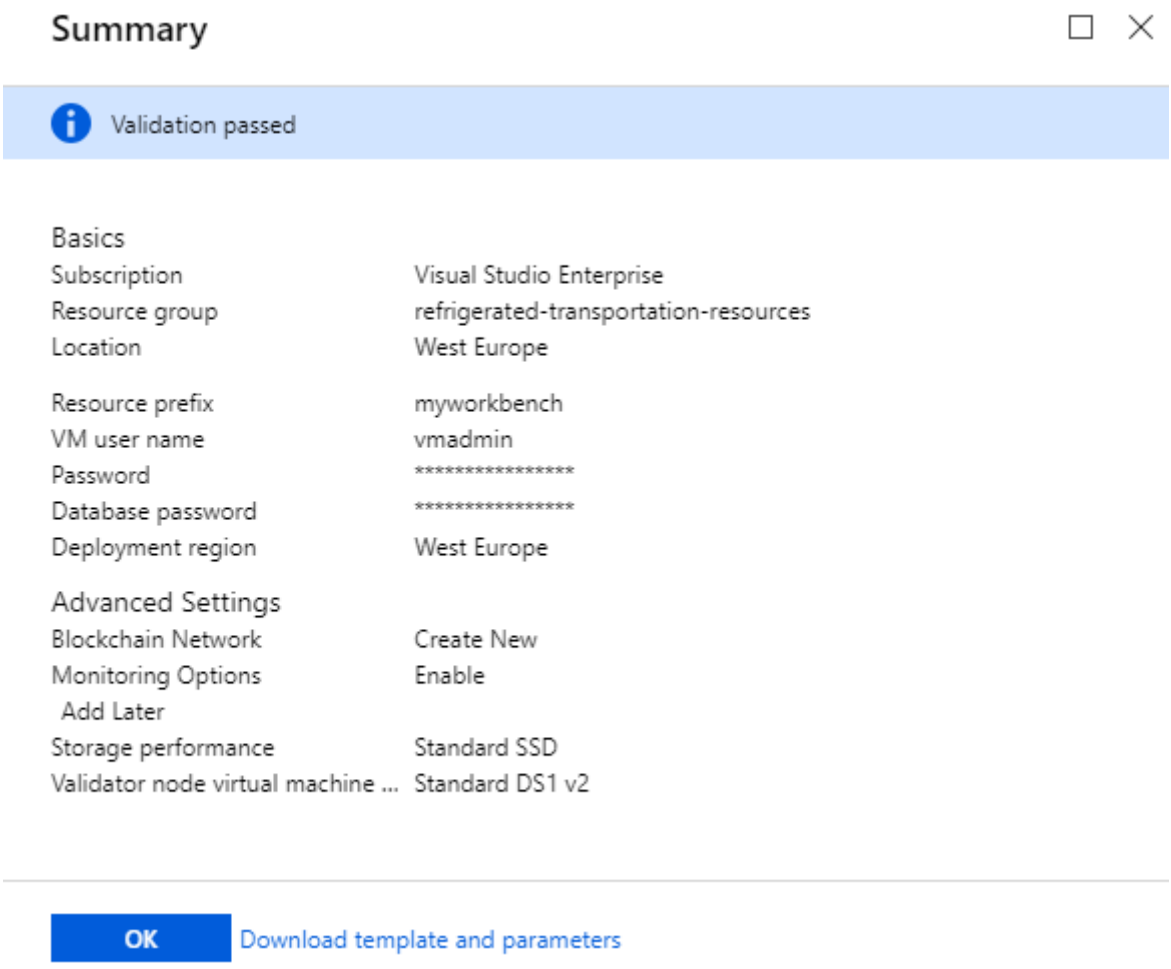
If you created an Ethereum Proof-of-Authority Consortium network on Azure before, you can use it here. You can also use your own external Ethereum network if it follows the following requirements.

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



Setting	**Description**
Existing network	URL of your Ethereum RPC Endpoint. If there isn't any port at the end of the URL, 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 Azure free tier.

Select OK to finish *Advanced Settings*, then review the summary to verify your parameters are accurate and click OK to launch the deployment.



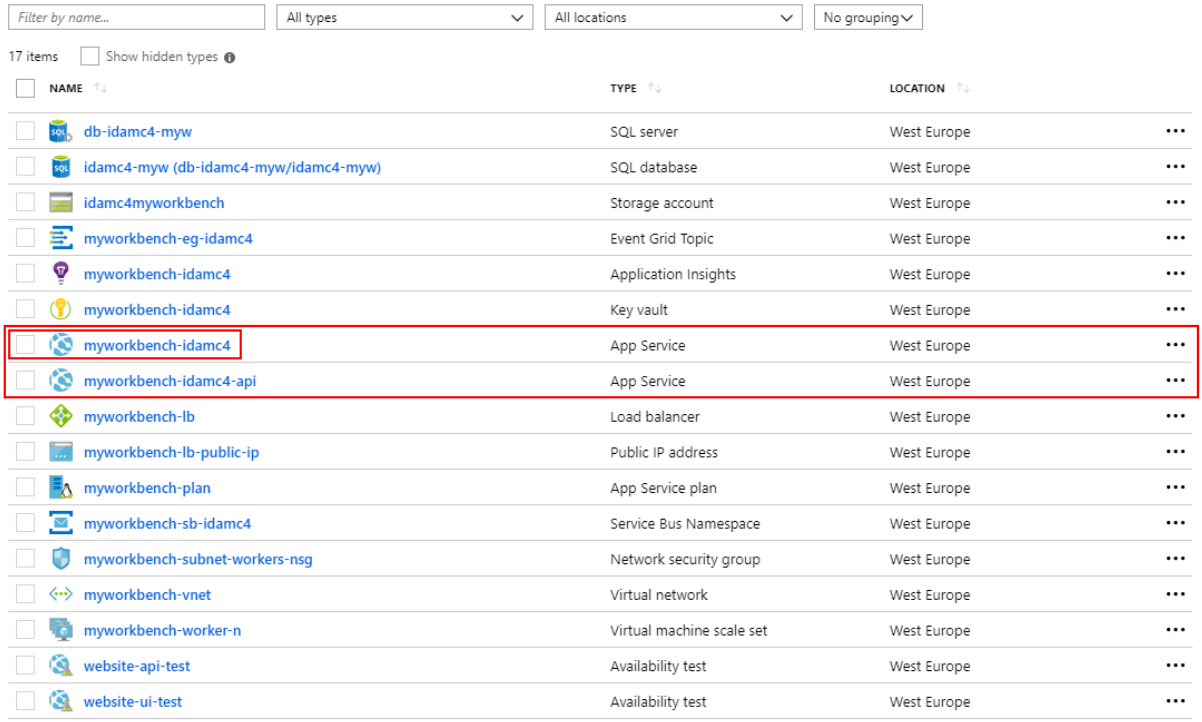
Configuring your Azure Blockchain Workbench

Link your Workbench to your AAD tenant

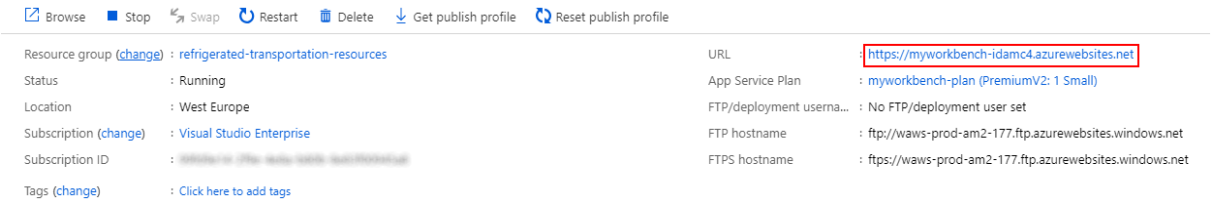
1. Find your Workbench URL

   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 App service resource finish by "-api", and it 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* app

   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 directory to *Azure Blockchain Workbench*. When it will be done, you will have the possibility 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 the *Shell* will open 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

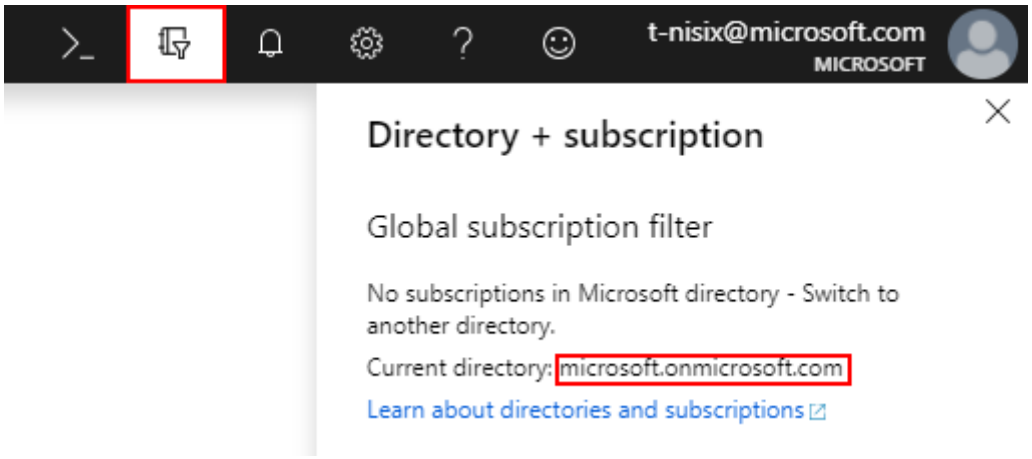
Azure Cloud Shell | Copy | Launch Cloud Shell

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

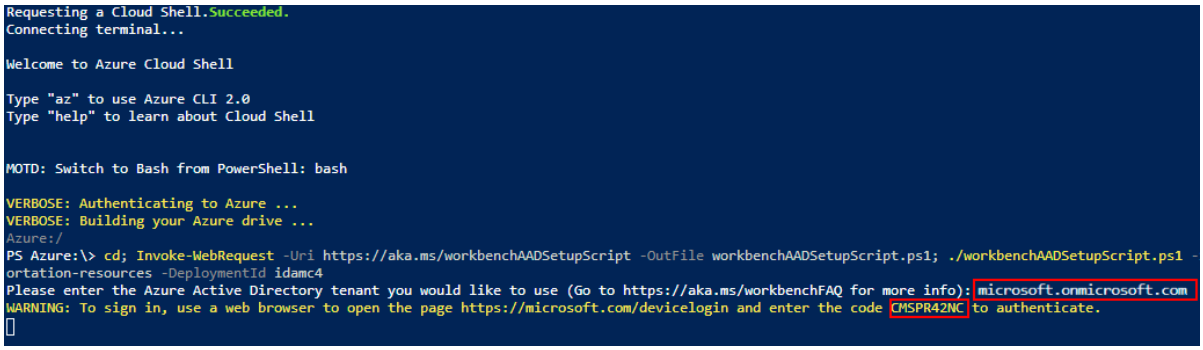
Step 2. Refresh your browser

This will open your Workbench

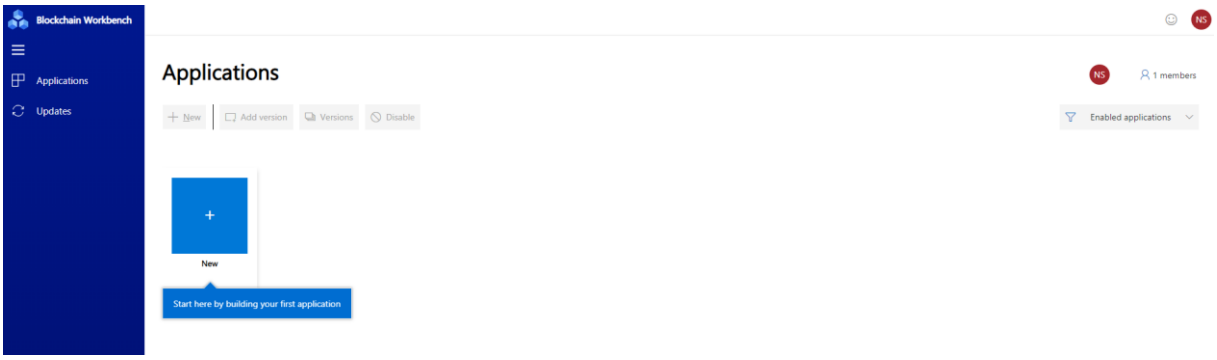
It will prompt you for a tenant name. You can find it by opening a new Azure tab, then clicking on the **Repertory icon** at the top of the window.



Enter this tenant name on the *Shell* and press **Enter**. The shell will ask you to authenticate into *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*, and now you can go back on your workbench then refresh it to finish the deployment.

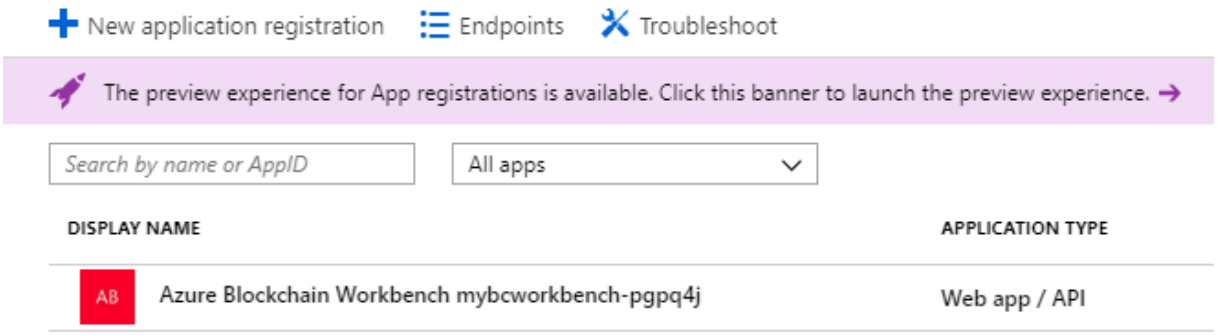


Auto-assign Administrator role to yourself

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

To fix that, follow the next steps.

1. Go into the AAD that you linked with the Workbench and select **App registrations,** then 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.



2. A tab with information about Workbench will appear, click on the link below *Managed application in local directory*.

3. You are now in the *Identity and Access Management* section of the application. Click on **Users** and groups, then click on **+ Add user**. Add yourself as an *Administrator*, then save. Disconnect and reconnect yourself on the *Azure Blockchain Workbench* and you should now be an Administrator.

   Note: Keep in mind this way of adding Administrators: you could potentially use it to add new Workbench Administrators.

Deploying the supply chain of refrigerated products application

Now that your *Azure Blockchain Workbench* is finally online, it's time to deploy your first application on it!

1. Get application files

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

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

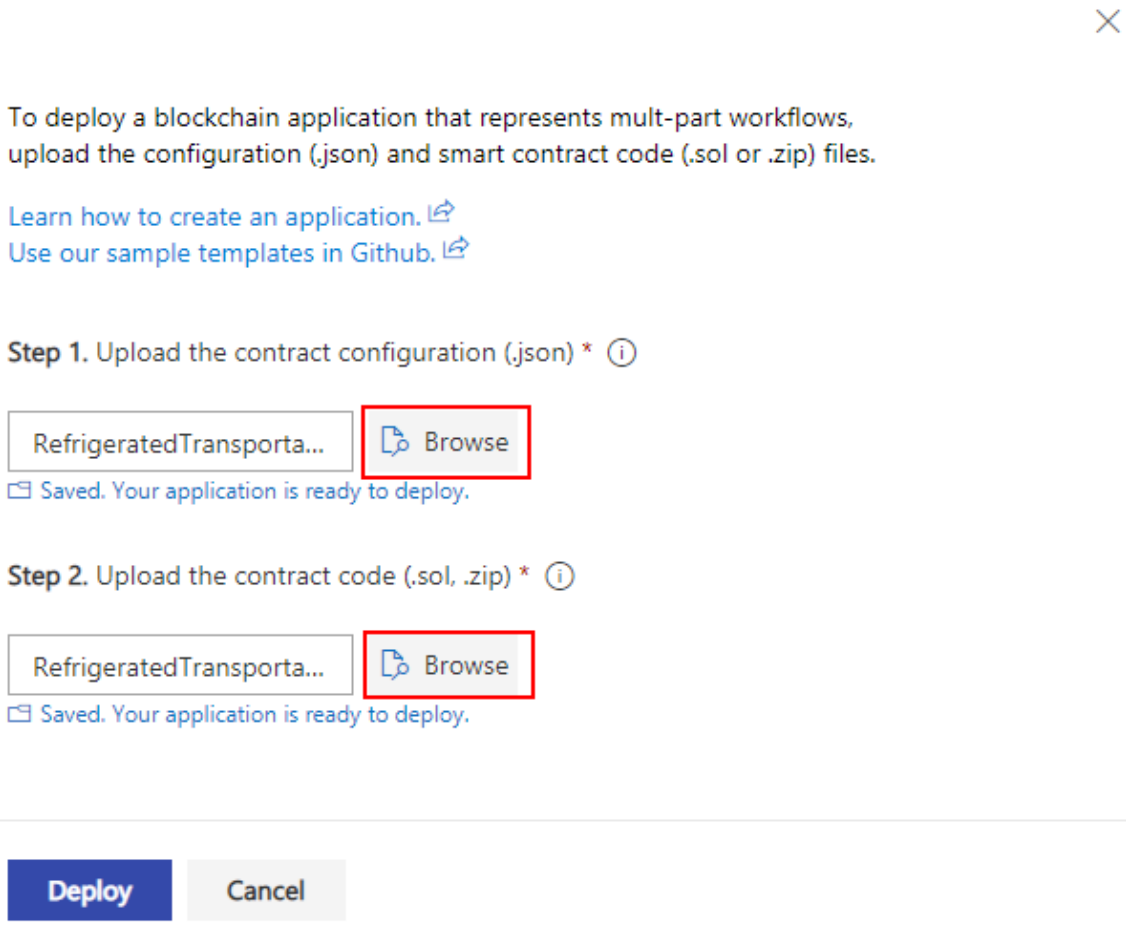
   For this solution, we will use the *Refrigerated Transportation Sample Application* for *Azure Blockchain Workbench*, available on Github.

   You can download the files below.

- Configuration file
- Smart contract code file

2. Deploy your application

   Go back to your workbench, then click on **New** to create your first application. You will need your two files here: the first one which will be used is the Configuration file. Click on Browse, then select your file on your computer. Then, it will ask you for the *Smart contract* code file: do the same.



Note: Each time you will 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 went well, your application is now deployed, and you can access it by clicking on its icon. But at the moment, you are still the only one which has access to the workbench and the application.

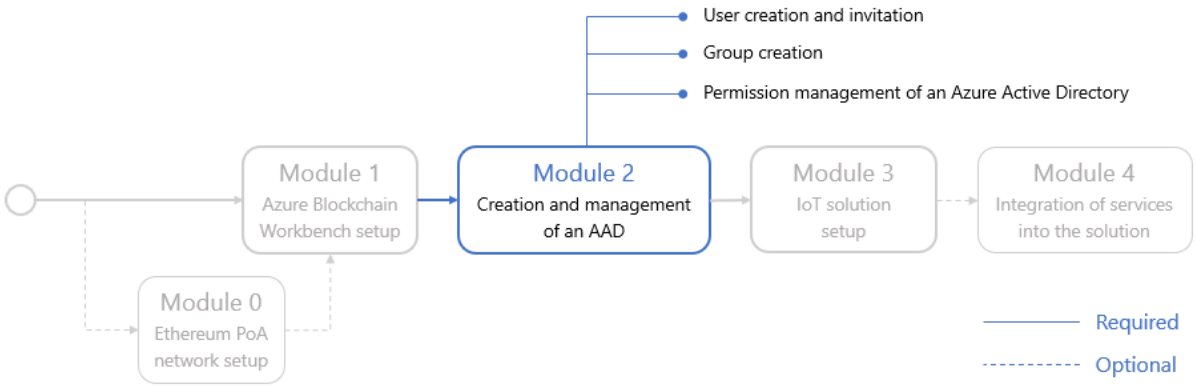
The main objective of this guide is to set up an application shared by organizations and users, and in the next part of it you will see how to create organization inside your *Azure AD*, and by extension your application.

Module 2 : Creation and management of an Azure Active Directory

Overview

An *Azure Blockchain Workbench* is always linked to an *Azure Active Directory*. This feature allows user management within *Azure*, so an Administrator doesn't have to bother about key management or user identities on the Workbench. Indeed, everything is done directly on *AAD* and an Administrator can just click on *+Add* on a Workbench Application to add new users (if they exist inside the *AAD*).

Our objective here will be to set up an *AAD* which represents a pseudo-consortium between two companies, *Contoso* and *Woodgrove*.



In this part, you will learn to:

- Create a user within the AAD portal (you will get a user account and his temporary password).
- Invite an external user into AAD.
- Create groups and assing user into them.
- Manage Workbench permissions (Workbench user, application user, Workbench administrator).

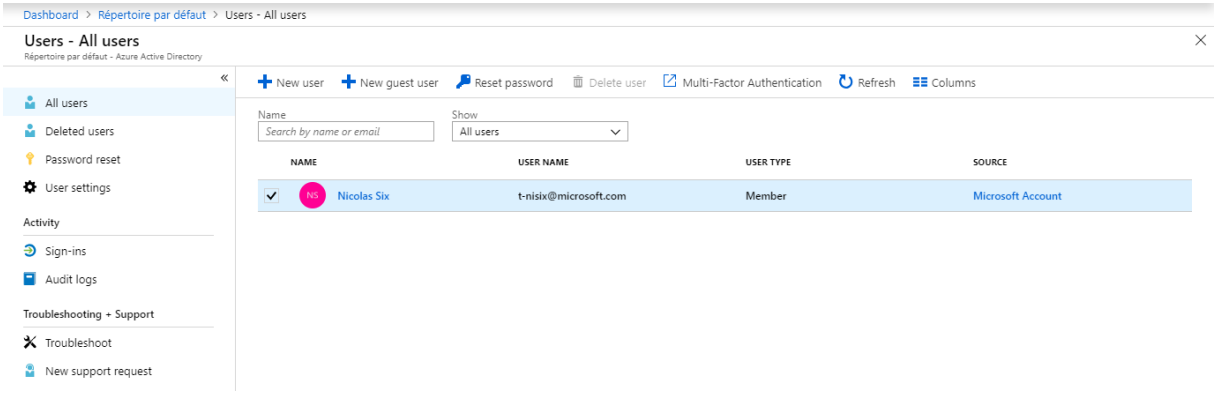
Adding new users in your AAD

Create a user in AAD

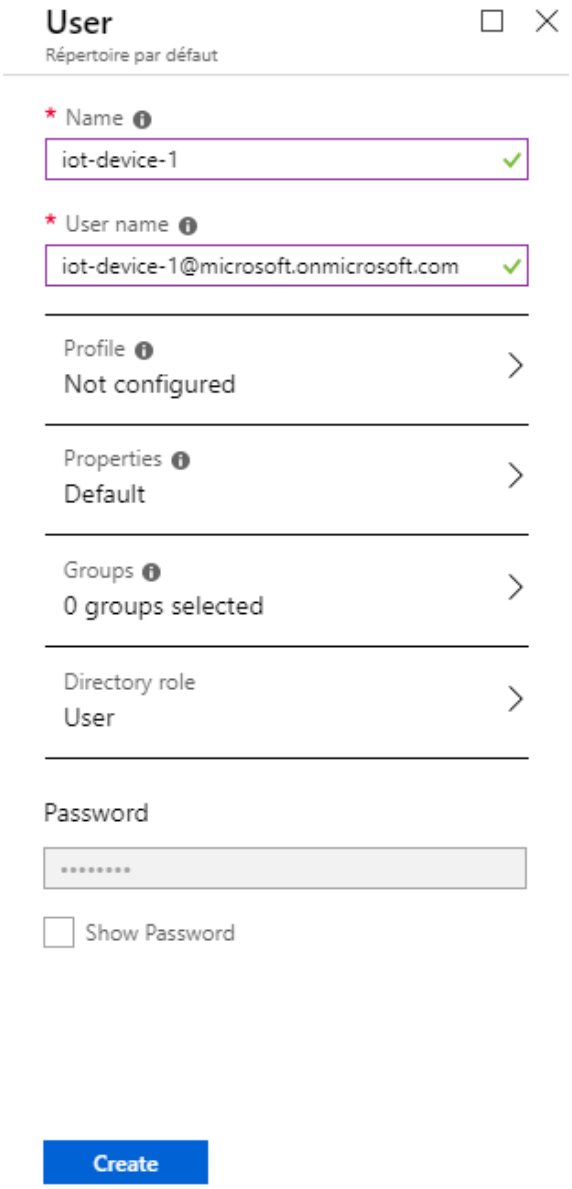
First of all, we're going to create users directly in our *AAD administration panel*. This will be convenient for creating our IoT devices identities for this guide.

Note: For a real production scenario, there is a possibility to create bulk identities for IoT devices, but we're going to create only 4 IoT devices in AAD for this guide so this is not necessary in this case.

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

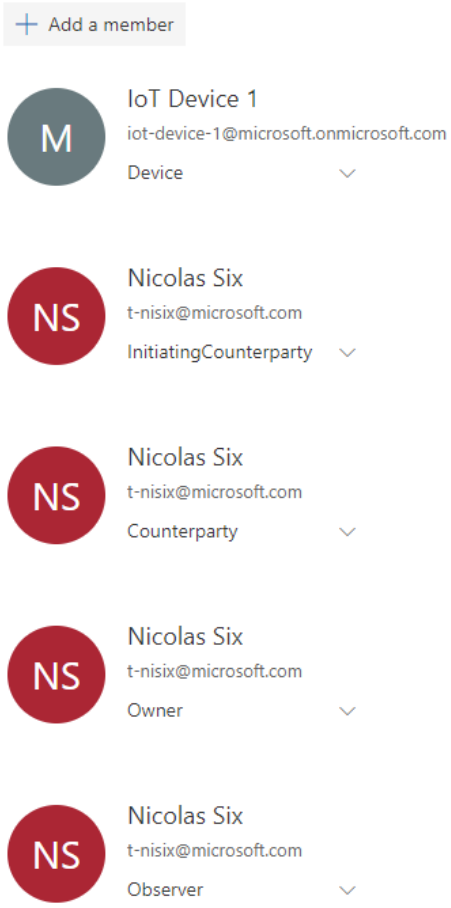


2. Click on **+New User**. Fill the form with accurate information.



Setting	Description
Name	The name of your new "user".
User name	The email address of the new "user". It must finishes by @<your tenant name>.onmicrosoft.com.
Profile	Click if you want to fill more information about the "user" (for example, 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 we're going to create and assign groups later.
Directory role	Let it by default, as we don't want to 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. We don't need it for IoT devices (and if you want, you can still get recover it on your AAD panel) but copy it if you want to create a new user. For a real user, you would have to provide him the password but for this demonstration, we will use it to log into the Workbench as if we were a real user.

3. The user profile is now created. Now, they can be added inside your *Workbench* application. Log 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 4 times yourself as all the different roles except *Device* and add your IoT device as the *Device*.

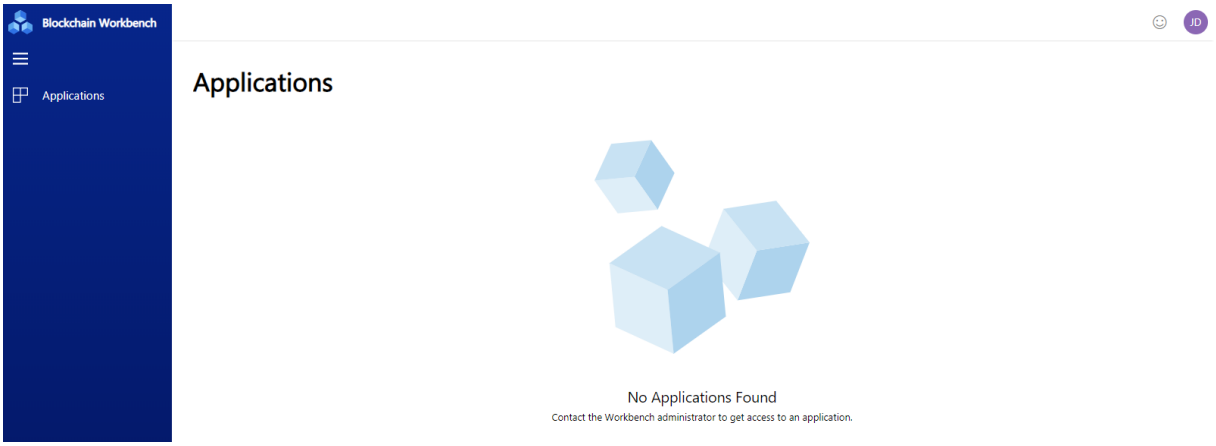


4. Repeat the process by adding 3 more IoT devices: *iot-device-2*, *iot-device-3,* and *iot-device-4*. We are going to use them later when we will implement our IoT architecture. Add also an account which will represent a user created by this method, you can give him the name you want, but here, I will choose *John Doe*.

To try the user account, we're going to log in with it on *Azure Blockchain Workbench*. Open a Workbench tab (preferably in a Private Browser), it will ask you to log in. Fill the email with the *User name* that you gave while creating the user, then the *temporary password* as the password.

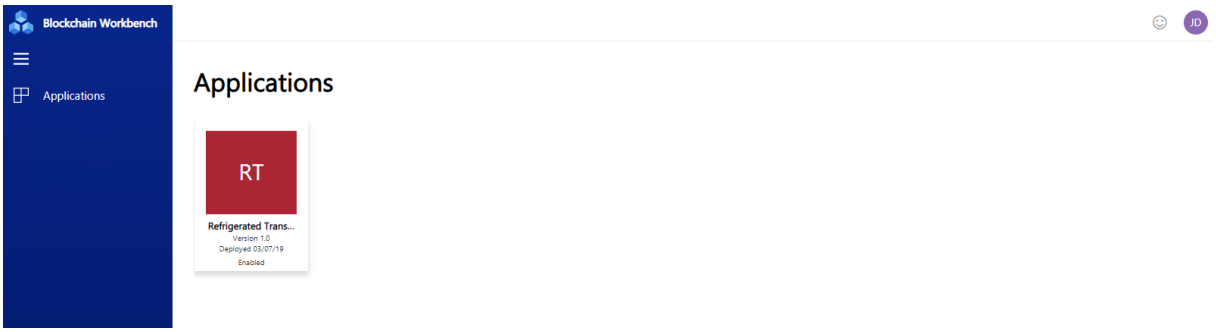
On the next tab, you will have to change that password, do it and click on **Connect**.

You are now logged into the Workbench, but you can't see any Application: that's because a user needs to be 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.

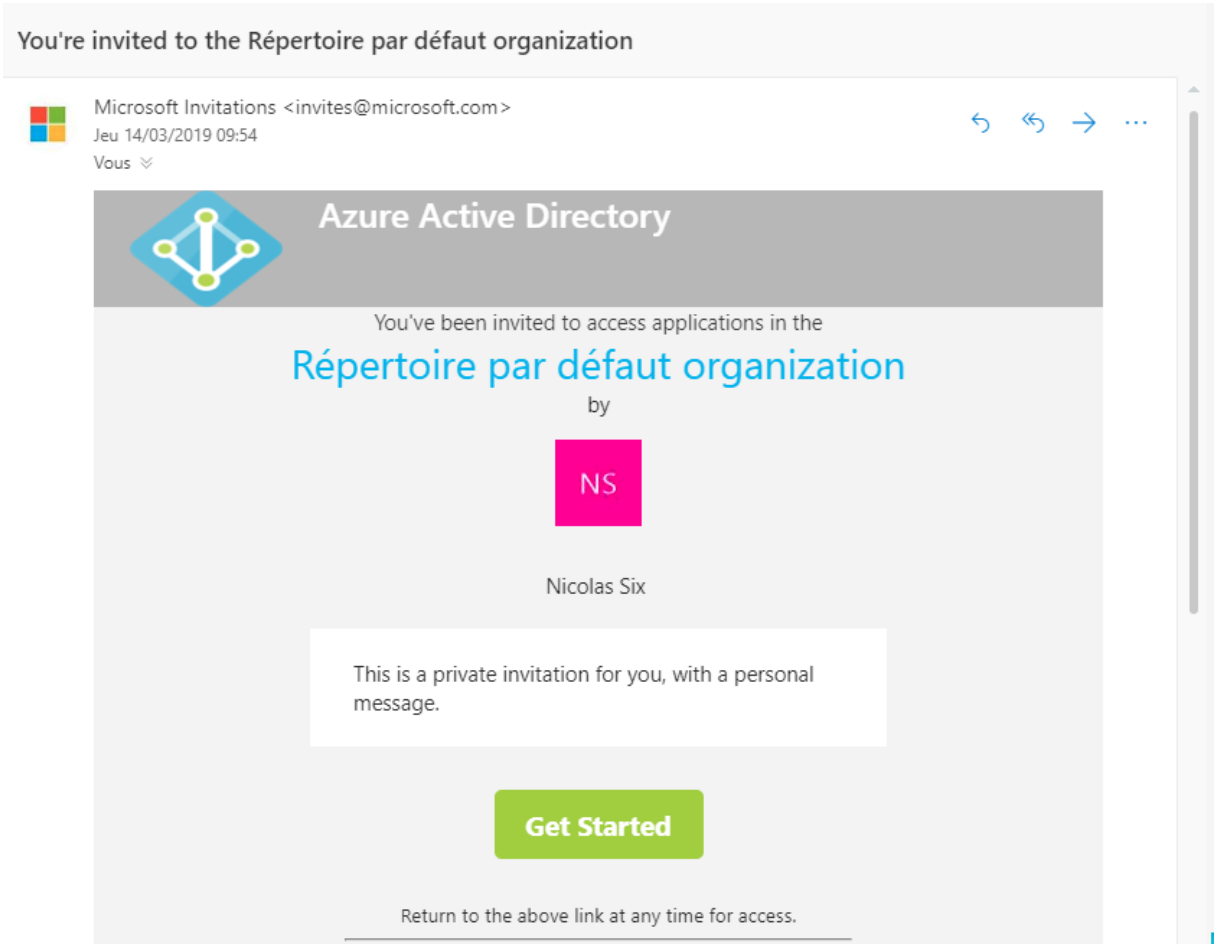


Invite a guest user in your AAD

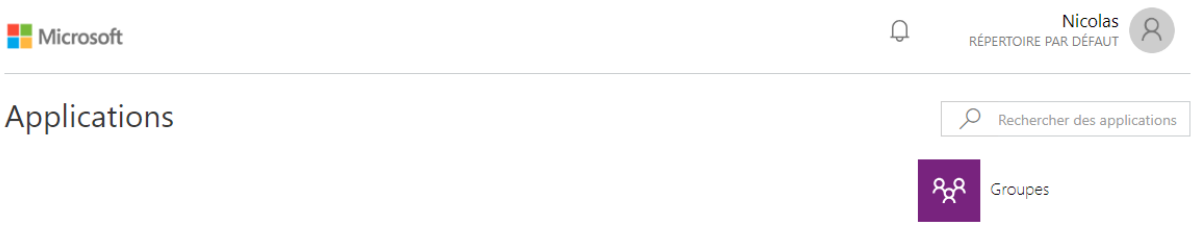
Instead of creating manually new users, you can directly invite people to join your Azure tenant. Therefore, guests will be able to login with their own professional email, and if this email is linked with an *Office 365 subscription*, get an access to the *Workbench* on their *AAD* home page like any other application.

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

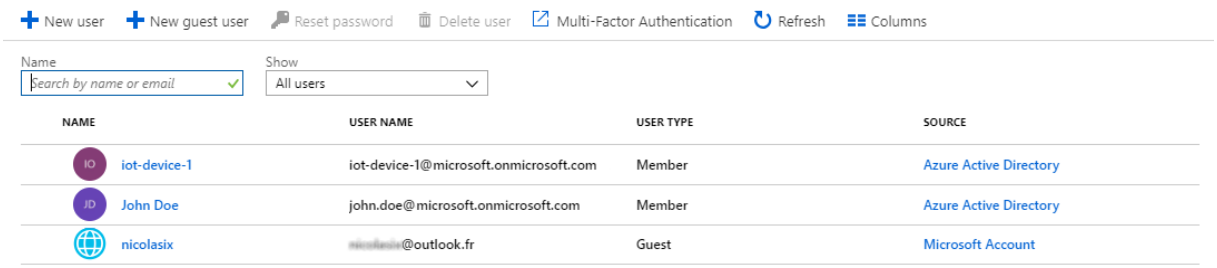
2. Verify that you received the email in your inbox.



If you follow the link, you will have to create a Microsoft account if this email is not linked to one yet, and you will be redirected to your *AAD* home page.



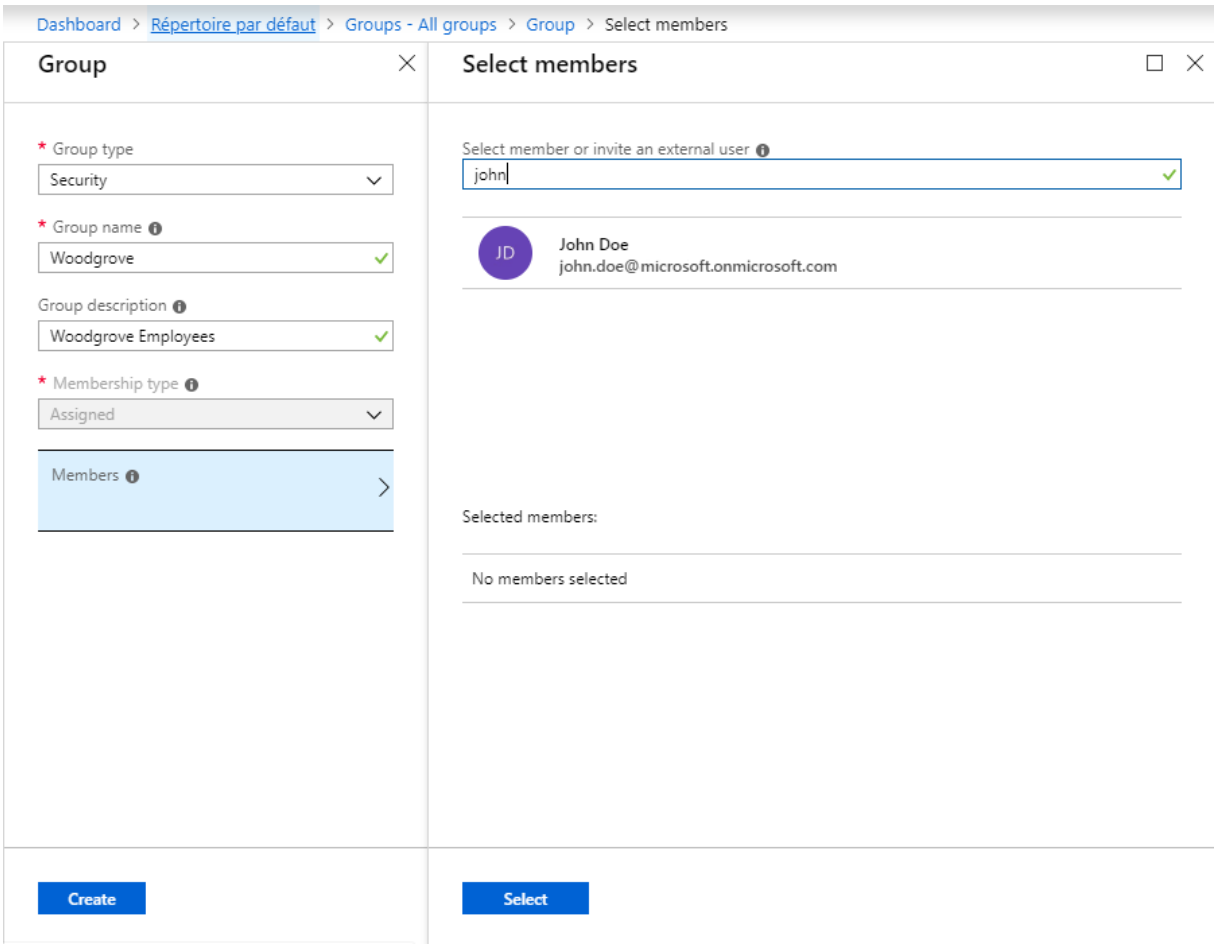
Note: You can see the difference between a regular used (created inside AAD) and a guest user (invited from outside) when looking inside the user panel on AAD).



Adding new groups inside your AAD

Now that we're able to create 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.

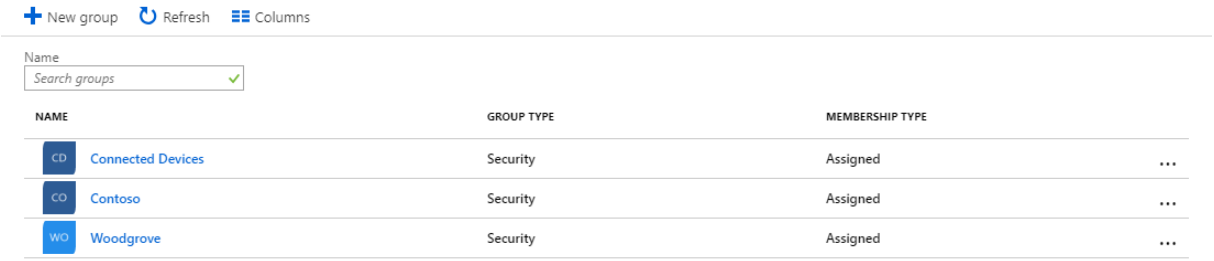
1. Go inside your *AAD*, then click on **Groups**. fxs-draganddrop
2. Once inside the panel, click on **+New group**.
3. Fill the form with accurate information.



Setting	Description
Group type	○ **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 it this way, 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 Manage access to resources with Azure Active Directory groups. ○ **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

Setting	Description
	to the group. For more info about Office 365 Groups, see Learn about Office 365 Groups.
Group name	Name of the new group.
Group description	Description of the new group.
Membership type	Let it by default (Assigned).
Members	Clicking of this field allow you to directly add existing 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.

Your newly created group should appear inside your group list.



For this guide, I've created 3 groups: one for future connected devices, and two for invited employees of their respective company, *Woodgrove* and *Contoso*.

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 were invited by an *Administrator* to join them. Therefore, an *Administrator* can add users to applications and give them defined roles inside them. They can also deploy new applications inside the Workbench.

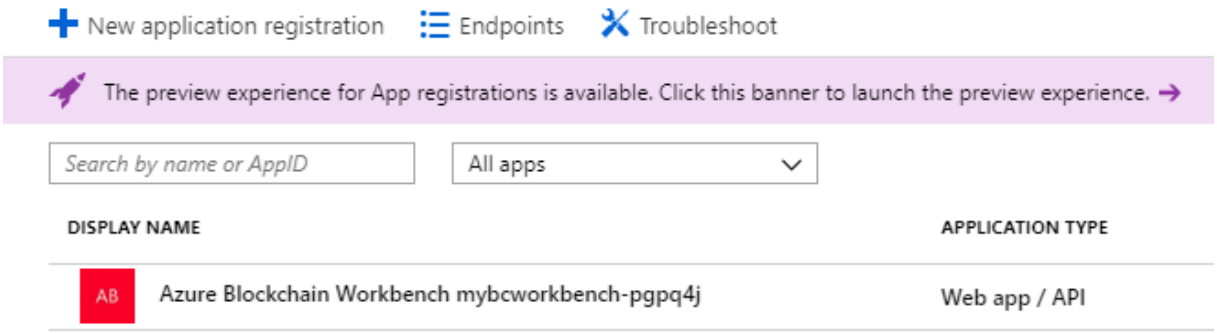
Adding users inside the *Workbench* is easy to do, they only need to be inside the AAD to have an access to the *Workbench*.

But managing *Administrators* is a bit more difficult. In this part, we're going to see how to add new *Administrators*, or groups which users inside it get automatically *Administrator* role.

Workbench roles and permissions

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

1. Go into the AAD that you linked with the Workbench and select **App registrations,** then 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.

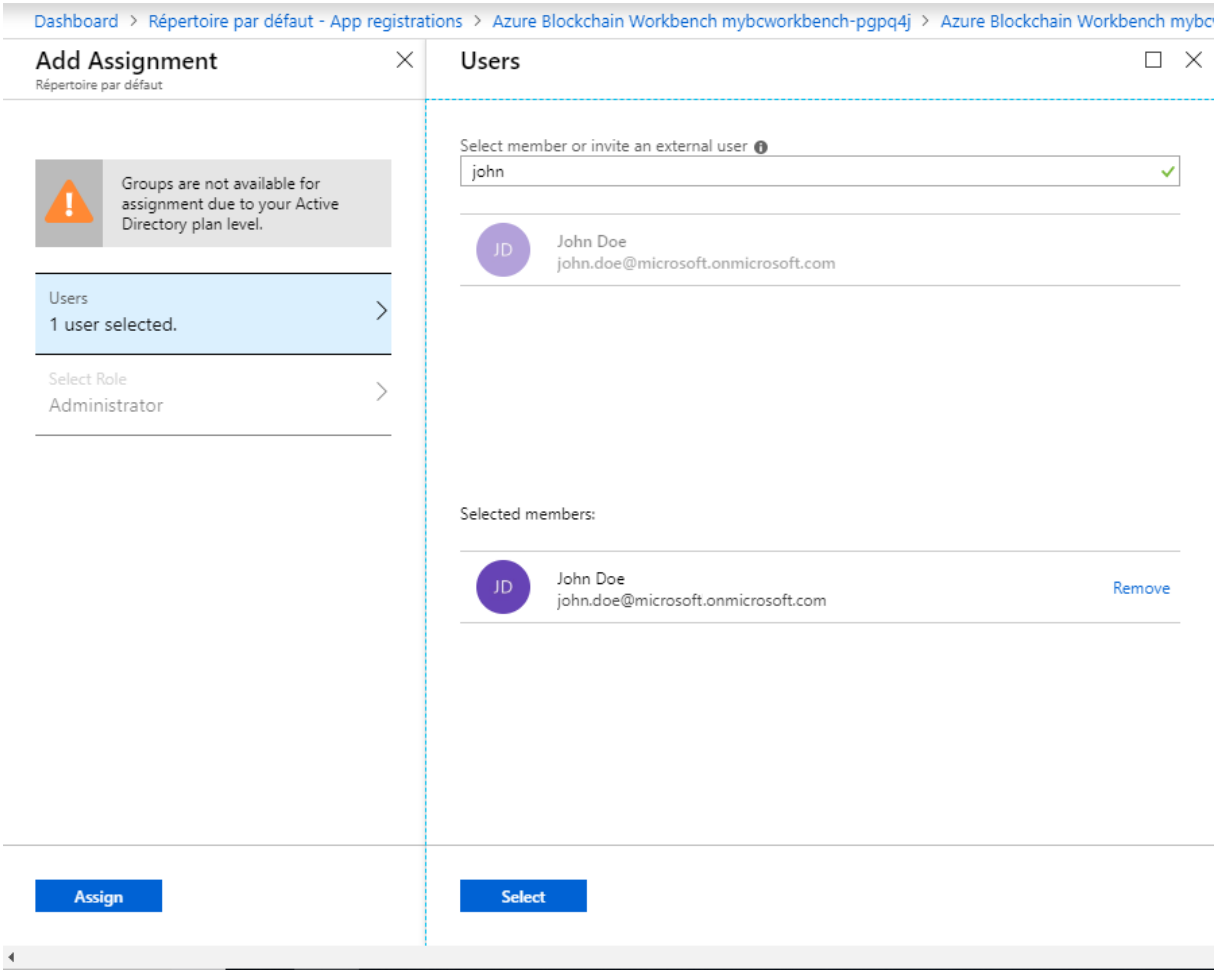


2. A tab with information about Workbench will appear, click on the link below *Managed application in local directory*.

3. 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).

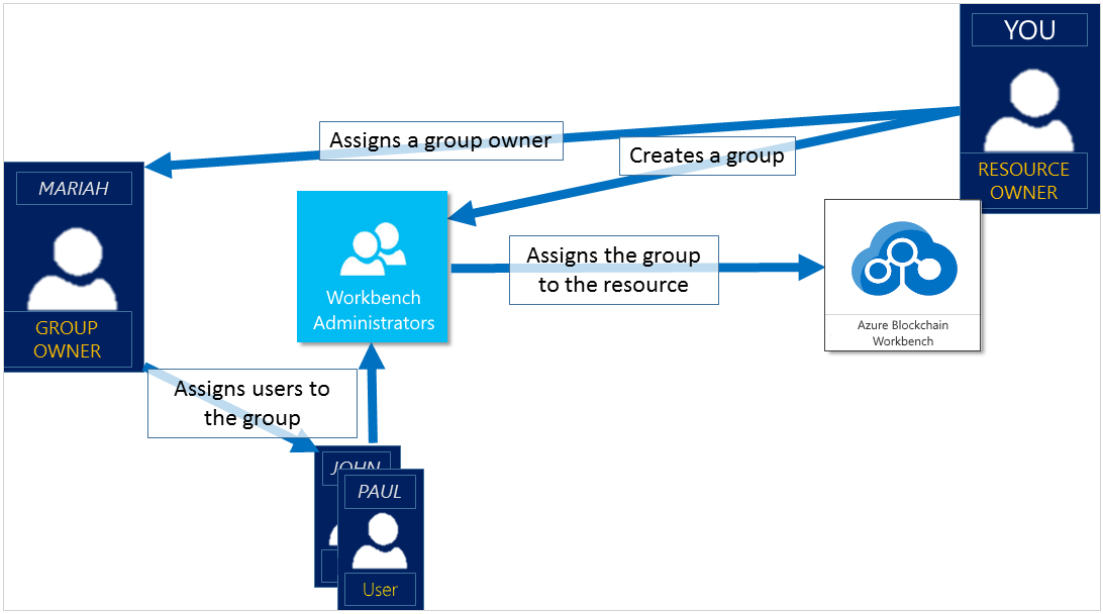
   Search and click the users and the groups that you want to add to the *Workbench*, then click on **Select** and **Assign**.



For a little 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 users into AAD in order to delegate *Workbench* management task.

AAD roles and permissions

In this part, we're going to look how to assign someone as a *Group Owner* in AAD, in order to let them invite users into groups, in order to automatically assign them as *Administrators* or just to regroup users into groups which represents their company.

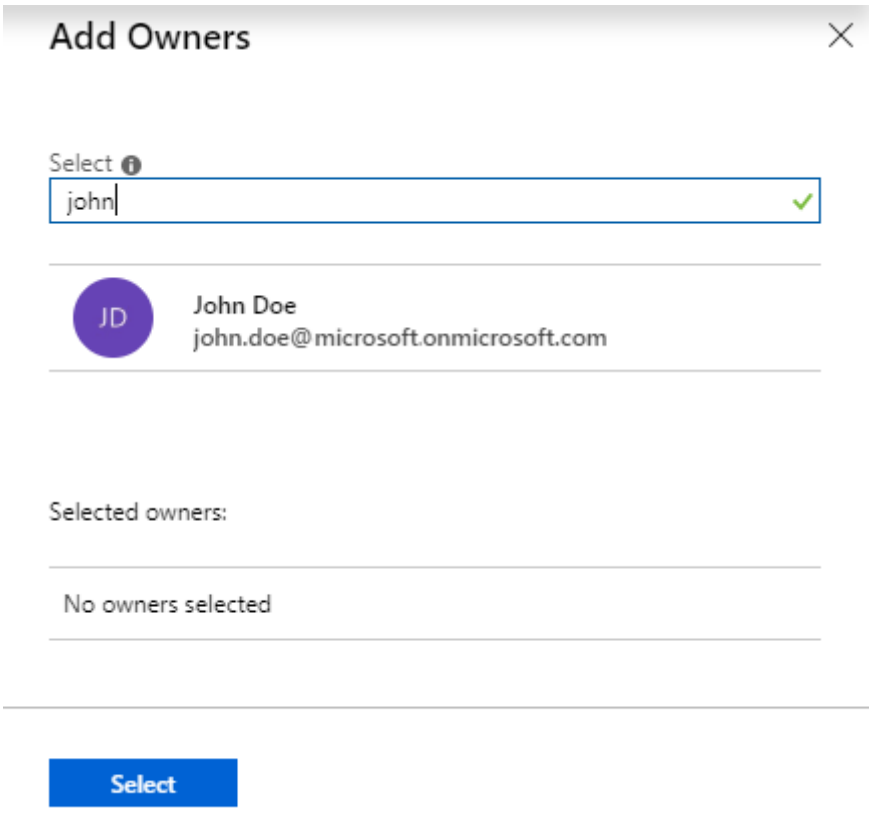


1. Go inside your **Azure Active Directory**, then click on Groups.

2. Select a group which is inside the list. Here, I will select **Woodgrove**.

3. We'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**.

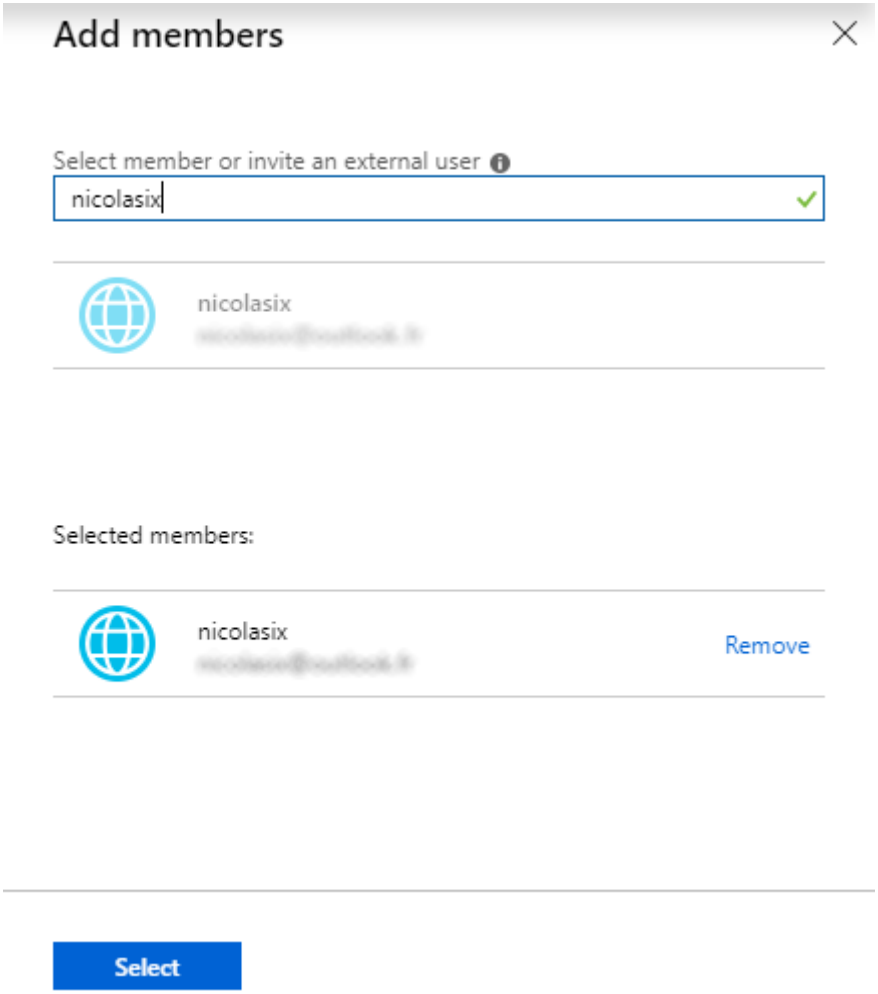
4. Search for the user you want to add as an owner, then click on it and click on **Select**.



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**.

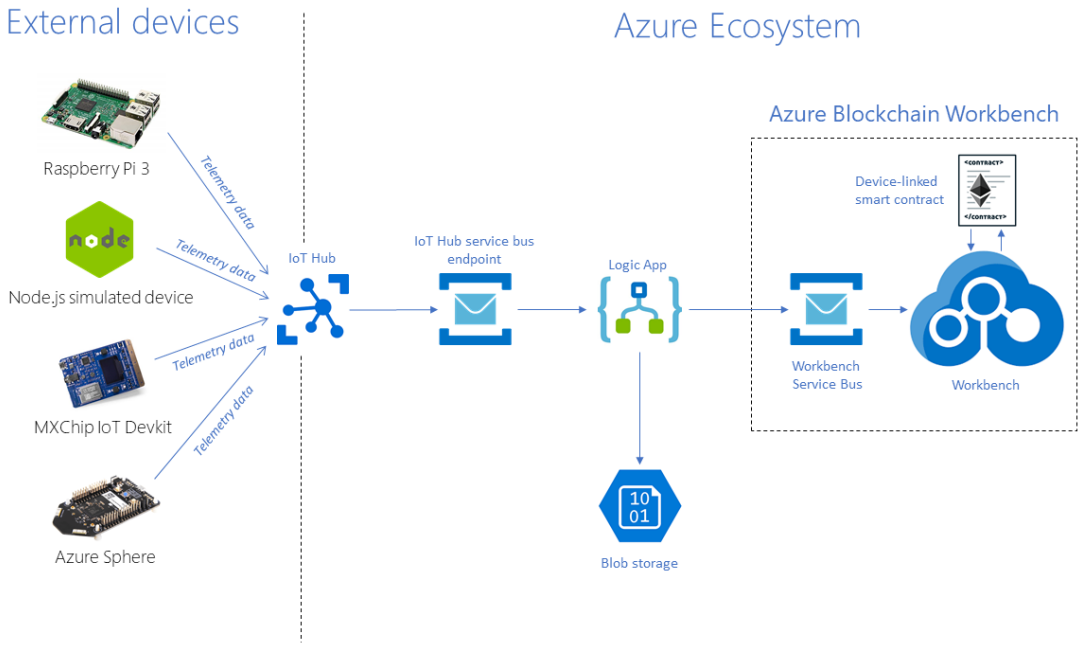


Now, you are able to invite new users into your Workbench and grand them *Administrator* role. If you want to go further with Azure AD, feel free to read the whole documentation here. We're now ready to build our IoT solution.

Module 3 : IoT solution setup

Overview

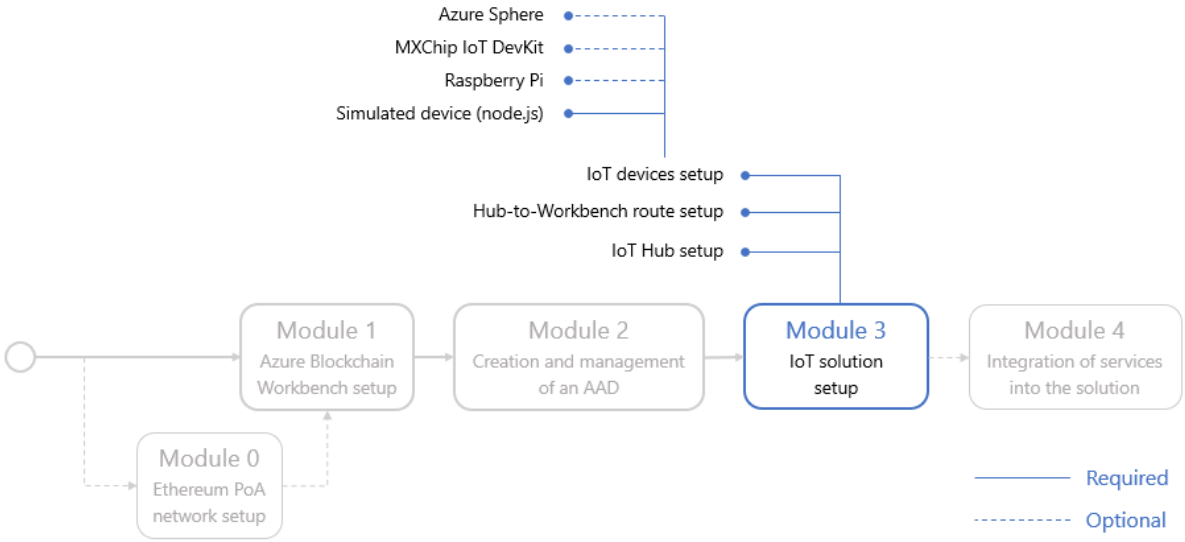
This part is one of the most important in this guide. The goal is to enable communication between an IoT part, which will be composed by some connected device linked to an *IoT Hub*, and the Workbench. This is not the easiest part: we will have to take care of device identity management through our Workbench, as well as message routing. To summarize the solution, let's have a look at this diagram.



We will start from our devices and connect them to the *IoT Hub*.

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

Thanks to that service, we will be able to route the message in *Azure Blockchain Workbench* but also in blob storage which will allow us to build graphics with *Power BI* later on.



On this part, you will learn to:

- Deploy an IoT architecture, composed of an IoT Hub, a Service Bus/Queue, a Logic App and connected devices which is capable of routing messages from devices to Workbench.

Let's begin with our first component: *an IoT Hub*.

Setup of the IoT Hub

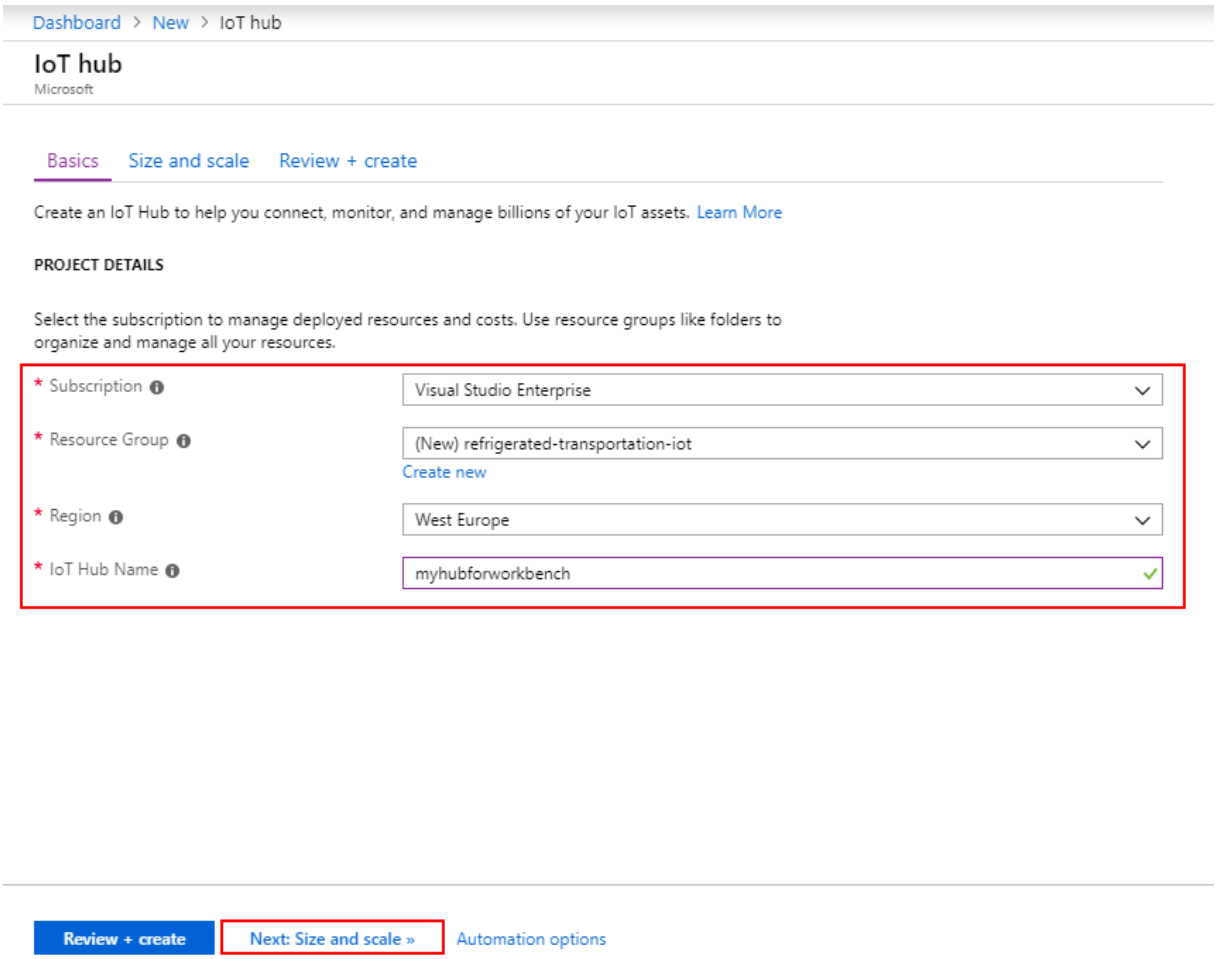
This section describes how to create an IoT hub using the Azure portal. The Hub will allow our connected devices to communicate with *Azure Blockchain Workbench*.

Note: The whole documentation about Azure IoT Hub is available at this link if needed. This guide is inspired by this documentation and summarizes it.

1. Log in to the Azure portal.

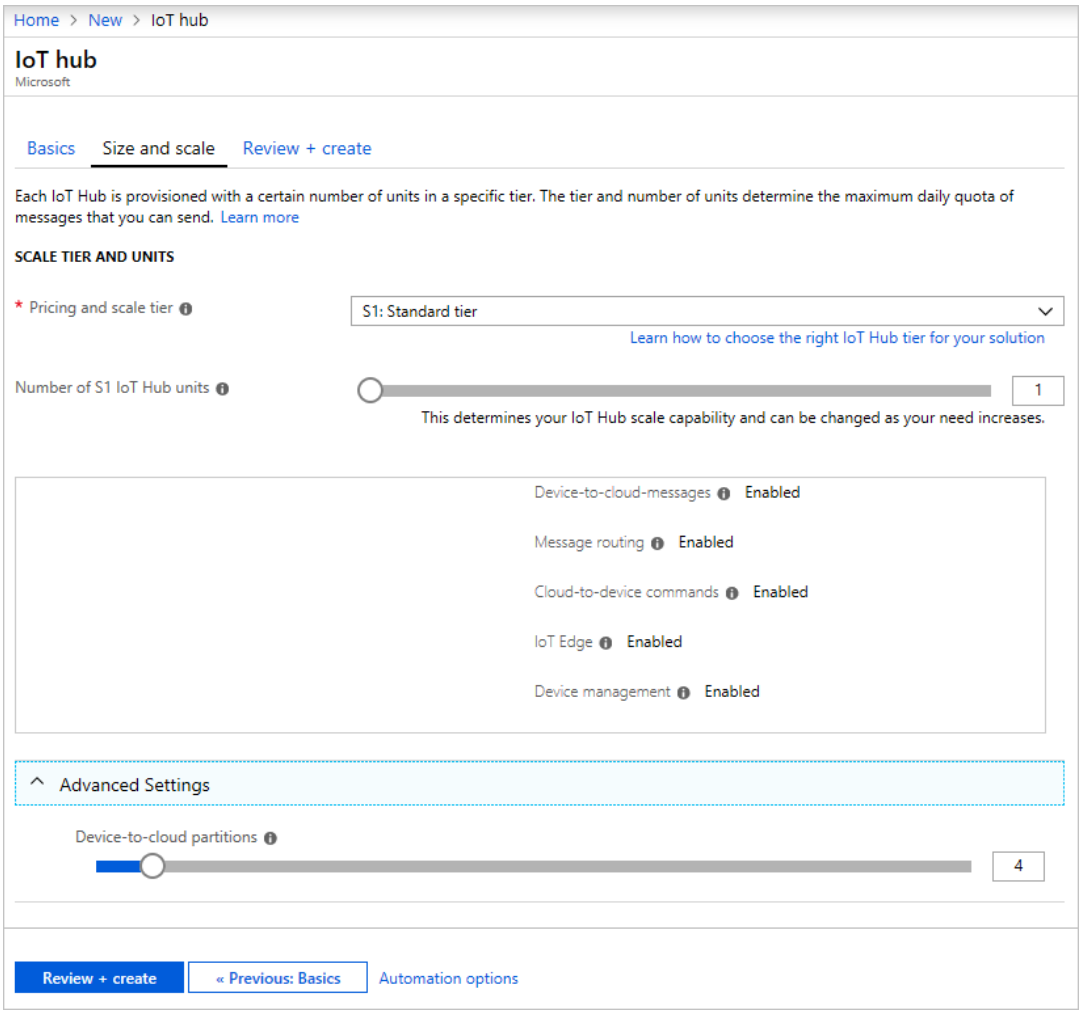
2. Choose +**Create a resource**, then choose **Internet of Things**.

3. Click **IoT Hub** from the list on the right. You see the first screen for creating an *IoT Hub*.



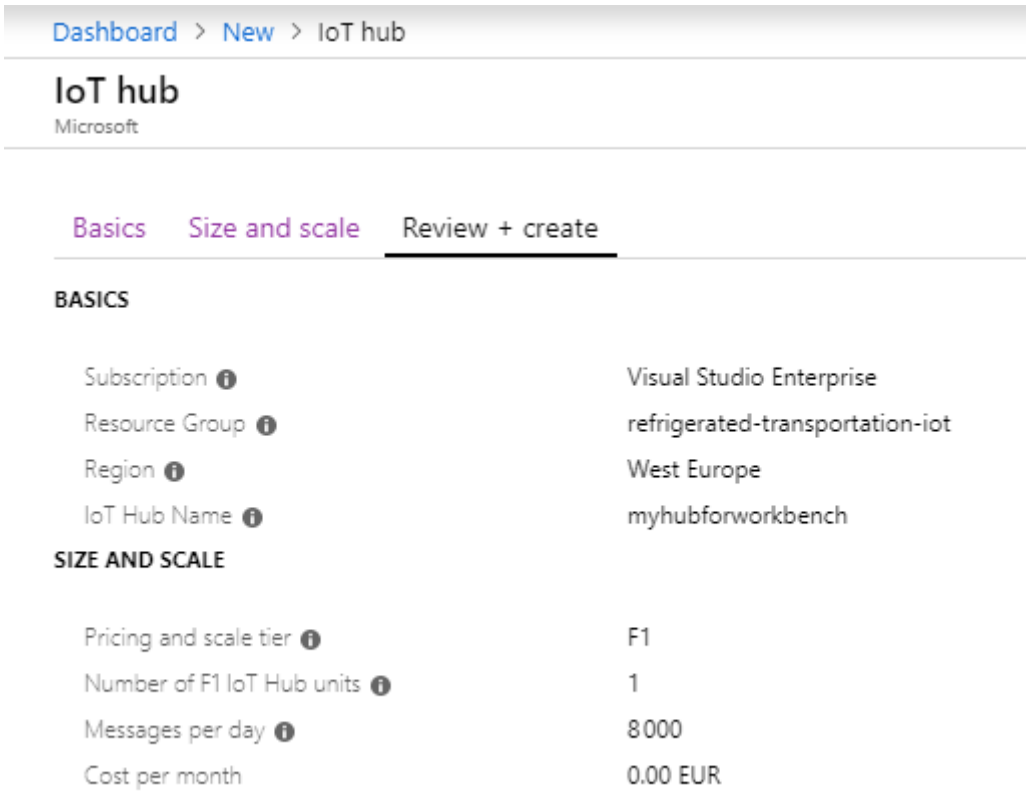
Setting	Description
Subscription	Select the subscription to use for your IoT hub.
Resource Group	You can create a new resource group or use an existing one. To create a new one, click **Create** new and fill in the name you want to use. To use an existing resource group, click **Use existing** and select the resource group from the dropdown list.
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	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.
IoT Hub unit	The number of messages allowed per unit per day depends on your hub's pricing tier. We don't need more than one unit for this guide.
Device-to-cloud partition	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 see something similar to this screen.



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

Routing messages from the Hub to the Workbench

Setup a Service bus for message routing

Before connecting new devices to the Hub, we need to set up message routing from the IoT hub to suitable services. All the messages will be stored into blob storage for analysis with Power BI and ingested by the Workbench. If temperature or 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 this tutorial, available on Github at this link.

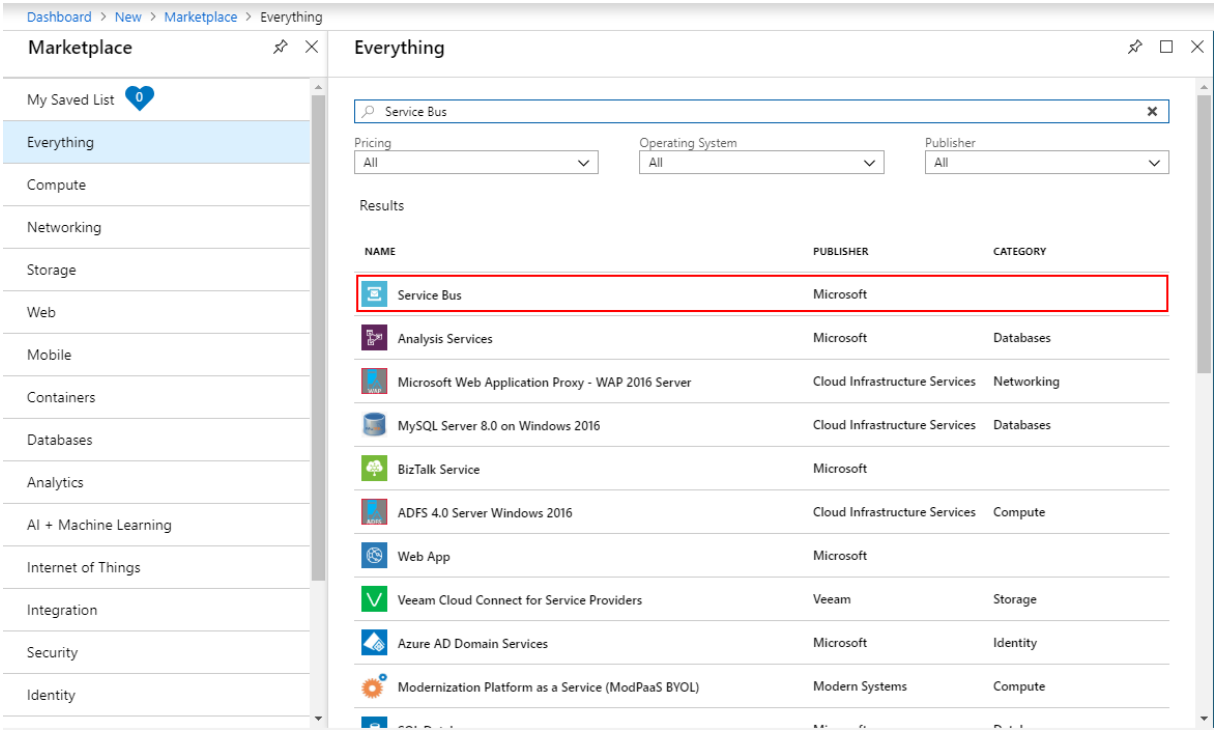
First of all, we are going to create the route between the *IoT Hub* and *Azure Blockchain Workbench*. To do that, we will create a *Service Bus*.

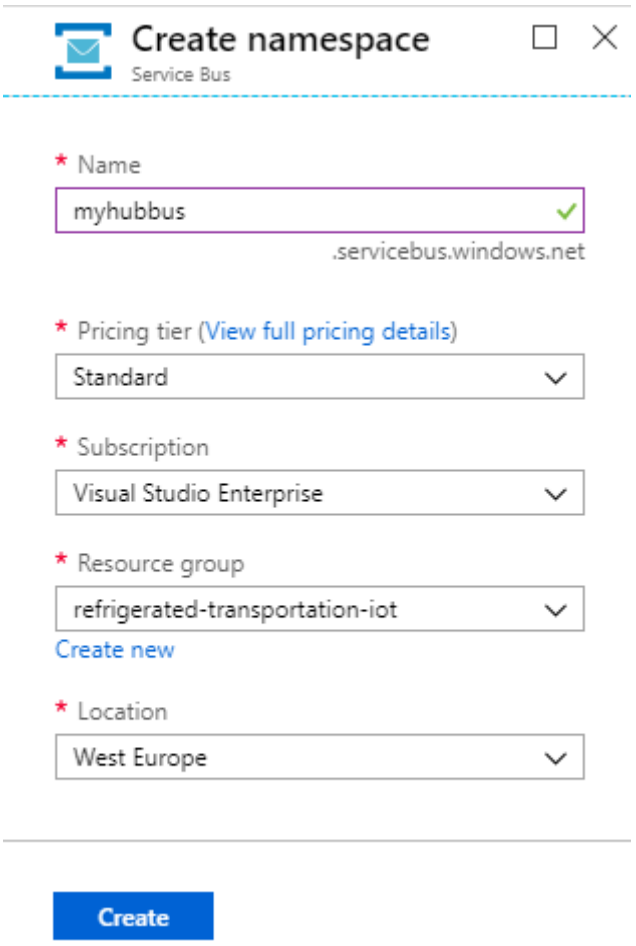
Service Bus queues support a brokered messaging communication model. When using queues, components of a distributed application do not communicate directly with each other; instead, they exchange messages via a queue, which acts as an intermediary (broker).

To create the *Service Bus*:

1. Sign into the Azure portal.

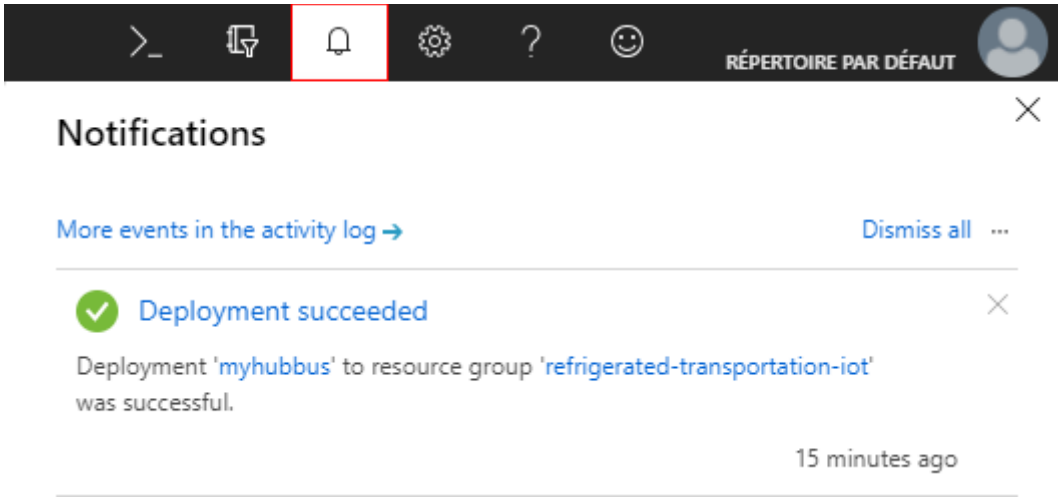
2. In the left navigation pane of the portal, select **+ Create a resource**, select **Integration**, and then select **Service Bus queue**.


Dashboard > New > Marketplace > Everything
Marketplace
My Saved List
Everything
Compute
Networking
Storage
Web
Mobile
Containers
Databases
Analytics
AI + Machine Learning
Internet of Things
Integration
Security
Identity
Everything
Service Bus
Pricing
All
Operating System
All
Publisher
All
Results
NAME
PUBLISHER
CATEGORY
Service Bus
Microsoft
Analysis Services
Microsoft
Databases
Microsoft Web Application Proxy - WAP 2016 Server
Cloud Infrastructure Services
Networking
MySQL Server 8.0 on Windows 2016
Cloud Infrastructure Services
Databases
BizTalk Service
Microsoft
ADFS 4.0 Server Windows 2016
Cloud Infrastructure Services
Compute
Web App
Microsoft
Veeam Cloud Connect for Service Providers
Veeam
Storage
Azure AD Domain Services
Microsoft
Identity
Modernization Platform as a Service (ModPaaS BYOL)
Modern Systems
Compute


Create namespace
Service Bus
* Name
myhubbus
.servicebus.windows.net
* Pricing tier (View full pricing details)
Standard
* Subscription
Visual Studio Enterprise
* Resource group
refrigerated-transportation-iot
Create new
* Location
West Europe
Create

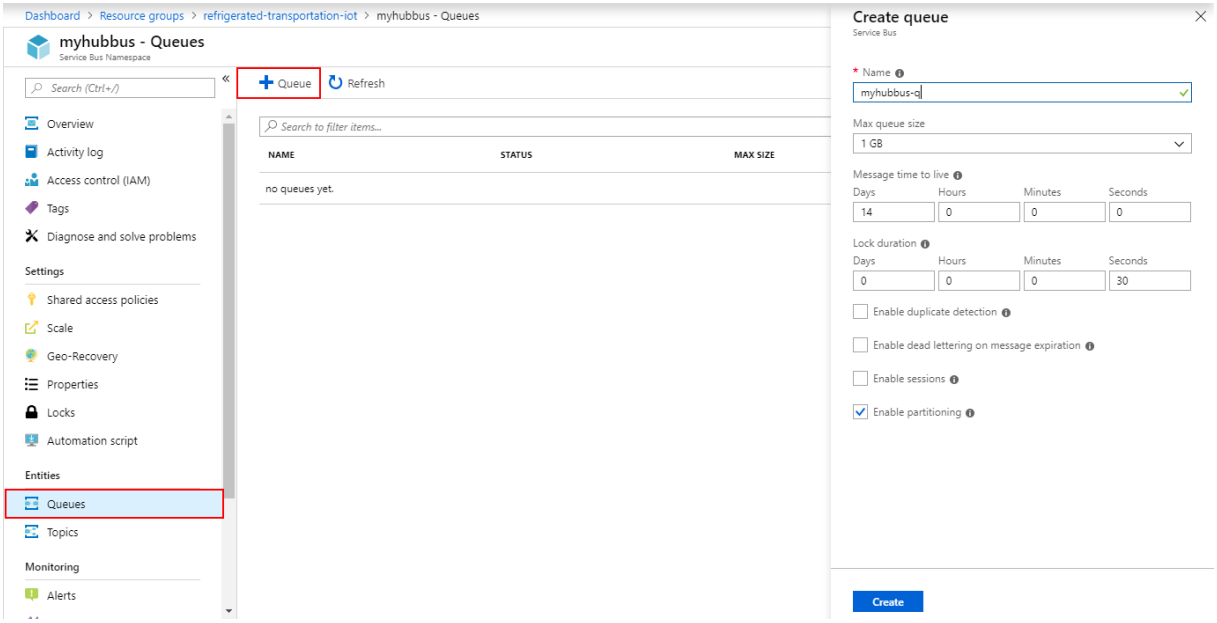
Setting	Description
Name	The name of your namespace for the bus. It should be globally unique, so the system will immediately check to see if the name is available.
Pricing Tier	The pricing tier of the resource (Basic, Standard or Premium).
Subscription	Select the subscription to use for your Service Bus. We 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, we 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, we 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.

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**.



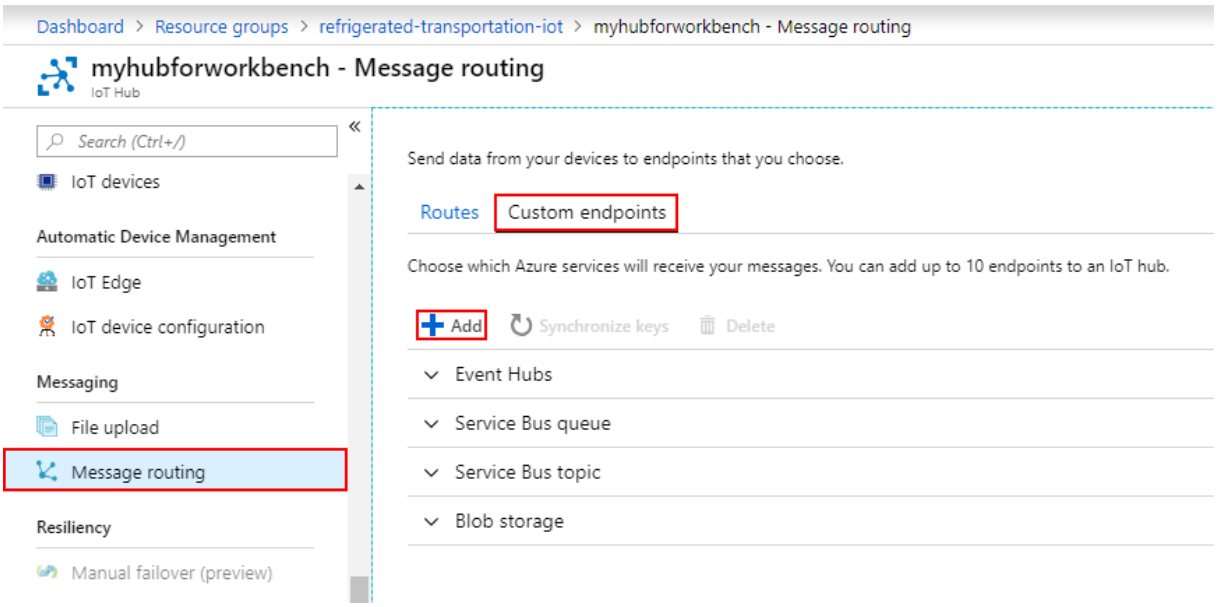
Your queue is now online and you're ready to configure routing in your Hub and your Service Bus.

Route messages to the Service Bus queue

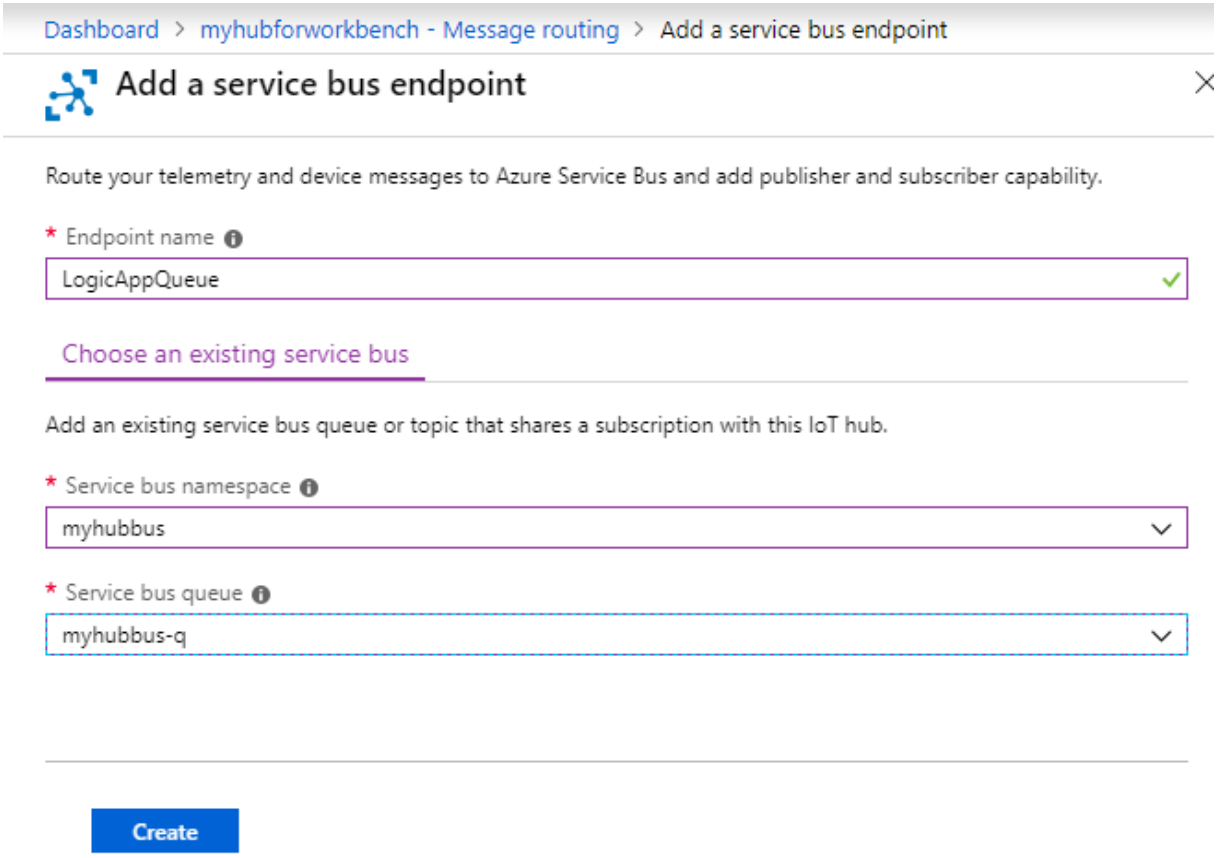
Now that the *Service Bus* is operational, you can define the routes for incoming messages.

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**.



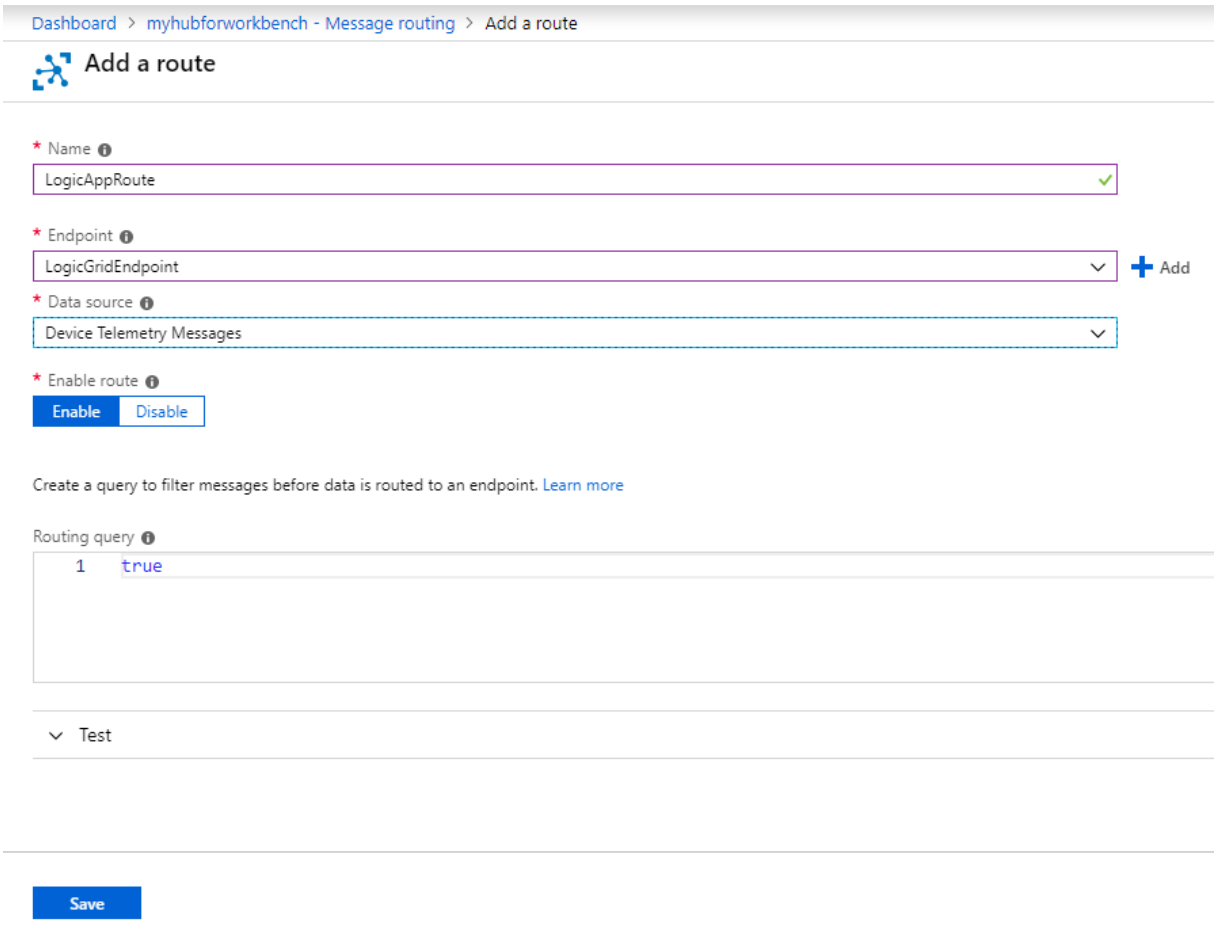
3. Choose **Service Bus queue** on the dropdown list, and a new tab will open. Complete it and click on Create.



Setting	**Description**
Endpoint name	This is the name of 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 route information.



Setting	Description
Name	This is the name of your future route, I chose **LogicAppRoute** here.
Endpoint	In the dropdown list, choose the name of the endpoint you've created in your 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, we want it **enabled** by default, so click on it.
Routing query	This field can contain a filter for our messages. As we need that all the messages are redirected to our Logic App, let the default **true** variable in.

Push routed messages to the Workbench

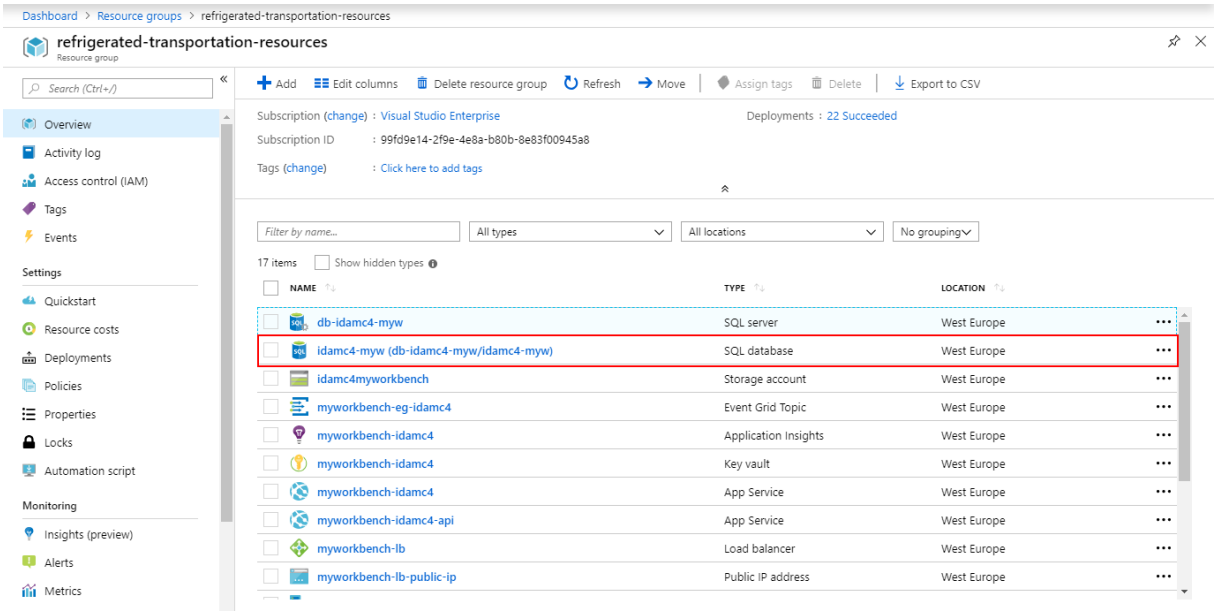
Device messages are now correctly routed to our *Service Bus queue*. But now, we need to push the message into *Azure Blockchain Workbench*.

This will need the deployment of a *Logic App*, which will be in charge of doing operations when a message is received by the *Service Bus queue*.

Before that, we will need to create a *Stored Procedure*, because at some point of the *Logic App* we will map the identity of the device who send a message to its *Azure AD identity*.

Create the required stored procedure

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**.

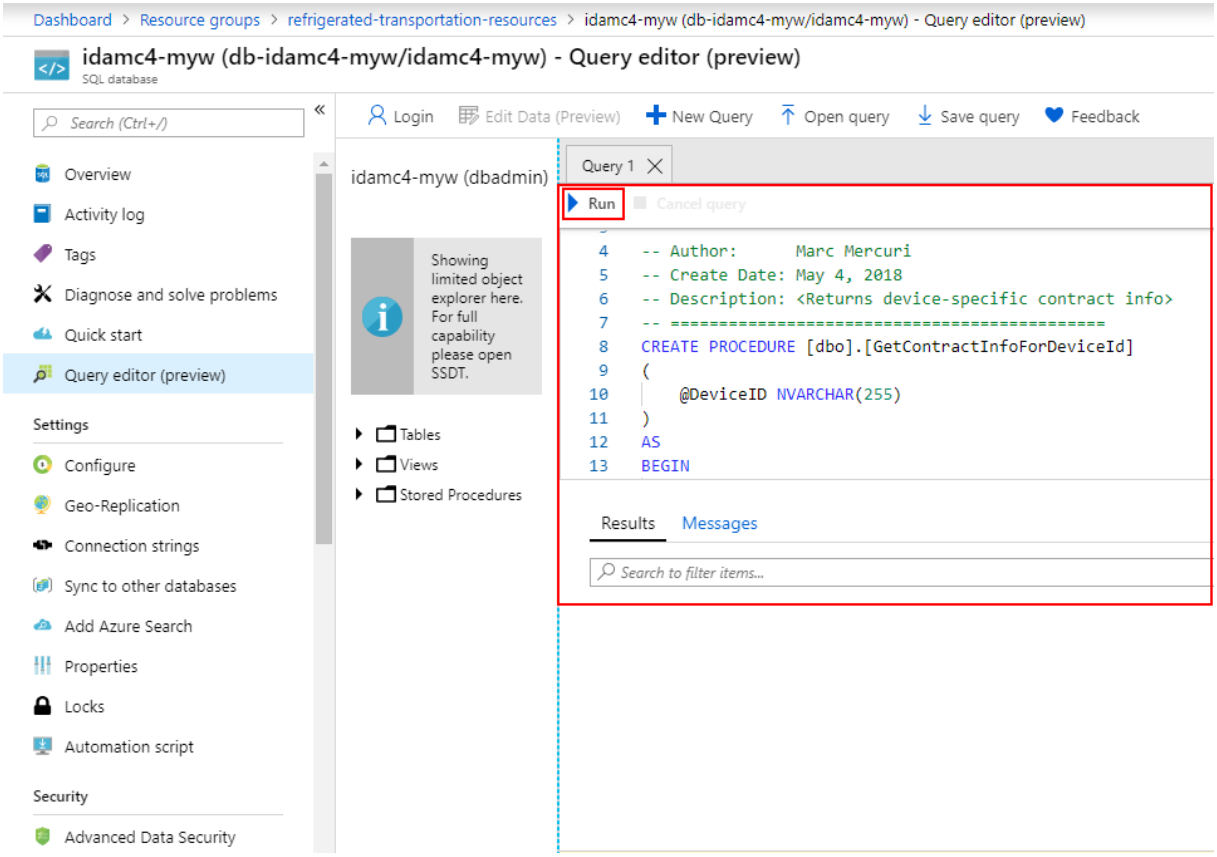
3. Log in using your credential defined when you created your Workbench.

4. Open the **refrigerated-sc-sample** which is the code for this guide that you downloaded before (or grab it here if you don't have it yet), then paste this command and click on **Run**.

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

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 inside **IoT setup** directory, paste its content into the editor and 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 *Logic App*.

Create 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 call a step "Get contract parties" instead of the default "Execute a stored procedure".

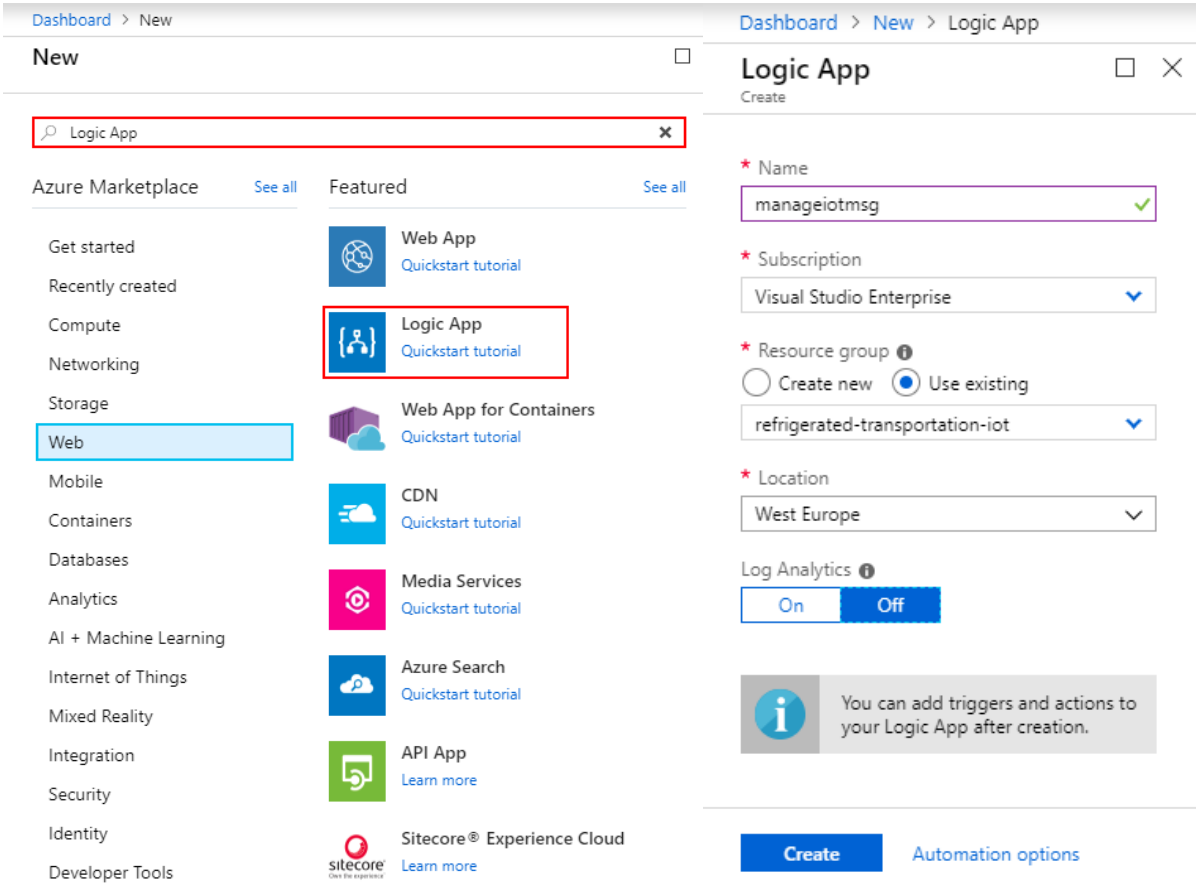
If you want to do that, keep in mind that you should do it when creating the step, and not at the end of the Logic App creation, because it's impossible to rename a step if some data from the step is used somewhere else. In this guide, we will keep default step names.

1. Log in to the Azure portal.

2. Choose +**Create a resource**, then choose **Web**.

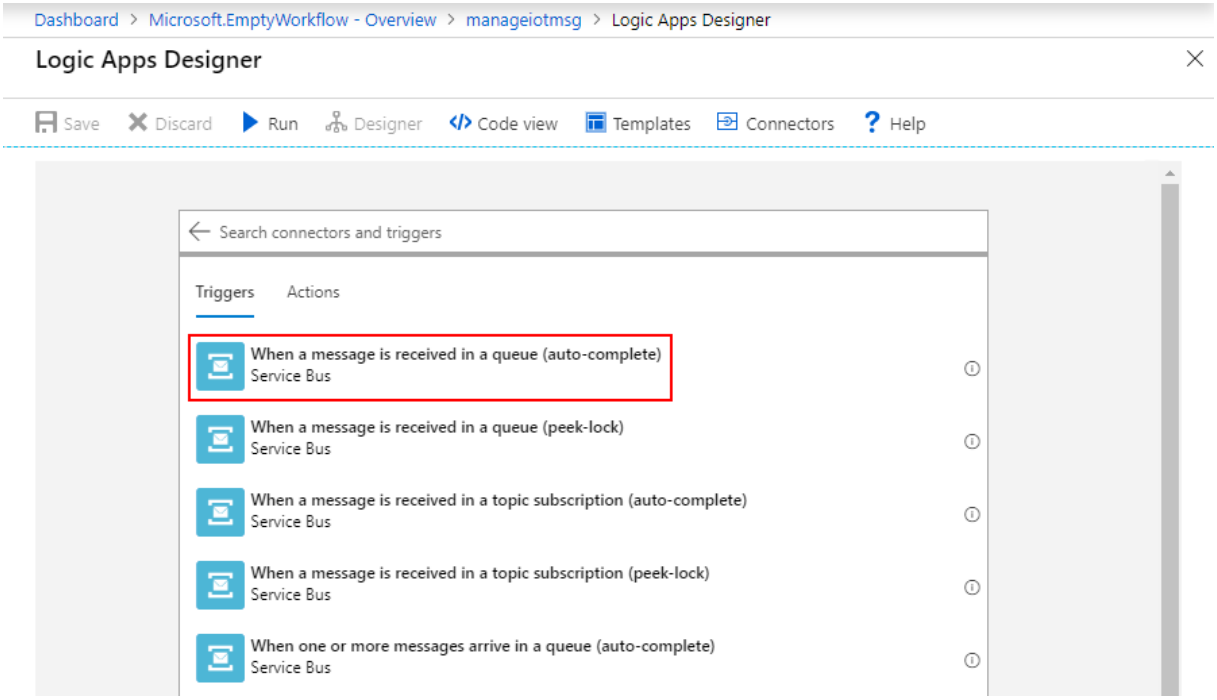
3. Click **Logic App** from the list on the right.

4. Fill the configuration tab.

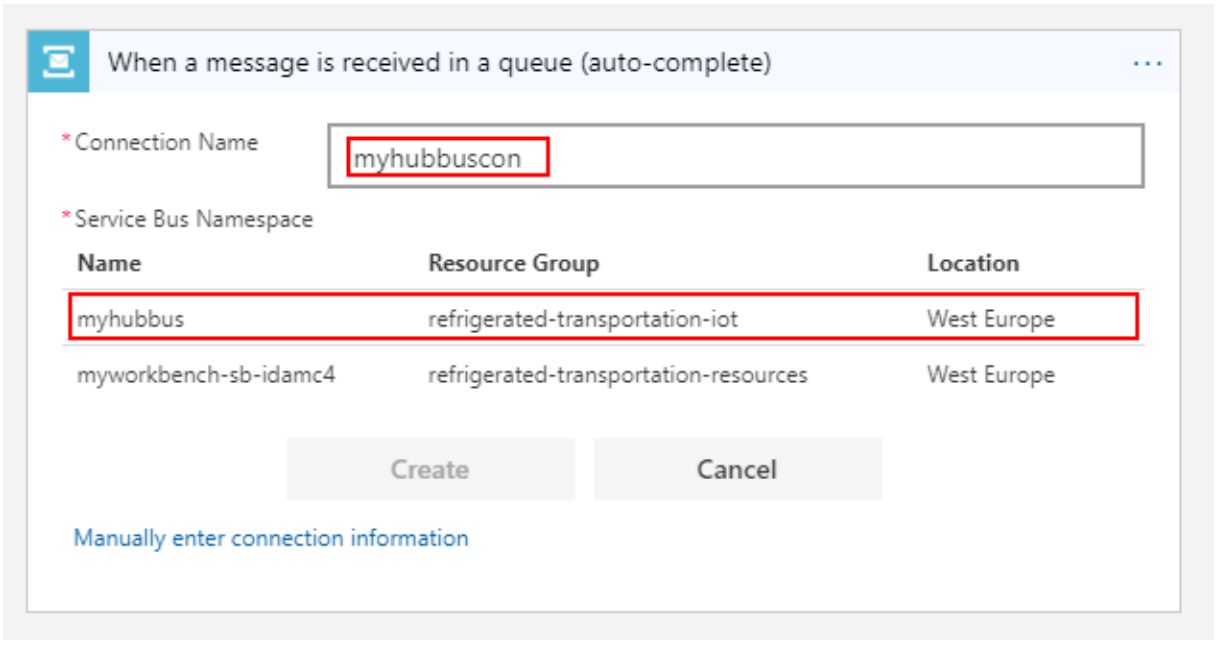


Setting	Description
Name	This is the name of your future app.
Subscription	Select the subscription to use for your *Logic App*.
Resource group	You can create a new resource group or use an existing one. To create a new one, click **Create** new and fill in the name you want to use. To use an existing 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 **Service Bus** and search for **When a message is received in a queue (auto-complete)**, then click on it.

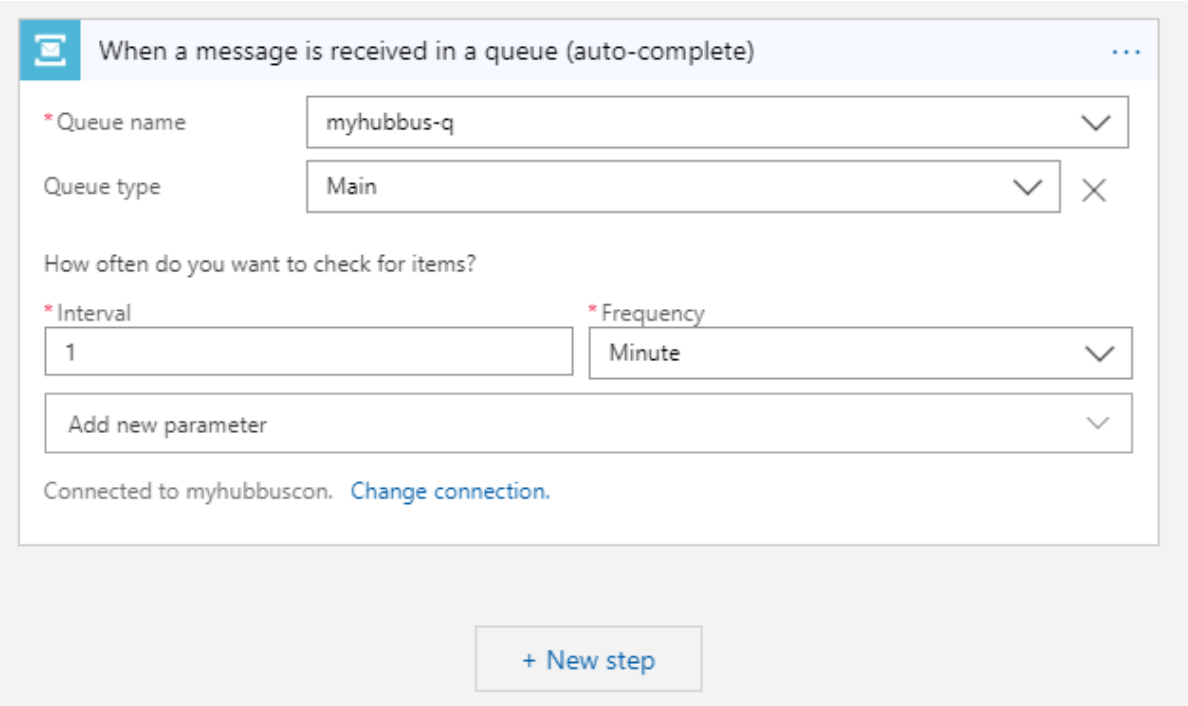


6. Create a connection to the Service Bus by entering a name and selecting the Service Bus in the list.



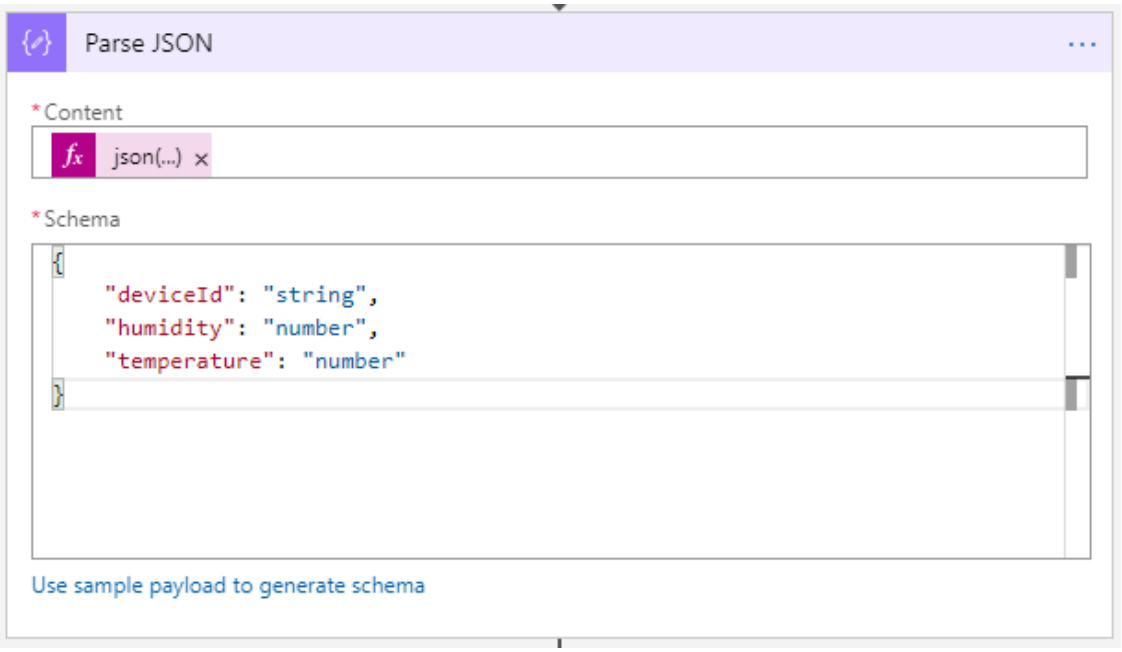
7. Click on the *Service Bus Policy* called **RootManageSharedAccessKey** and then click on **Create**.

8. In the next step, change the trigger interval from the default value to **1 minute,** then click on **+ New Step**.

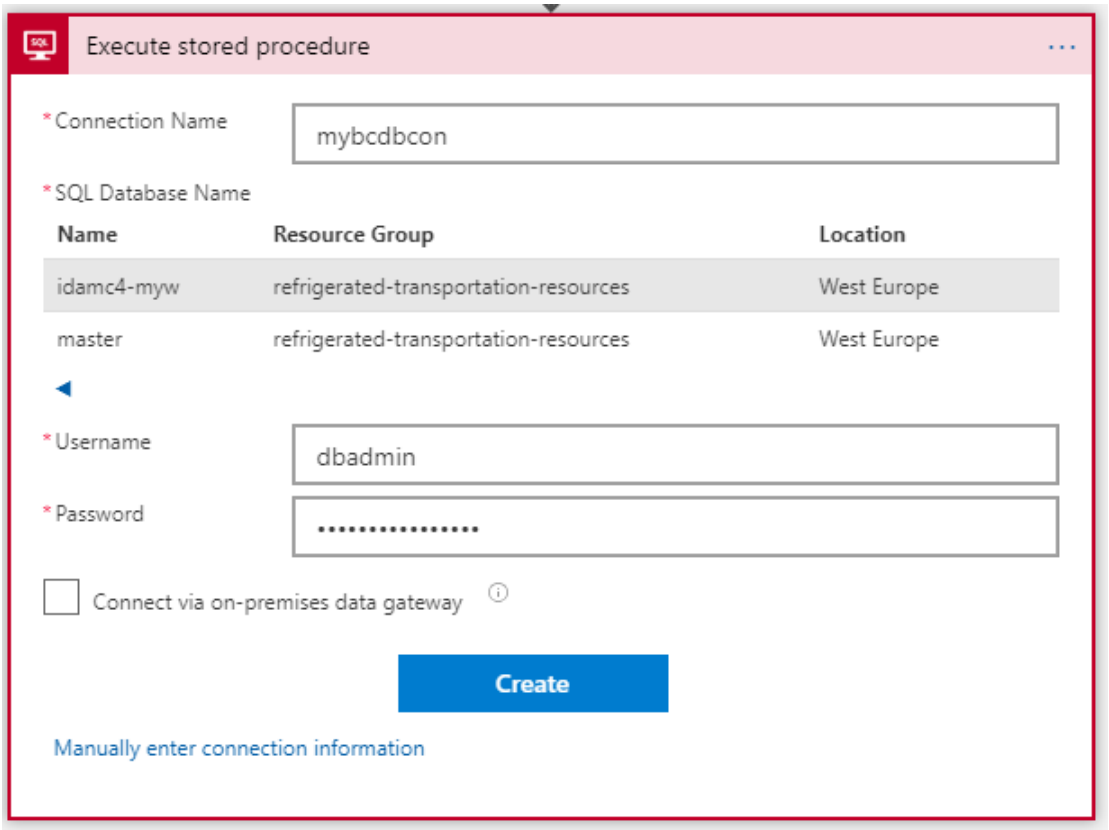


9. Search for **Parse JSON** action. Into the content field, paste *json(base64ToString(triggerBody()?['ContentData']))*. And, into the schema property, paste the code below. Finally, click on **+ New step**.

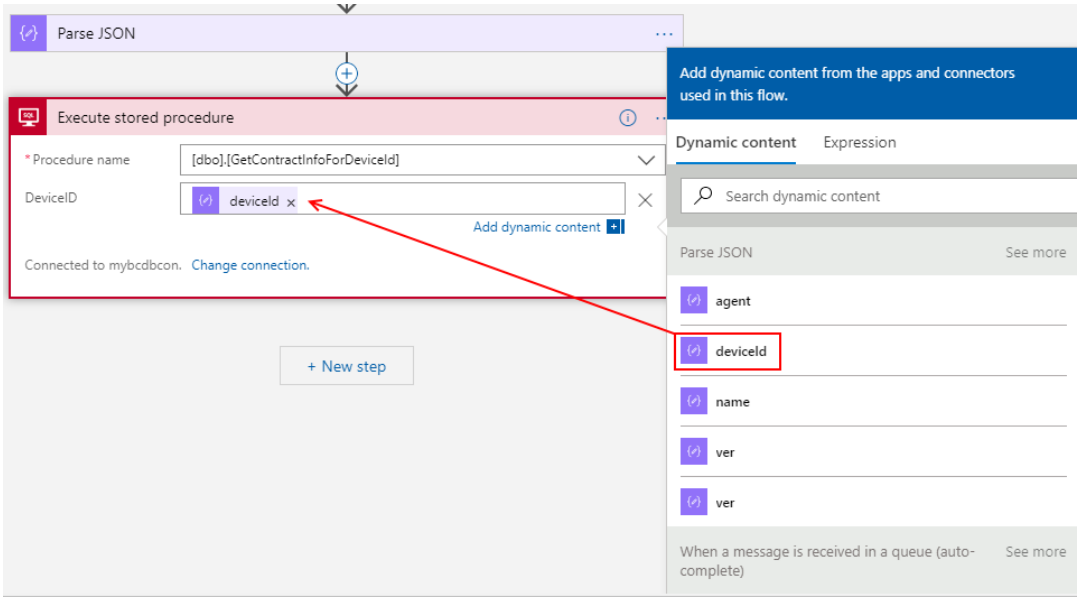
```
{
    "type": "object",
    "properties": {
        "deviceId": {
            "type": "string"
        },
        "temperature": {
            "type": "integer"
        },
        "humidity": {
            "type": "integer"
        }
    }
}
```



10. Search for **SQL Server**, then **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.



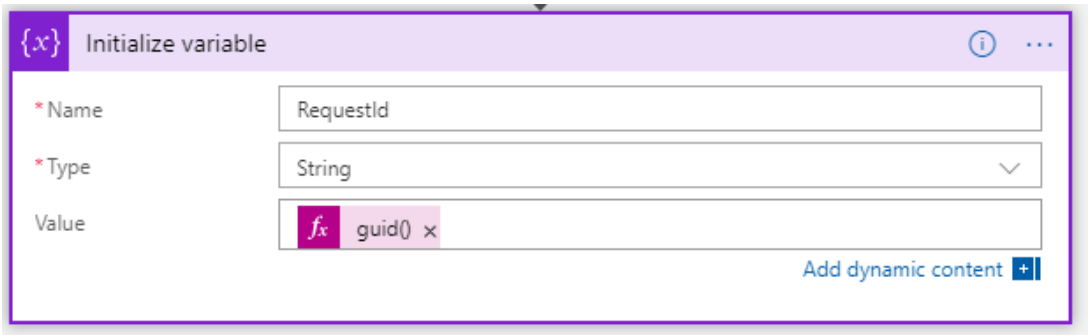
11. Search and click on the procedure **[dbo].[GetContractInfoForDeviceId].** Click on **Add new parameter** and select the only one available (**DeviceID**). If you click on the blank field which appeared, a new tab will open. Search for **deviceId** and select it.



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

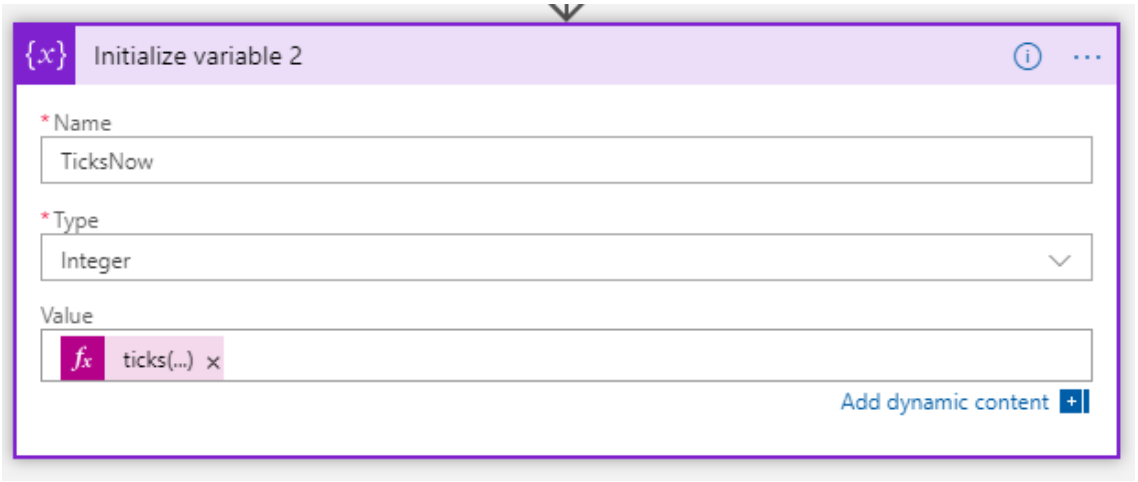
In the case that the value is different, remove *deviceId* and click in the field to add dynamic content, then paste the correct value in the **Expression** tab and click on **OK**.

13. Select the **Initialize Variable** action and fill it.
    - Name: RequestId
    - Type: String
    - Value: Click on the field, then go in the Expression tab and type *guid()*, then press **Enter**.



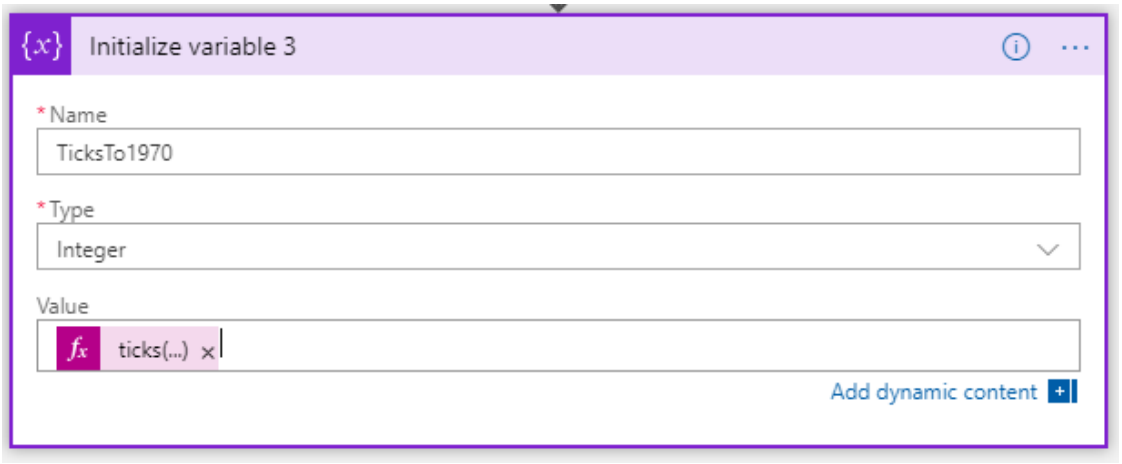
Finally, click on **+ New step**.

14. 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())* then press **Enter**.



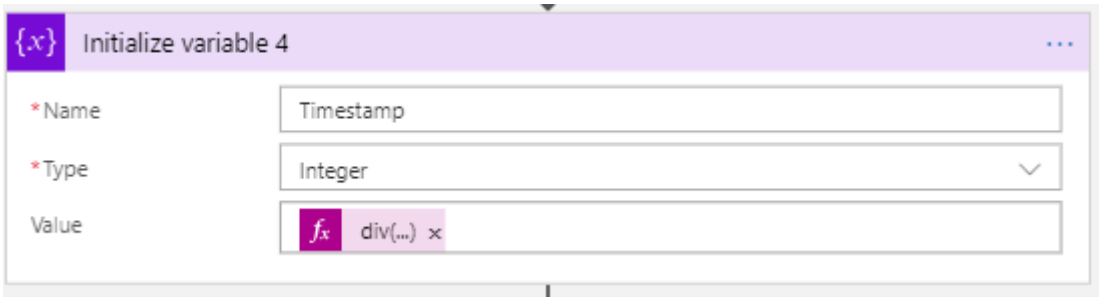
Finally, click on **+ New step**.

15. Select the **Initialize variable** and fill it.
- Name: TicksTo1970
- Type: Integer
- Value: Click on the field, then go in the Expression tab and type *ticks('1970-01-01')* then press **Enter**.



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

16. Select the **Initialize variable** and fill it.
- Name: Timestamp
- Type: Integer
- Value: Click on the field, then go in the Expression tab and type *ticks(*`div(sub(variables('TicksNow'),variables('TicksTo1970')),10000000)` then press **Enter**. The difference between those 2 variables generates a Timestamp.

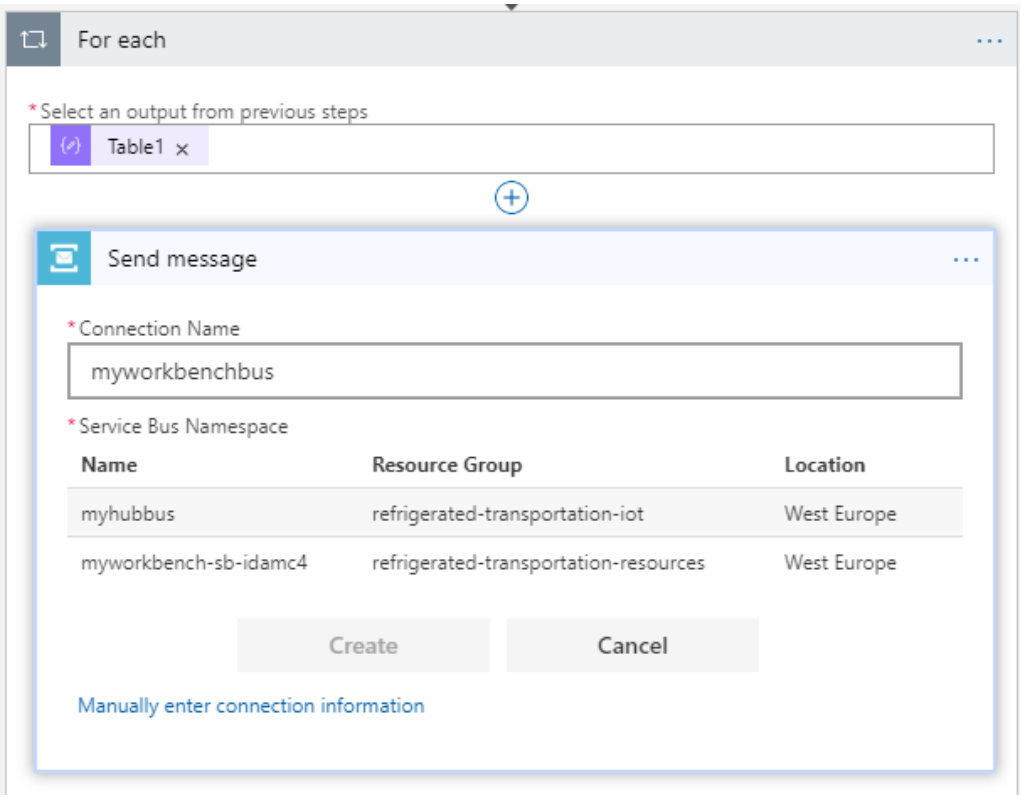


Finally, click on **+ New step**.

17. Select the **Control** action. Click on the Content field, search and click for **For each**. Click on the **output** field and select **Table1**, then click on Take an action.

18. Inside the new action within the *For each* block, click on **Service Bus**, then 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.

19. 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.



20. To finish, we must aggregate the device data, the timestamp variable and the Workbench information (like userId, contractID ...) 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 called **RequestId**.

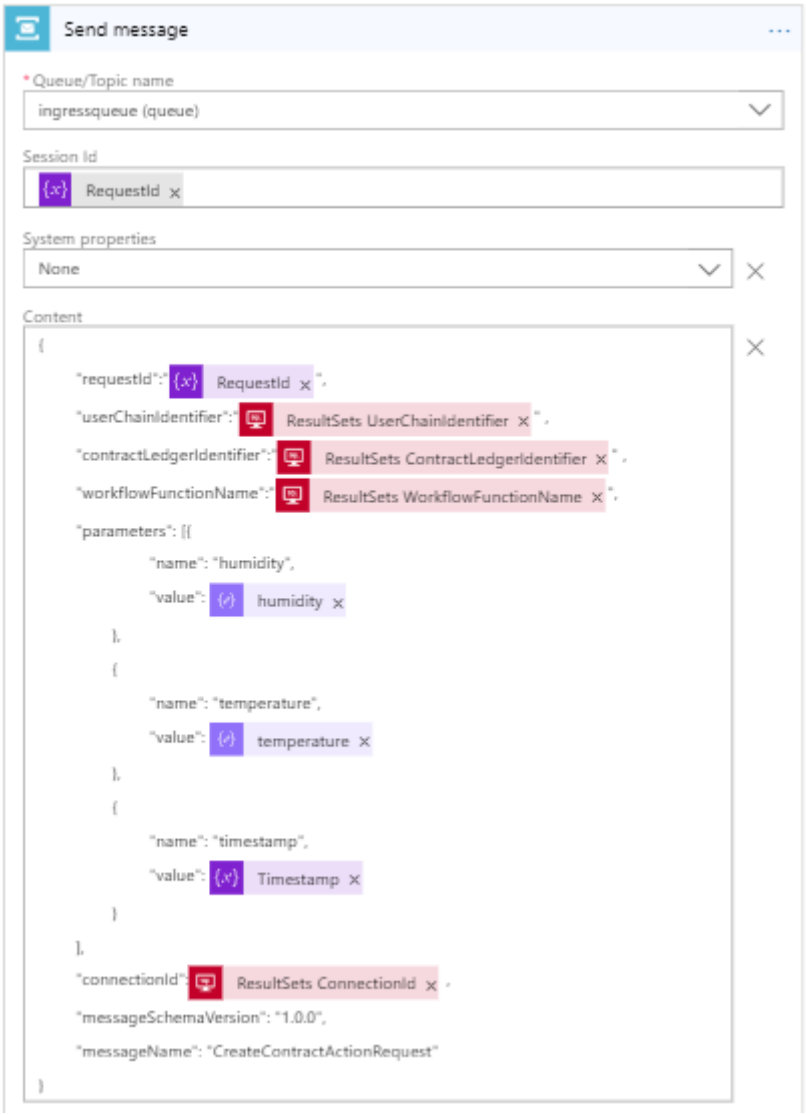
    Now, we need to add the content of the message. Click on **Add new parameter** and check the box in front of *Content*. A text area will appear, paste this code inside.

```
{
    "requestId": "",
    "userChainIdentifier": "",
    "contractLedgerIdentifier": "",
    "workflowFunctionName": "IngestTelemetry",
    "Parameters": [
        {
            "name": "humidity",
```

```
            "value": ""
        },
        {
            "name": "temperature",
            "value": ""
        },
        {
            "name": "timestamp",
            "value": ""
        }
    ],
    "connectionId": 1,
    "messageSchemaVersion": "1.0.0",
    "messageName": "CreateContractActionRequest"
}
```

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

To do that, 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 thinks that they are out of context, but you can display it by clicking on **See more** in the *Dynamic content* tab.



Our *Logic App* is now complete and ready to forward messages to *Azure Blockchain Workbench*. Now, we need to set up some devices to test our 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 our solution

We are now ready to connect external devices to our solution. For this guide, I will demonstrate how to connect and provision a device, and how to send messages to our *IoT Hub* on 4 different types of devices:

- A *node.js simulated device*
- An *MXChip IoT DevKit*
- An *Azure Sphere DevKit*
- A *Raspberry Pi*

You can skip any part in which you don't have the sensor or the device because the *node.js simulated device* will be enough for our guide.

Node.js simulated device

Overview and prerequisites

The first "device" that we 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 configure a device to this link.

Before starting this guide, 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 (if you don't have it yet, you can get it from here), and once inside open go through **IoT setup** > **simulated-device**, then type:

```
npm install
```

It will install required packages for our application, like *Azure IoT device*.

Register the simulated device

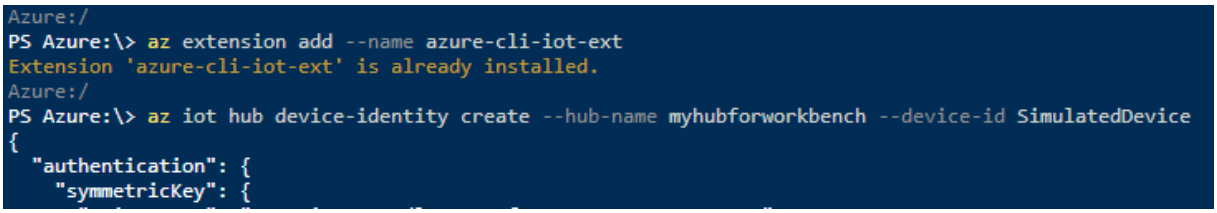
To work, our device needs to log in to the IoT Hub. That's why we need to create an identity for our device to work, by adding it to the Hub device list.

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 Shell, run this command by replacing *YourIoTHubName* by the name of your IoT Hub and SimulatedDevice by a custom name if you want to change it.

```
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. Device's identity is now created.



3. We now need to get the connection string that our connected device will use. Run this command in your *Azure Shell*.

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

   Make a note of the device connection string, which looks like:

   ```
   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 favorite code editor. Search for the connection string line and replace it with your own connection string.

5. Finally, we need to link your device ID to an existing user account in your *Azure AD tenant*. We are going to use the user *iot-device-1@<your tenant>.onmicrosoft*.com that we'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. Log 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'*.

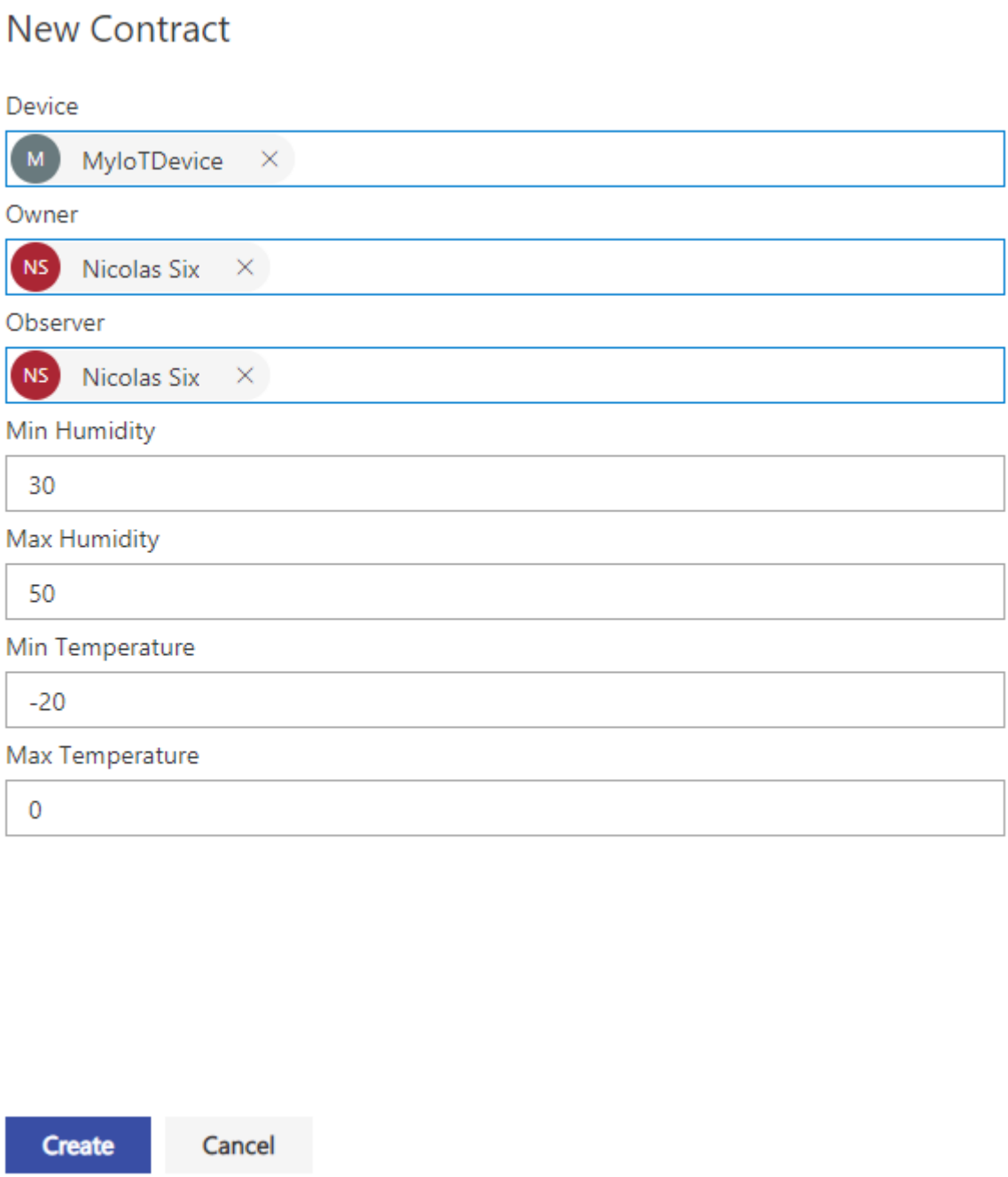
8. Check that the command has been correctly executed by looking into the messages tab below the *Query Editor:* it should be *Query succeeded: Affected rows: 1*.

Our device is now correctly registered into our *Azure IoT Hub* but also into our *Azure AD Tenant* and therefore *Azure Blockchain Workbench*. We are ready to test it!

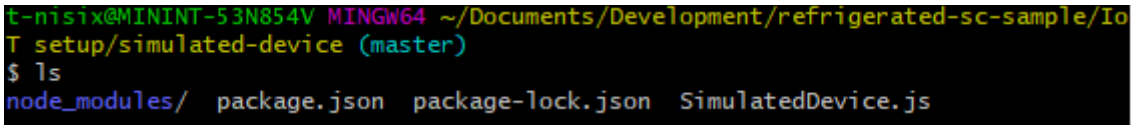
Send simulated telemetry and test application

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. Now that the Contract is deployed, you are ready to start the simulated device. In your computer, start a Terminal and navigate to the root folder of the simulated device.

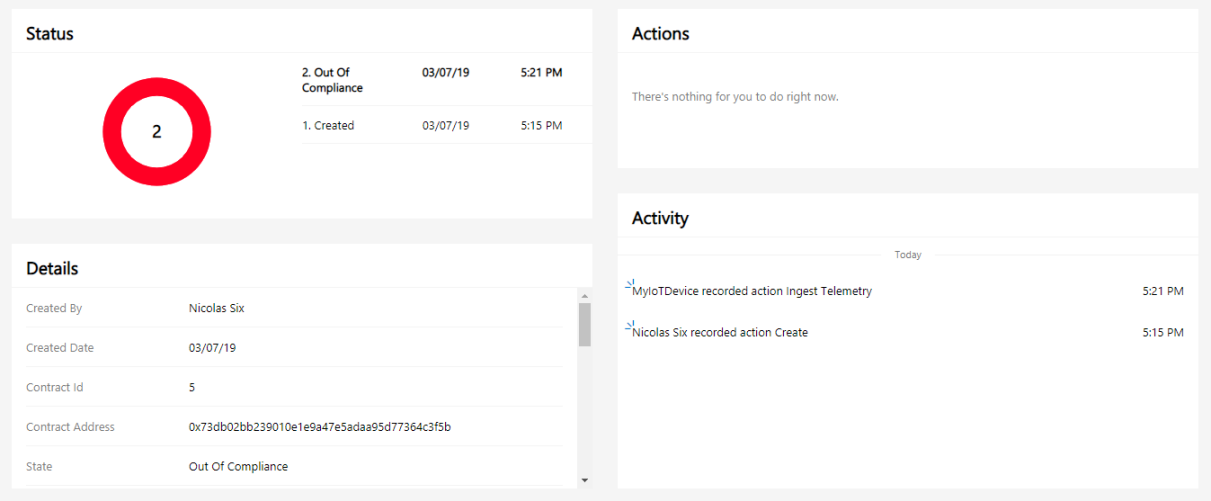


4. In this terminal, type those commands.

```
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 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'*.



Your simulated device is now working and having done that is a good way to understand all the process between your device and the Workbench in order to apply the same pattern to other real devices that you will see in the next parts of this guide.

MXChip IoT DevKit Device

Overview and prerequisites

In this part, we are going to set up our first real device, the *MXChip IoT Devkit*.

The *MXChip IoT DevKit* is an all-in-one IoT Device Kit, you can use it to develop and prototype IoT solutions that take advantage of 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.

To program the device, we will also use *Arduino IDE*. If you haven't downloaded it yet, you can grab it here.

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.

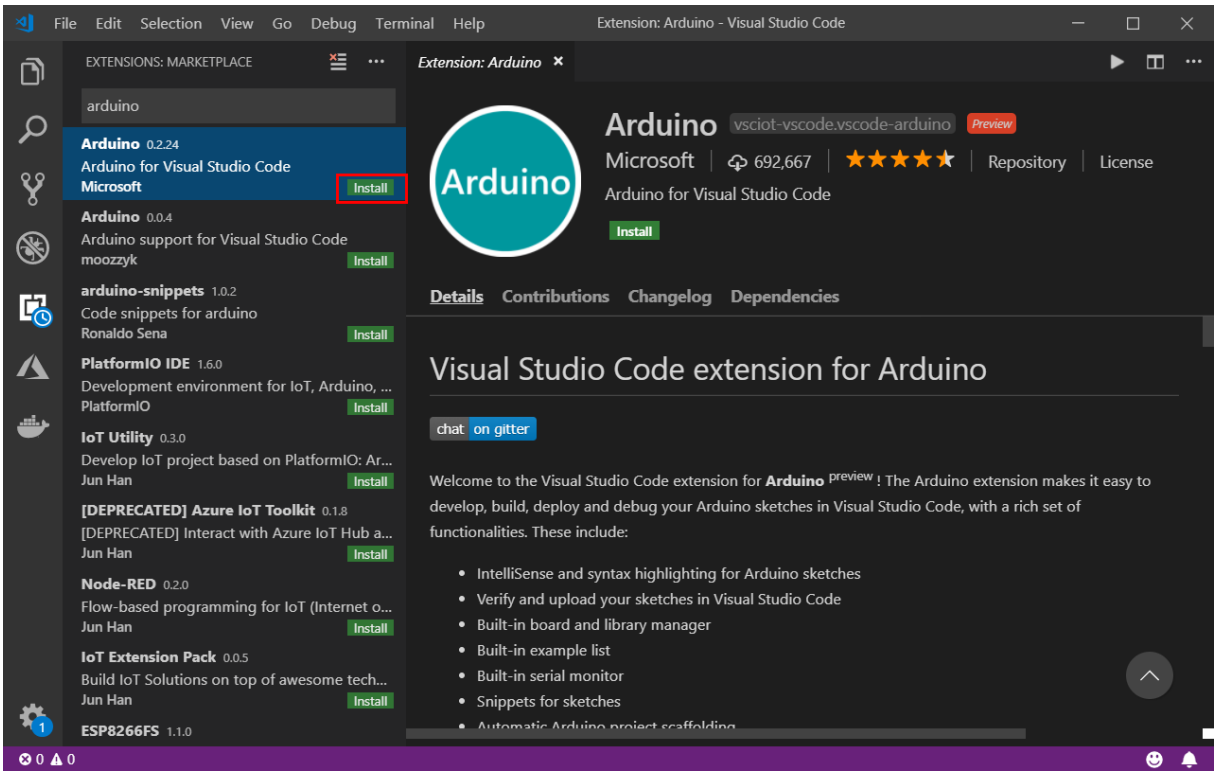
MacOS: No driver is required for macOS.

Ubuntu: Run the following in terminal and log out and log in for the group change to take effect:

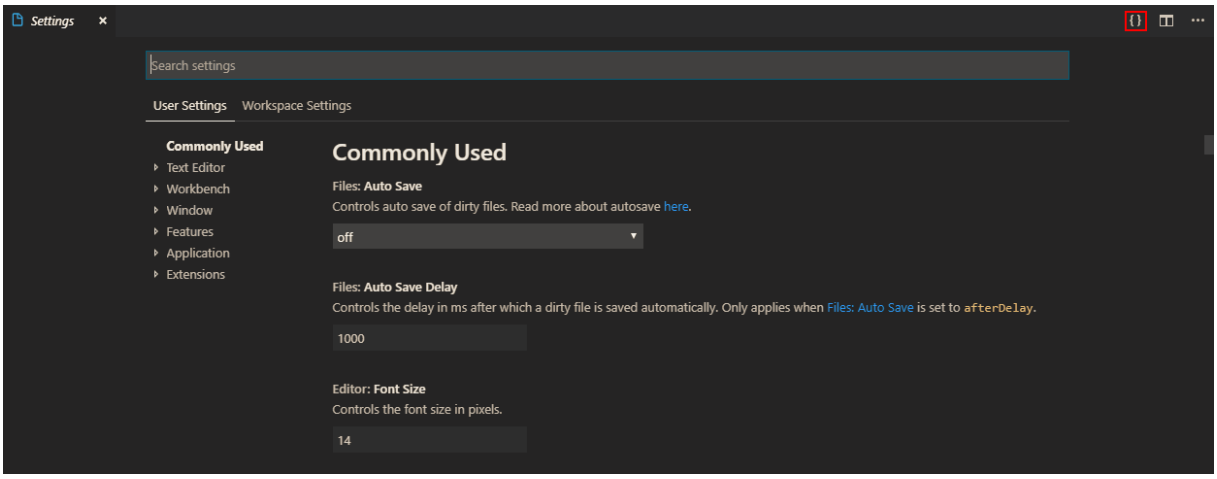
```
# Copy the default rules. This grants permission to the group 'plugdev'
sudo cp ~/.arduino15/packages/AZ3166/tools/openocd/0.10.0/linux/contrib/60-openocd.rules
/etc/udev/rules.d/
sudo udevadm control --reload-rules

# 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 extensions to Visual Studio Code in order to support Arduino and Azure development.

Launch VS Code, look for **Arduino** in the extension marketplace and install it. This extension provides enhanced experiences for developing on the Arduino platform.



Configure VS Code with Arduino settings. In Visual Studio Code, click **File > Preference > Settings**. Then click the **{} icon** to open **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**:

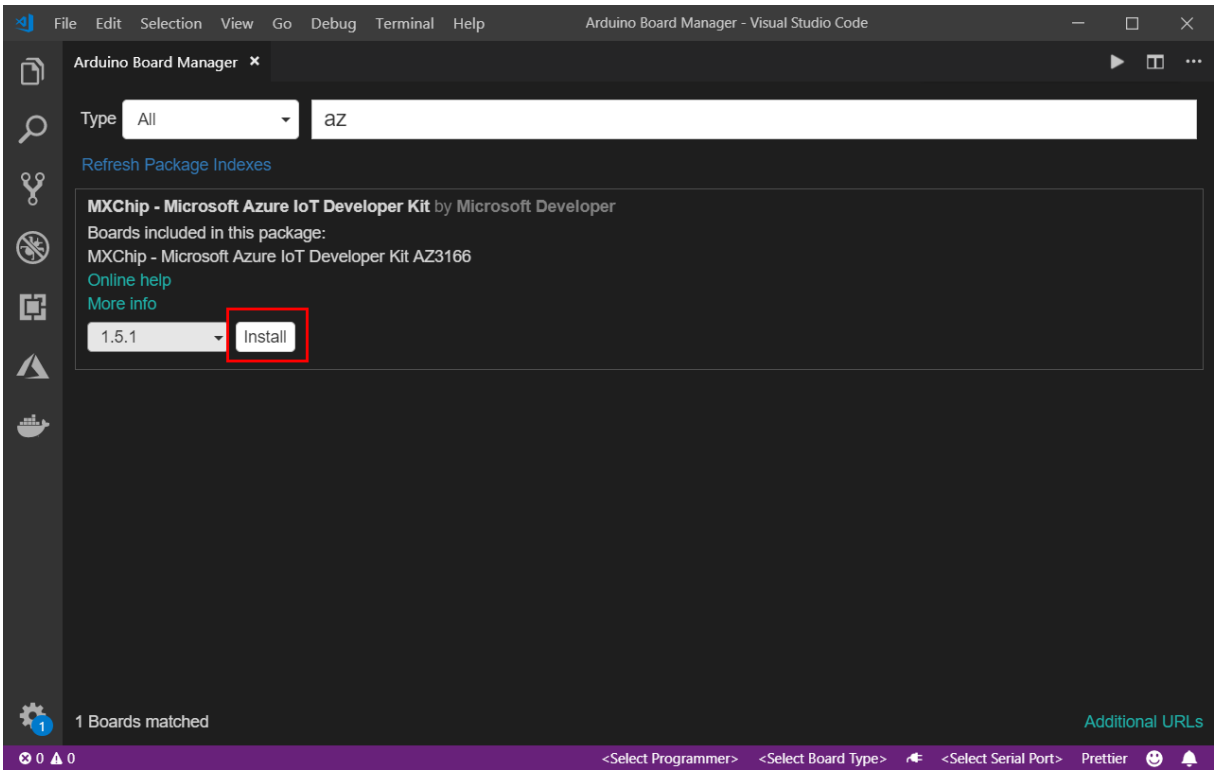
  ```
  "arduino.path": "/Applications",
  "arduino.additionalUrls":
  "https://raw.githubusercontent.com/VSChina/azureiotdevkit_tools/master/package_azureboard_index.json"
  ```

- **Ubuntu**:

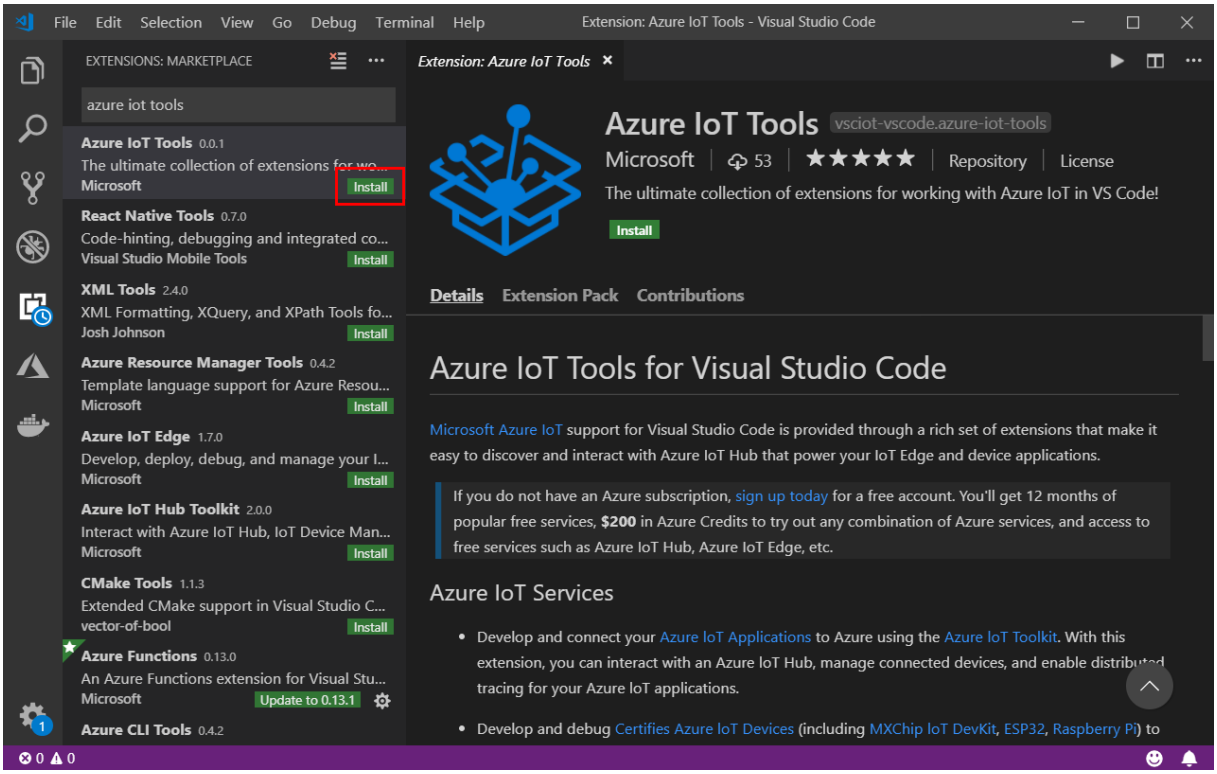
  Replace the **{username}** placeholder below with your username.

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

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



Then look for **Azure IoT Tools** in the extension marketplace and install it.

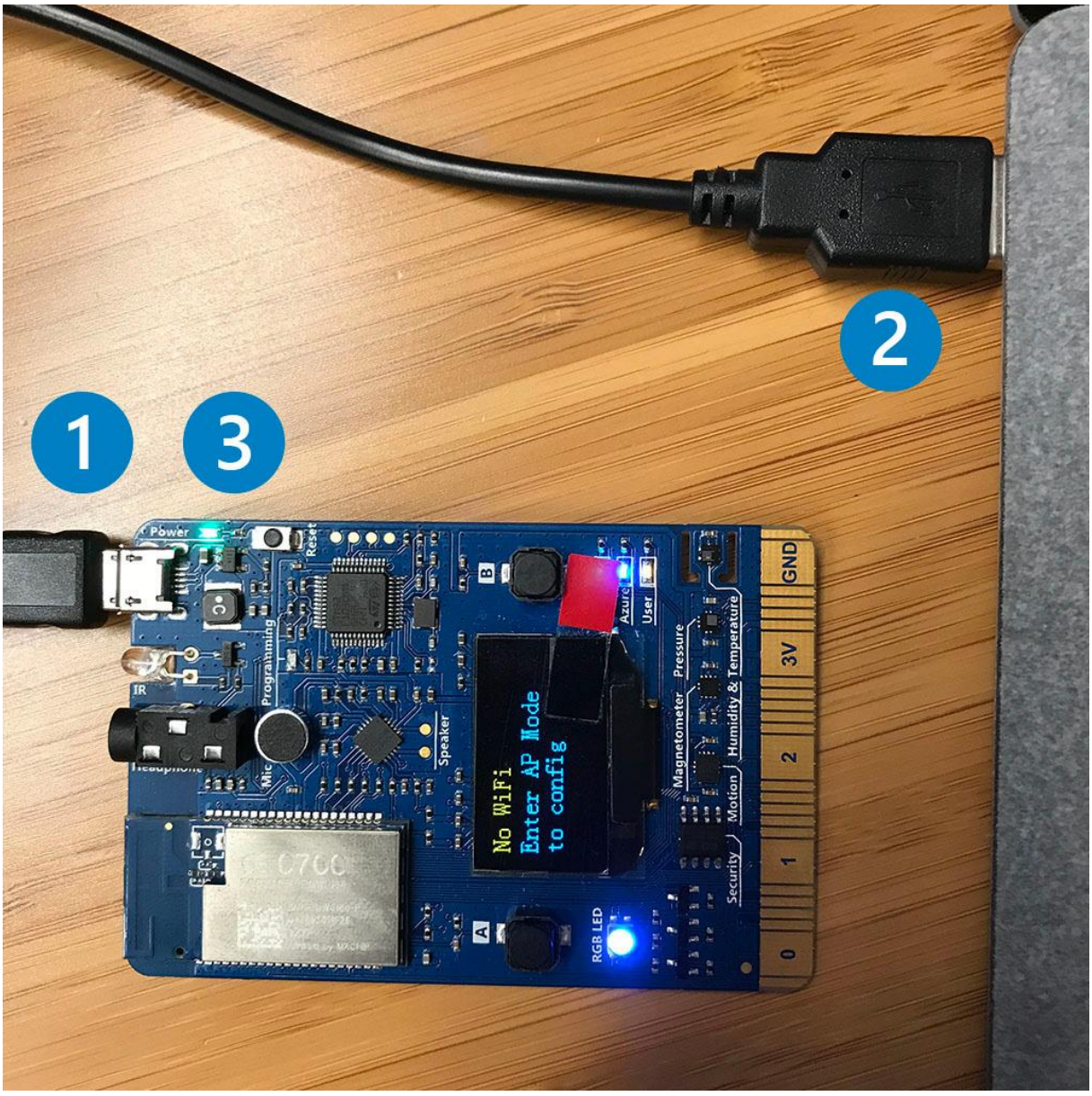


Now that everything is installed, you are ready to open the project and prepare it for our device.

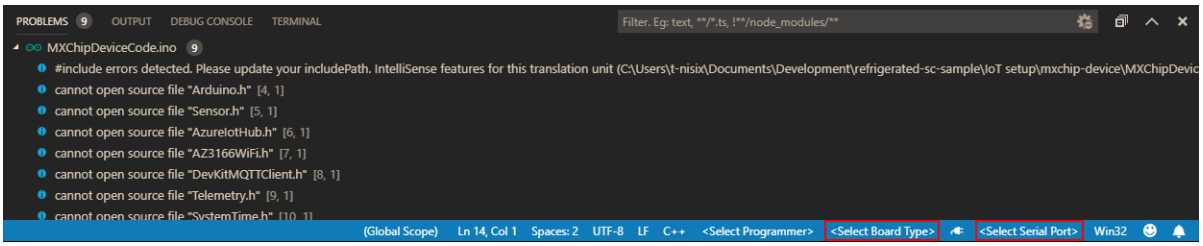
Set up the project and the device

1. In VSCode, open the **mxchip-device.code-workspace** which is inside **refrigerated-sc-sample > IoT setup > mxchip-device** directory.

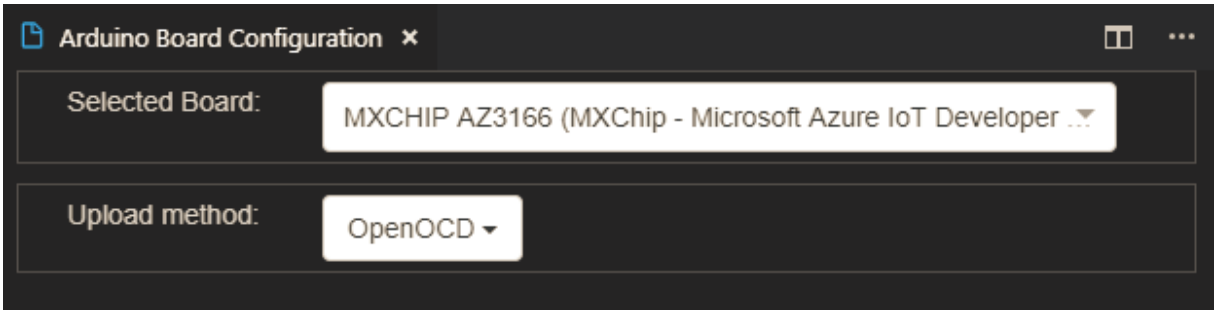
2. Once opened, connect the *MXChip device* to your computer by plugging the USB end to your computer (1) and the Micro-USB end to the device (2). The LED (3) should turn on if the connection succeeded.



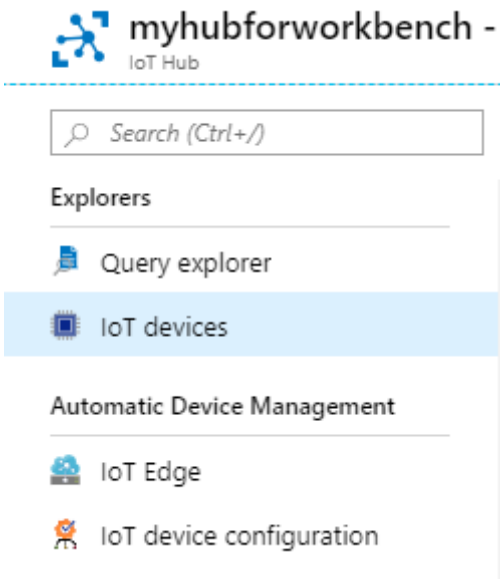
3. Now that the device is plugged in, and if they are not specified yet by *Visual Studio Code*, you can specify which board and port you're using to program the *DevKit*. In *Visual Studio Code* at the bottom of the screen, click on **<Select Serial Port>**, then select **COM3**.



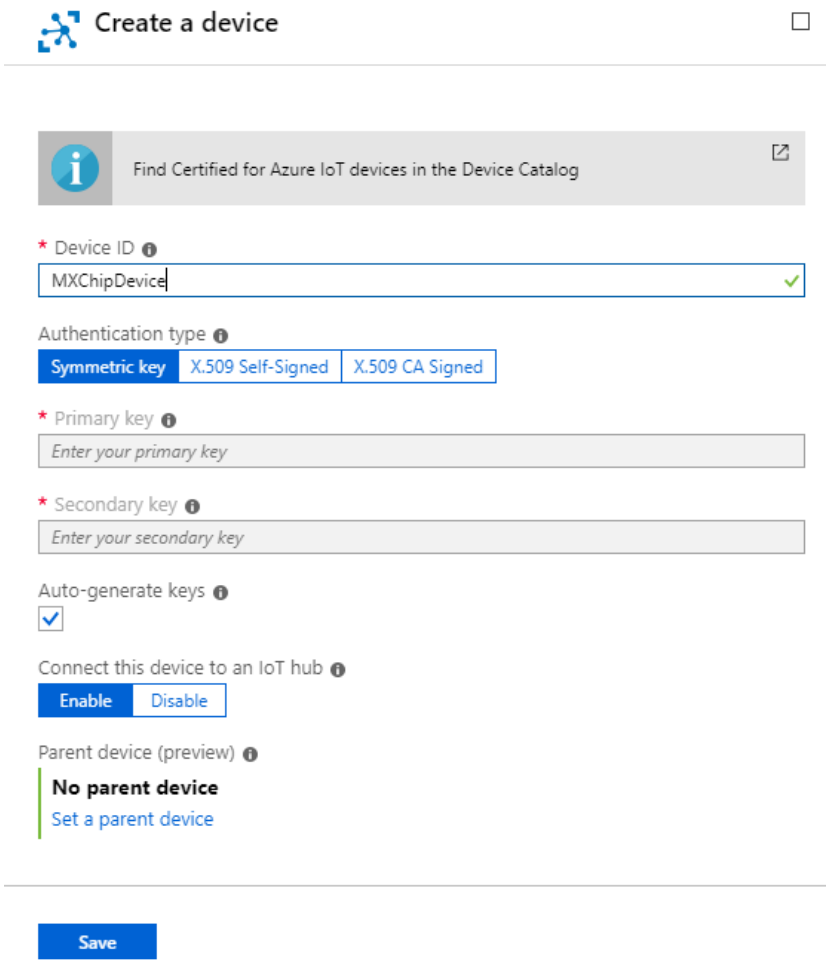
4. Click on **<Select Board Type>.** An *Arduino Board Configuration* tab will open, search for MXCHIP AZ and select the one which appears. Let the other parameter by default.



5. Now, we need to set up the connection between *Azure* (our *IoT Hub*) and our *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. Simply don't forget to also change it inside the code and the SQL request that you will see below.

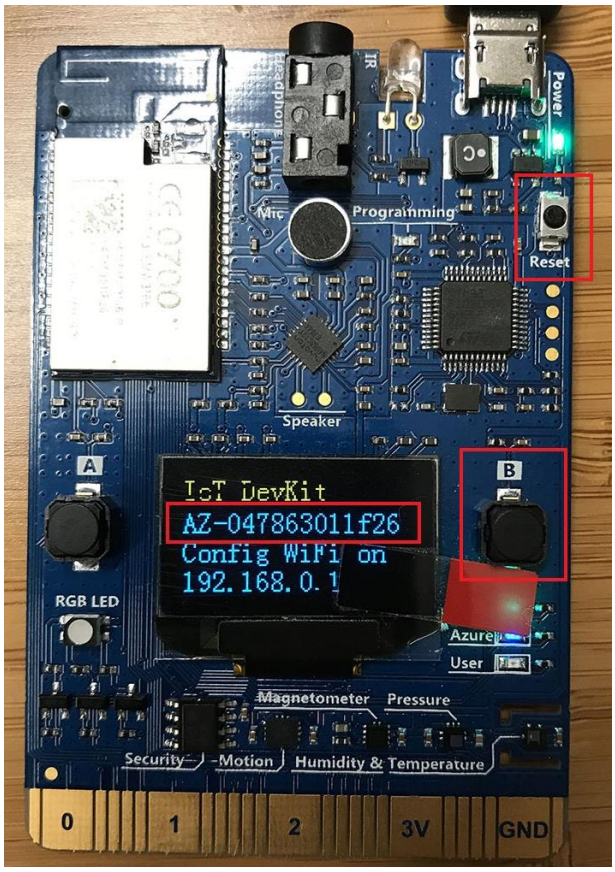
7. You've been redirected to your *IoT devices list*, click on your newly created device and inside the details, copy the **Connection string (primary key)**.

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**.

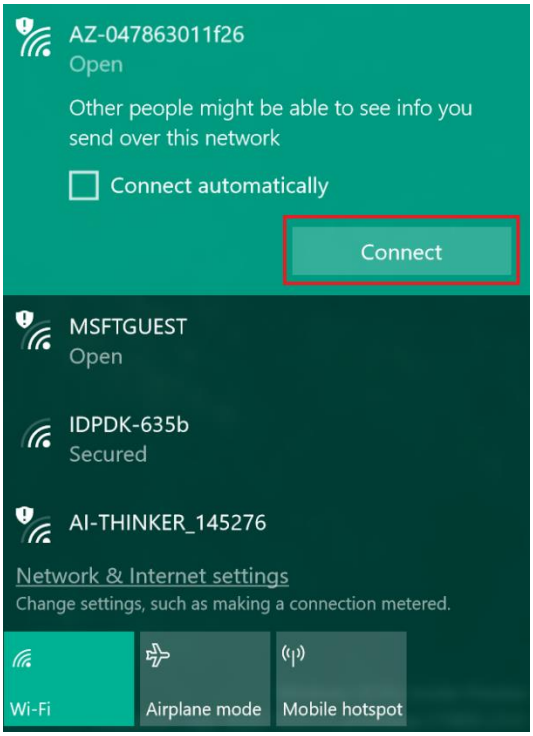
9. On DevKit, hold down **button A**, push and release the **reset** button, and then release **button A**. Your DevKit enters configuration mode and saves the connection string.

10. Press **F1** again, type and select **Azure IoT Device Workbench: Upload Device Code**. It starts to compile and upload the code to DevKit.

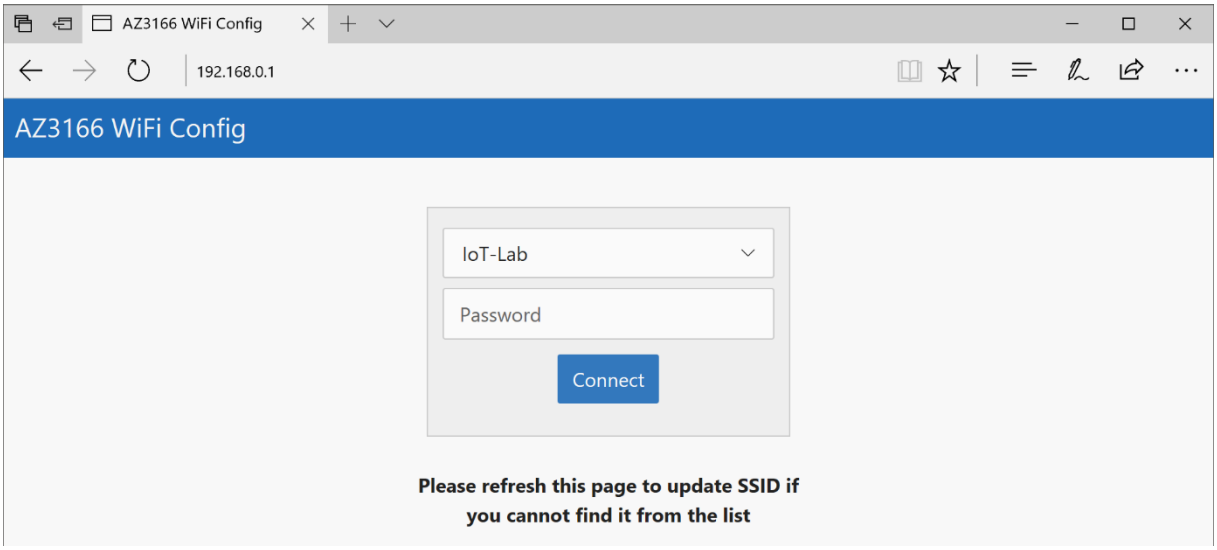
11. The last step to configure your device is to enable Wi-fi connection. Hold down button B, push and release the reset button, and then release button B. Your *DevKit* enters AP mode for configuring Wi-Fi. The screen displays the service set identifier (SSID) of the *DevKit* and the configuration portal 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, we need to link your device ID to an existing user account in your *Azure AD tenant*. We are going to use the user *iot-device-2@<your tenant>.onmicrosoft*.com that we'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. Log 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*.

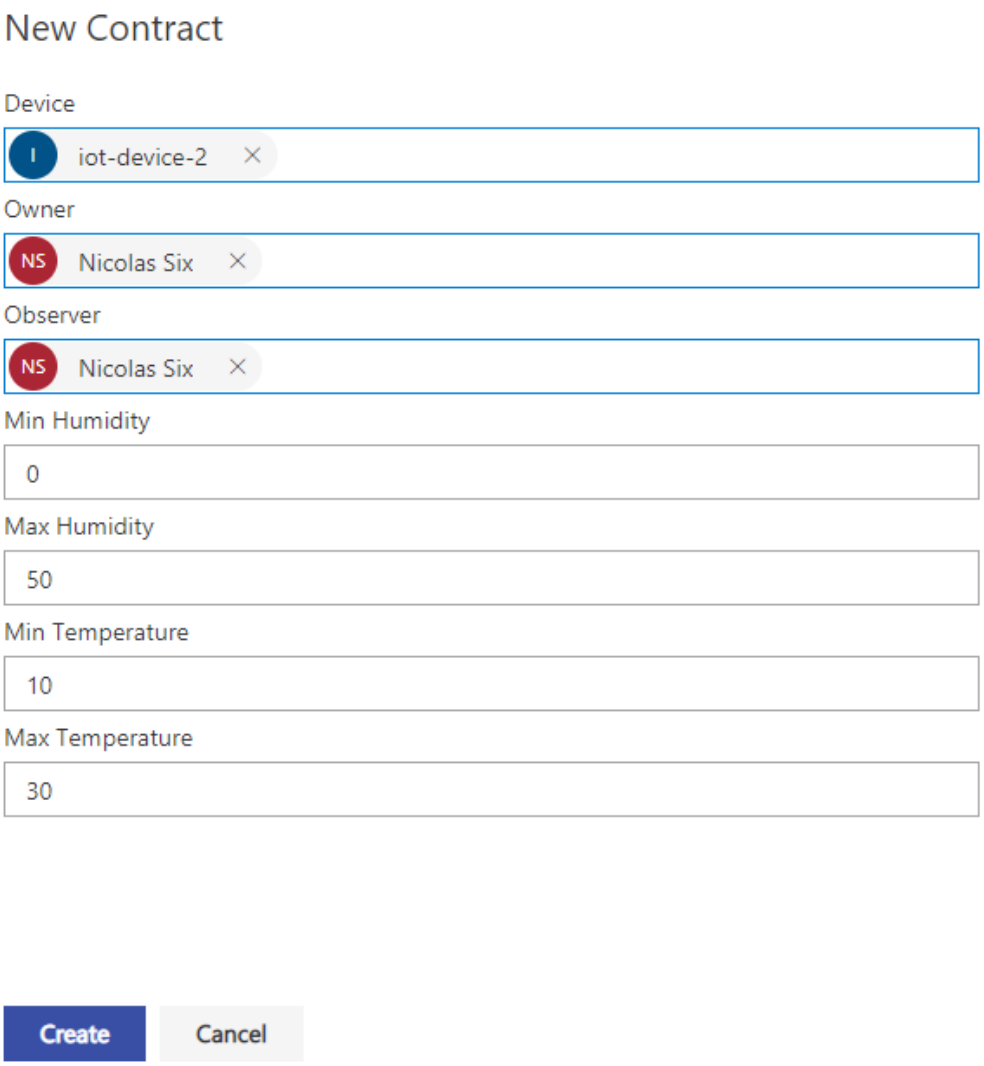
Everything is now set up, let's try our device.

Test the device

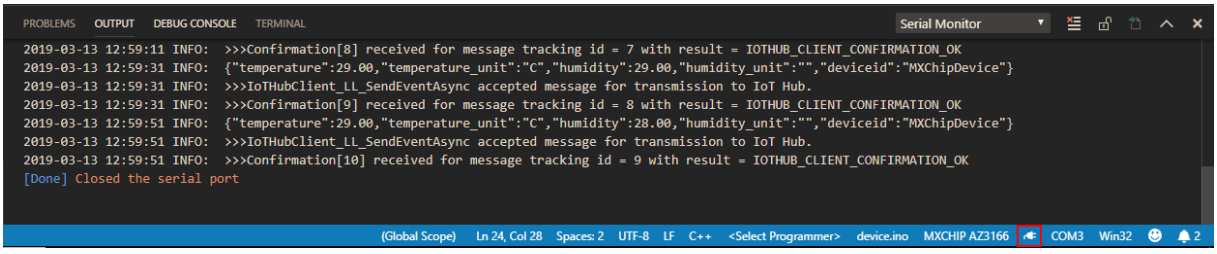
1. Go inside your Workbench application on your browser and click on your deployed **Refrigerated Transportation** application.

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.

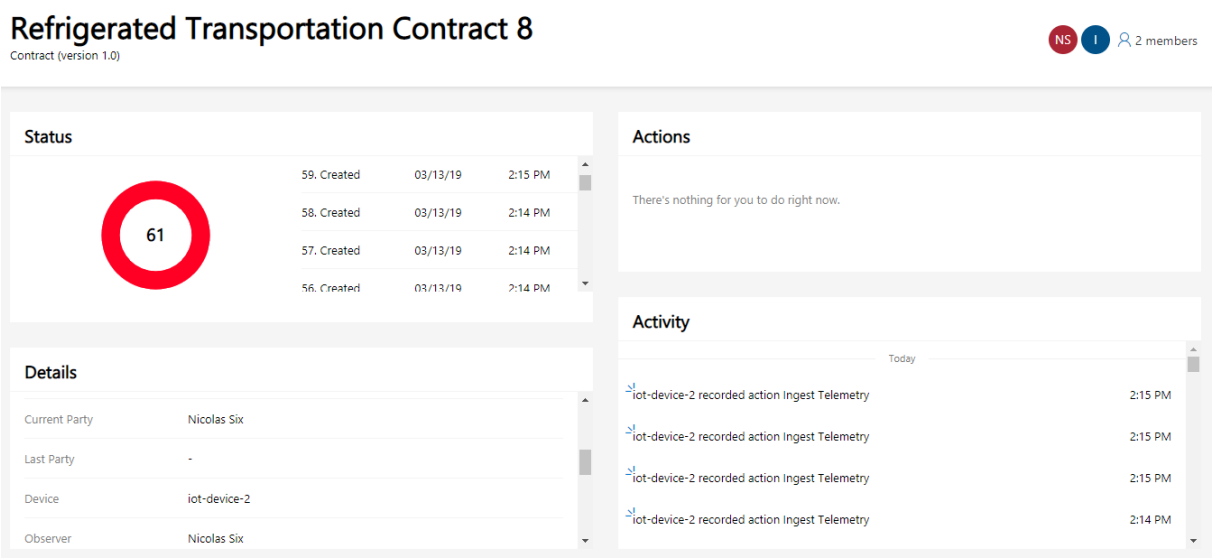
   Note: Of course, those values aren't realistic for refrigerated transportation, but I guess that you will test your device to room temperature!



3. Now that 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, the *IoT Device* should appear in it, invoking **Ingest Telemetry** method. If you heat the *Temperature sensor* on your device up to 30 degrees, you will see that the contract state will automatically change to *"Out-of-compliance"*.



The MXChip is now working, and it shows a great representation of what could be a good 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.

Azure Sphere MT3620

Overview and prerequisites

The second device that we 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. You can also find more samples on Seeed website.

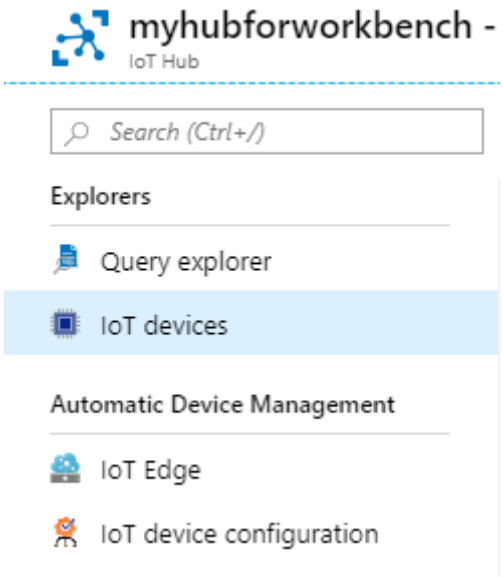
For setup this solution, you will need to:

- Get an Azure Sphere (available here), a MT3620 Grove Shield (here) and a Temperature & Humidity Sensor (here) 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*.
- Download Visual Studio for free if you choose Community Edition (at this link).
- Download the Azure Sphere SDK Preview for Visual Studio, available here.

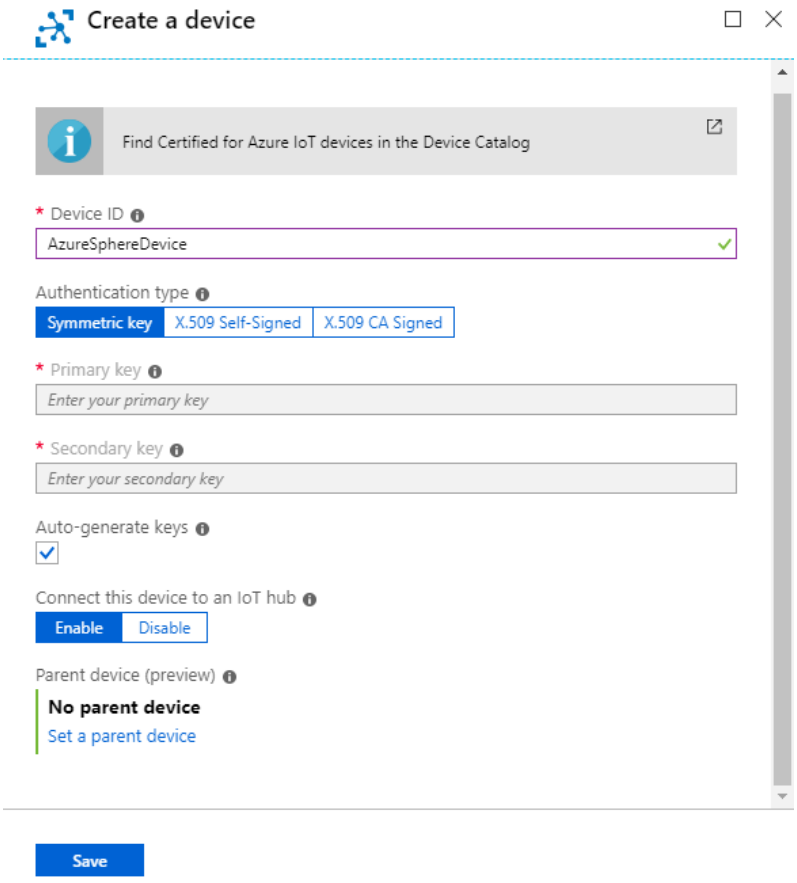
Set up the device

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*.

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**.



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, we need to link your device ID to an existing user account in your *Azure AD tenant*. We are going to use the user *iot-device-3@<your tenant>.onmicrosoft.*com that we'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**.

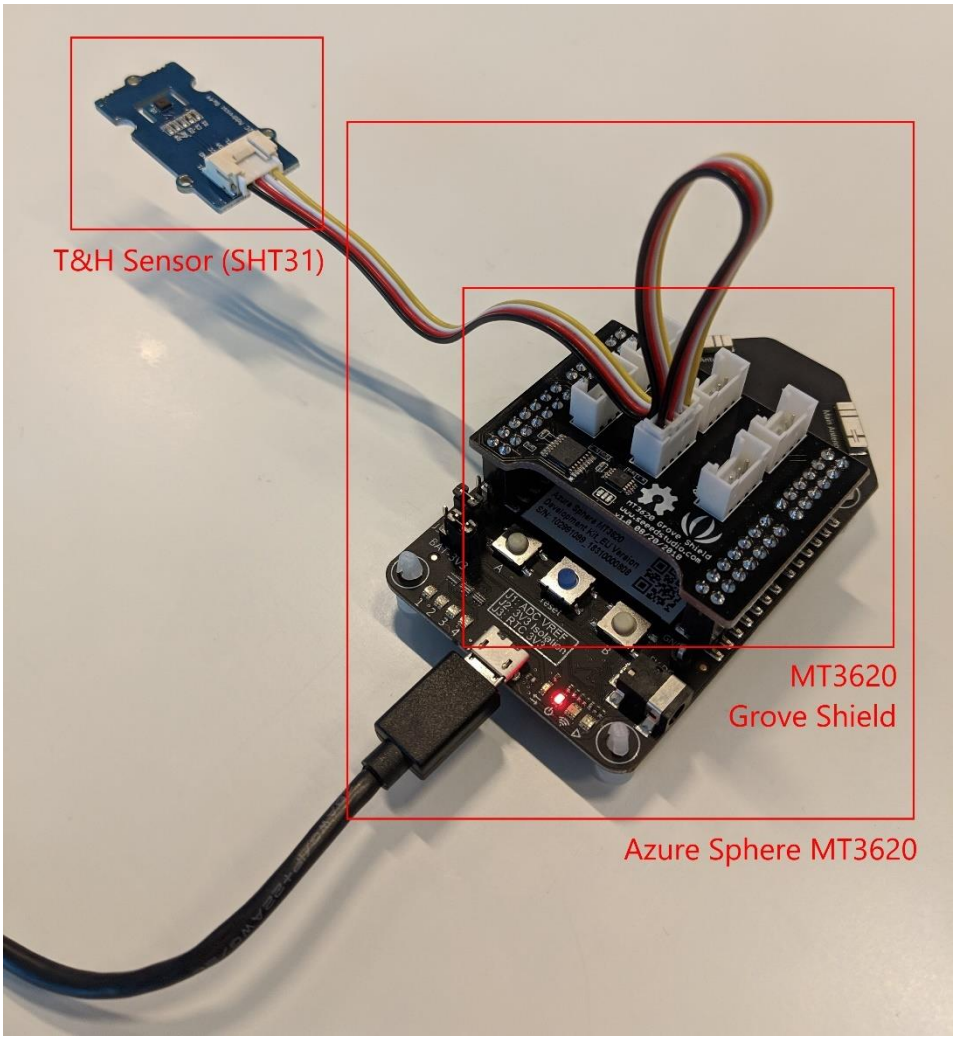
5. Log in using your credential defined when you created your Workbench.

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

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 *azsphere device recover* into the terminal, to update the Azure Sphere OS. 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 *AAD* tenant. Type the command *azsphere login* into the terminal. You will have to login with your Microsoft account, do it.

5. You will get a list of available *AAD 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 *azsphere device claim* to link your device with the selected *Azure AD tenant*.

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.

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, we 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>
```

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"
```

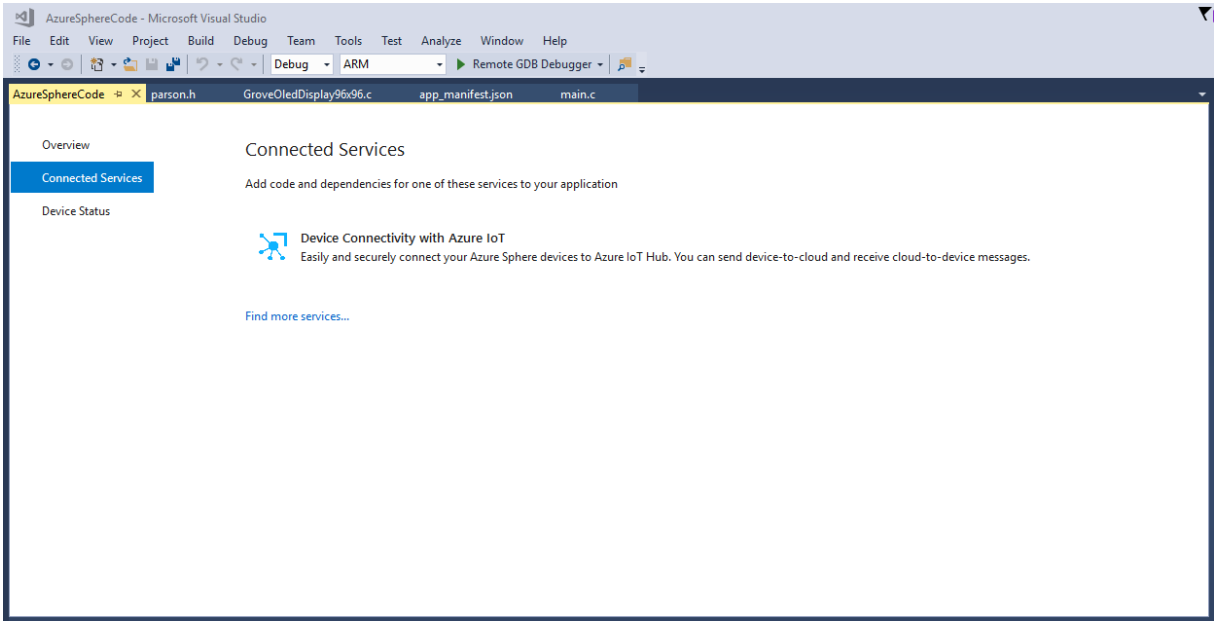
It typically takes several seconds for networking to be ready on the board, but might take longer, depending on your network environment.

Set up the project

Now that the board is ready, our next goal will be to open the Visual Studio project, compile the project code and send it to our *Azure Sphere*.

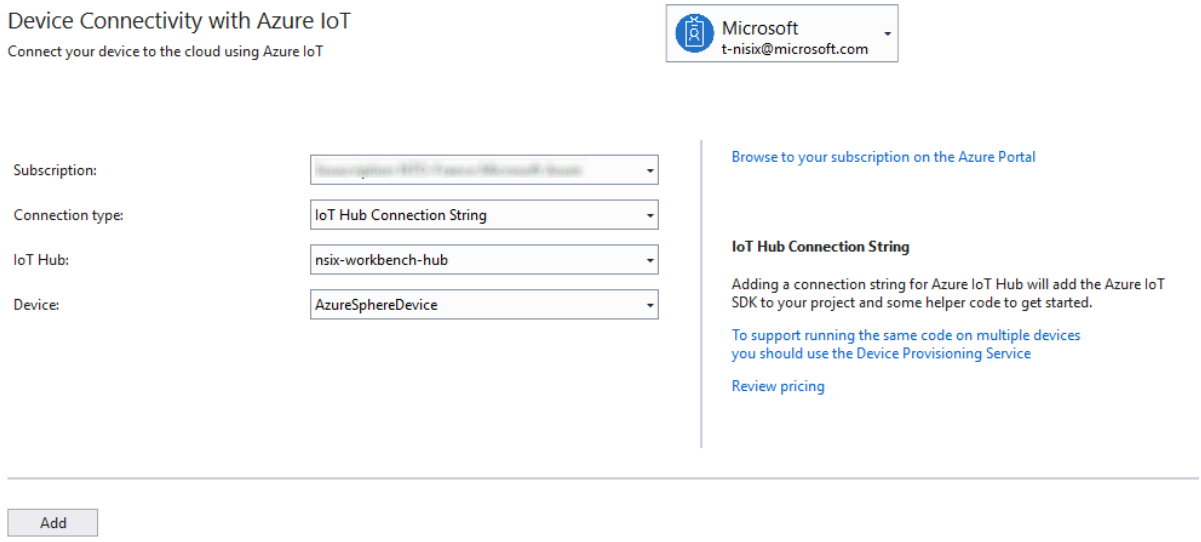
1. In VSCode, open the **AzureSphereCode.sln** which is inside **refrigerated-sc-sample > IoT setup > azure-sphere-device** directory.

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 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.



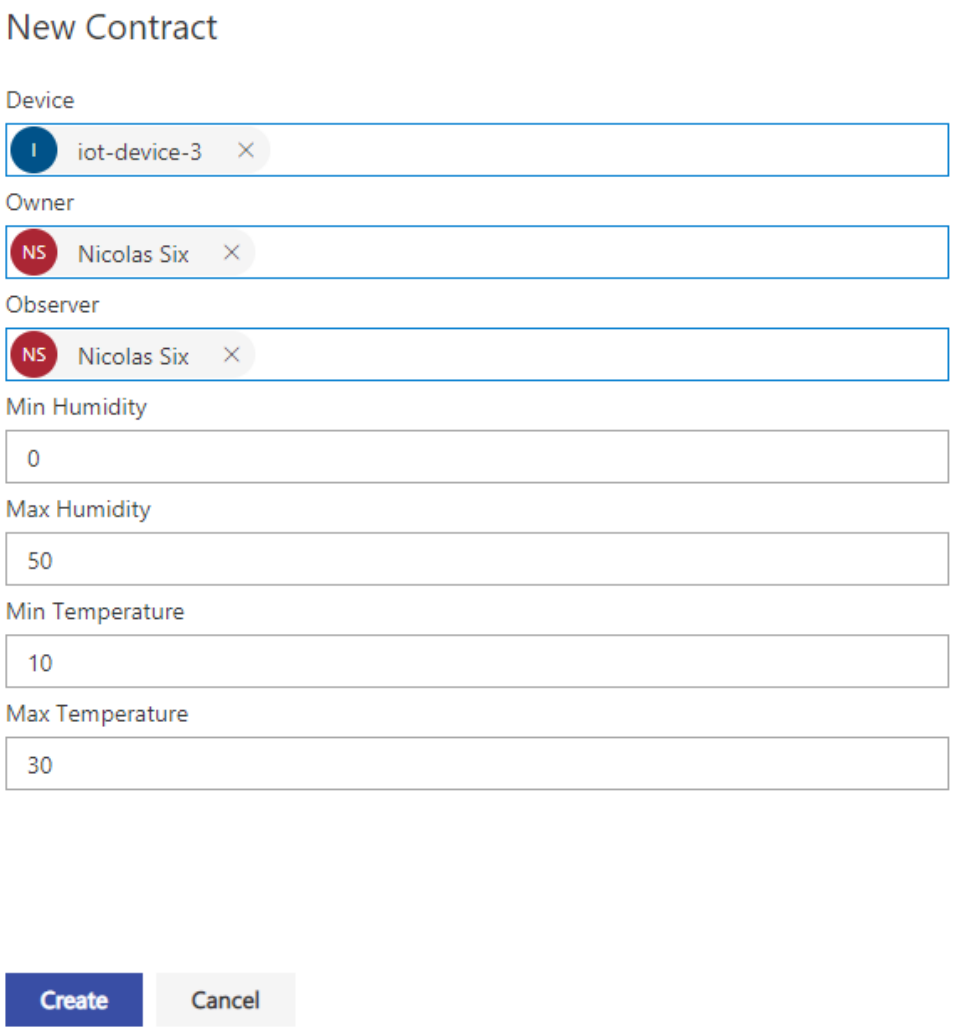
4. Right-click on **References** again, and click on **Add Reference ...**. Check the box linked to *MT3620_Grove_Shield_Library* and click on OK.

Test the device

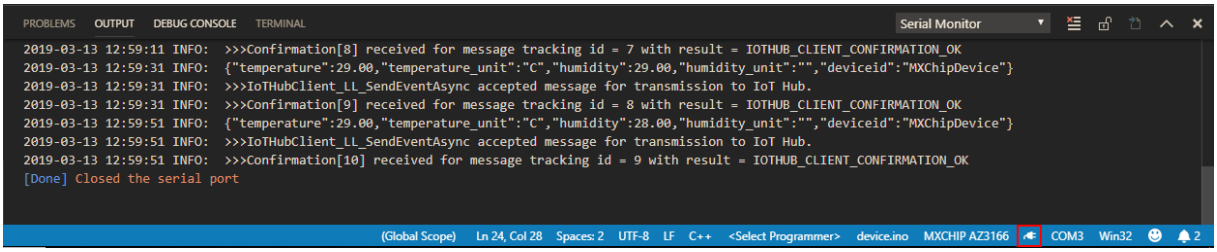
1. Go inside your Workbench application on your browser and click on your deployed **Refrigerated Transportation** application.

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.

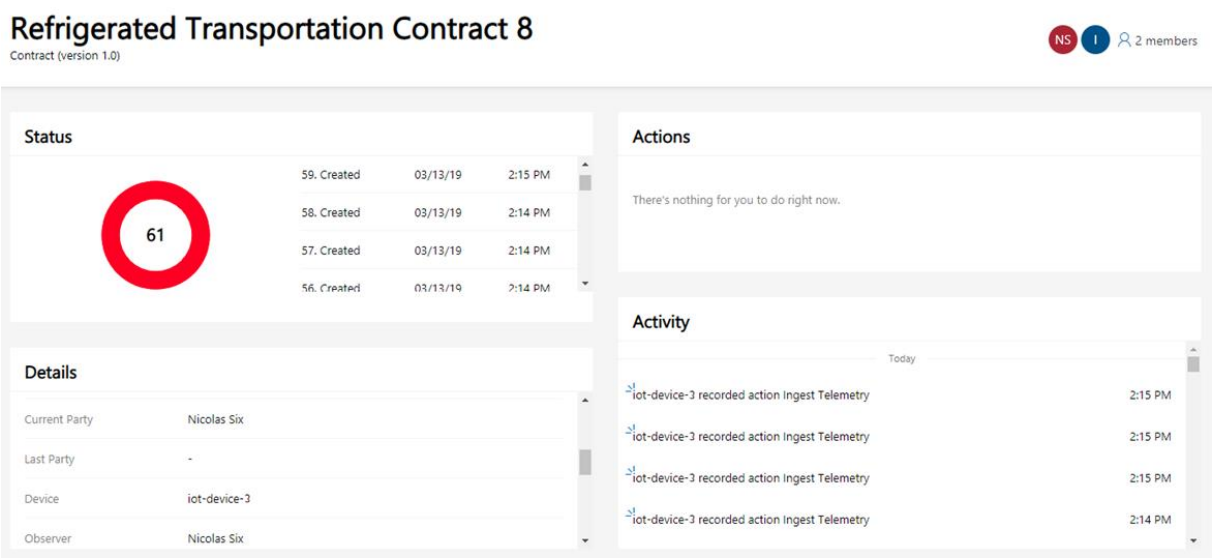
   Note: Of course, those values aren't realistic for refrigerated transportation, but I guess that you will test your device to room temperature!



3. Now that 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, the *IoT Device* should appear in it, invoking **Ingest Telemetry** method. If you heat the *Temperature sensor* on your device up to 30 degrees, you will see that the contract state will automatically change to *"Out-of-compliance"*.



The Azure Sphere is now working, and it shows a great representation of what could be a good 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.

Raspberry Pi

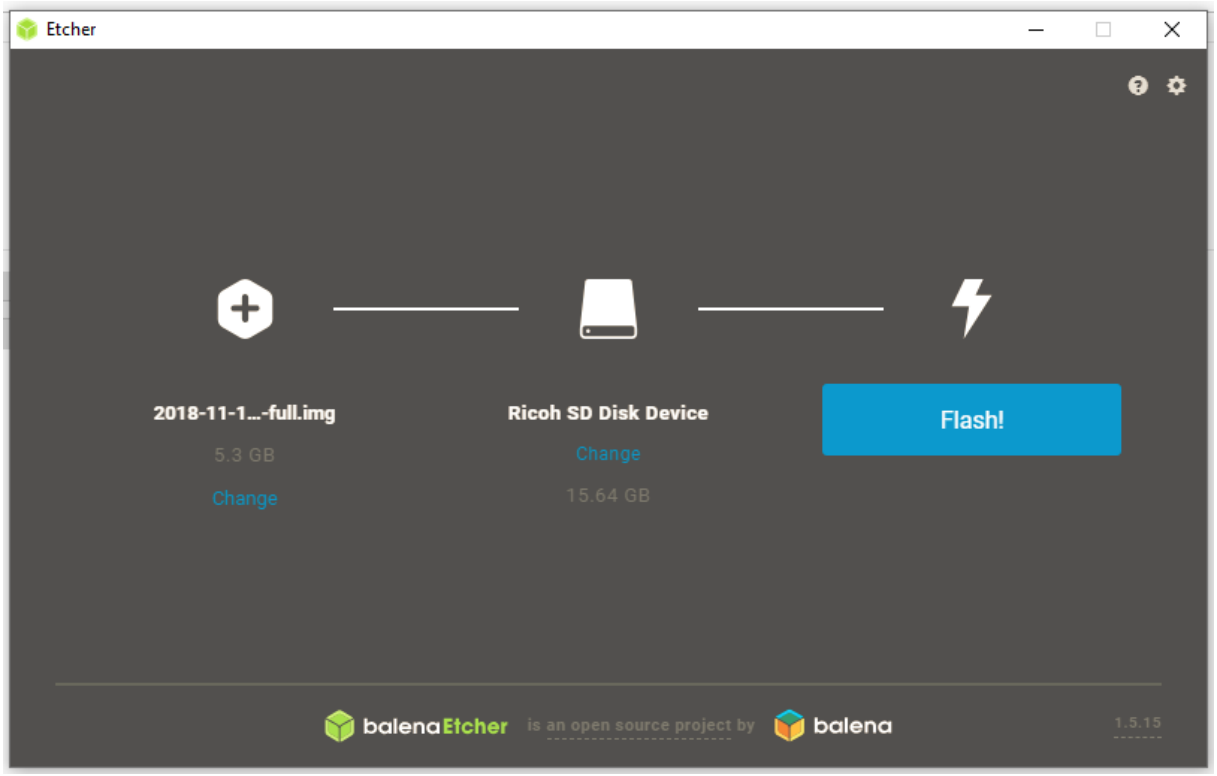
Overview and prerequisites

The last device which will be introduced here is one of the most known device in the IoT world, the *Raspberry Pi*. This device is different from the 2 previous devices introduced here, where 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 working microcomputer.

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

First, we have to create a bootable Micro SD card for our *Raspberry Pi*. If you didn't downloaded it yet, go on Raspbian website and download the disk image of *Raspbian*. Also, you will need a flash software to write the disk image to your Micro SD card. For that, we're going to use Etcher, available here.

Insert your clean 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**.

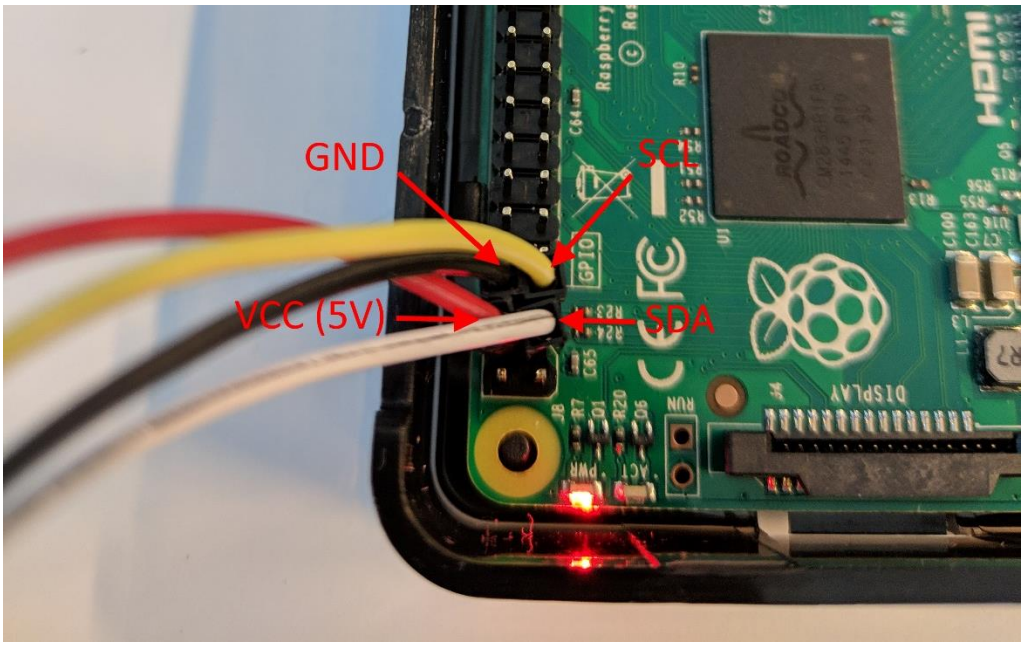


Wait for the program to finish flashing, and eject your Micro SD card from your computer when everything is done.

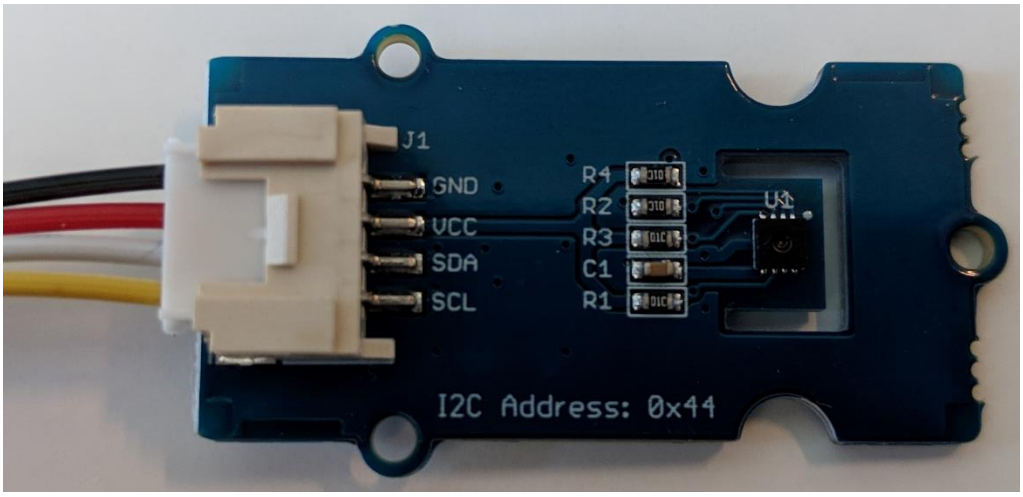
Before launching the Raspberry Pi for the first time, we need to plug the sensor to the card. For this guide, I chose the Grove SHT31 Temperature & Humidity sensor, but another solution is to use the Adafruit BME280 sensor (you can find more information here).

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 we will deploy on our device later, as the solution introduced in this guide is only one of the many possible solutions. We could have also used the Windows 10 IoT core as the Raspberry OS for example.

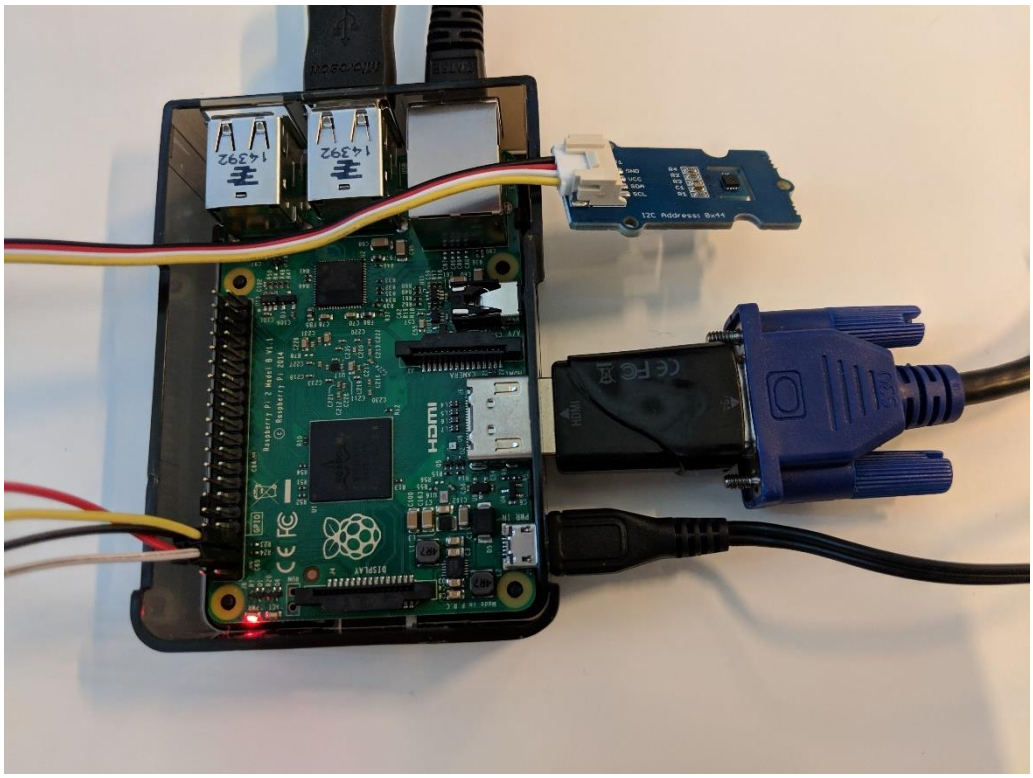
The SHT31 sensor can be plug on a board compatible with I2C, like the Raspberry Pi. To connect the sensor to the board, plug the 4 male-to-female cables to the 4 pins of the SHT31 sensor, then plug the other side of the cables to the board like the picture below.



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 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 our solution.

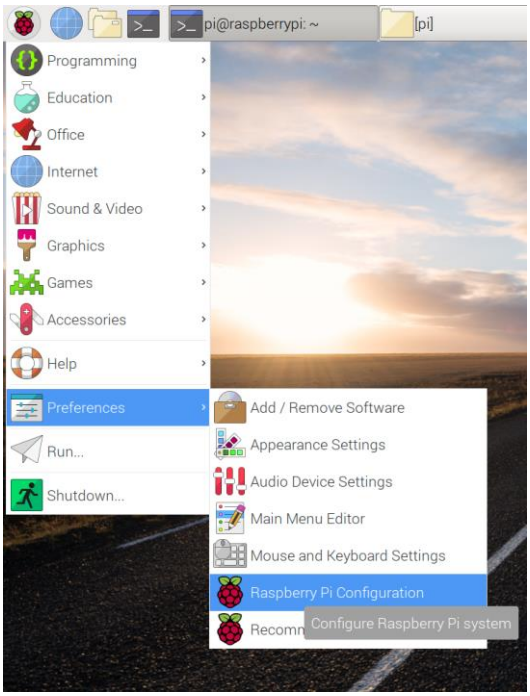


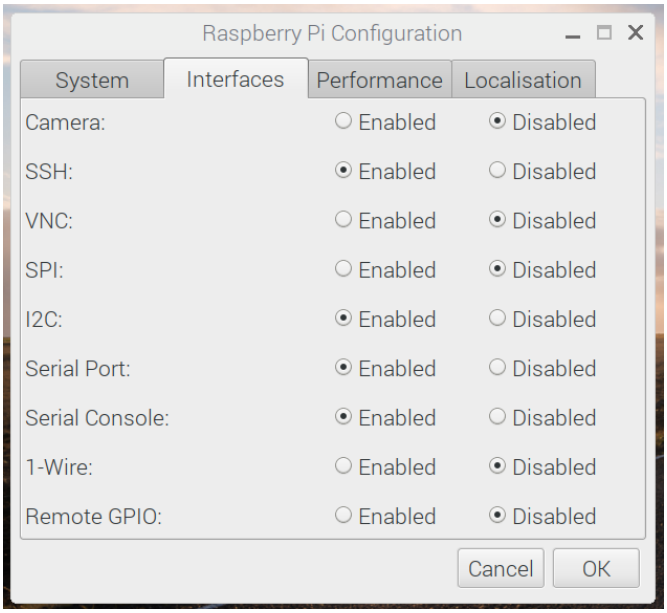
Setup the device

Now, we can start configuring the Raspberry Pi.

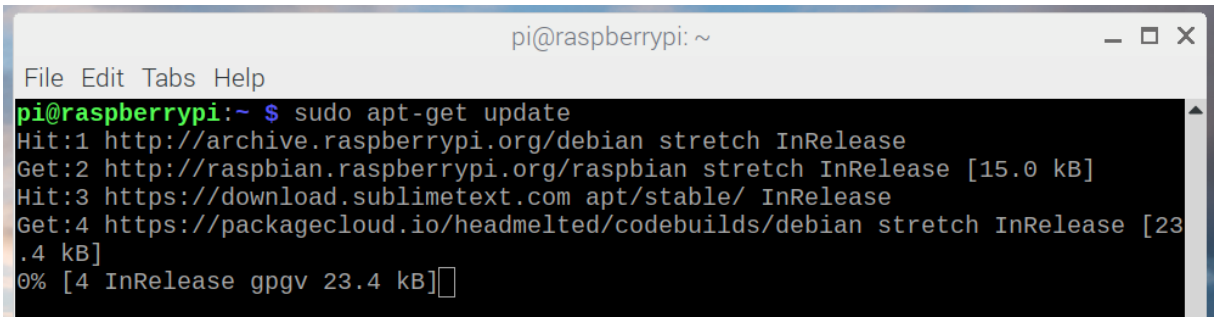
1. Click on the raspberry at the top of the screen, then **Preferences** > **Raspberry Pi Configuration**.

2. Go in the **Interfaces** tab, and enable the I2C communication by clicking on **Enabled** radio button.

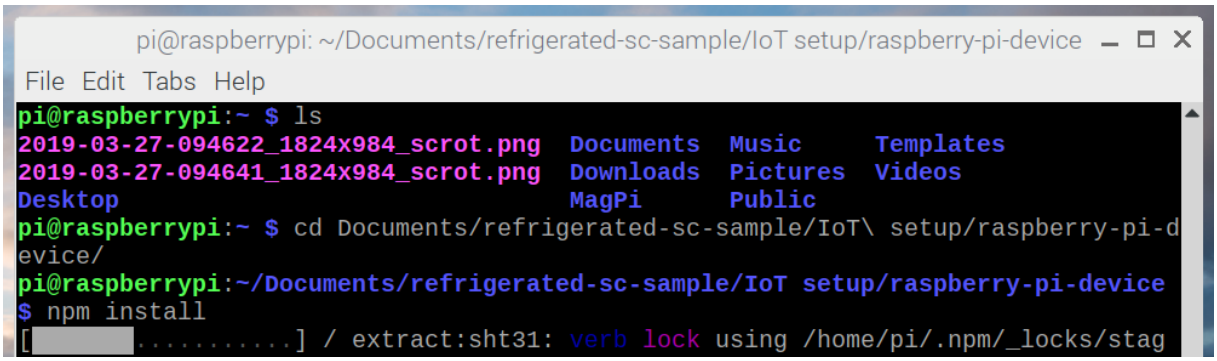




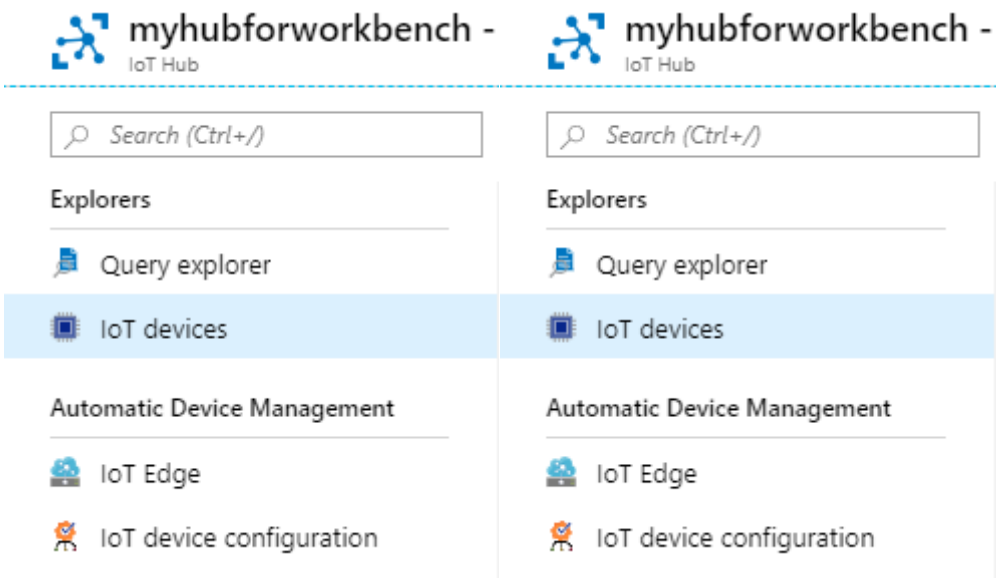
3. Open a Terminal, and type type the command *sudo apt-get update* to update your system.



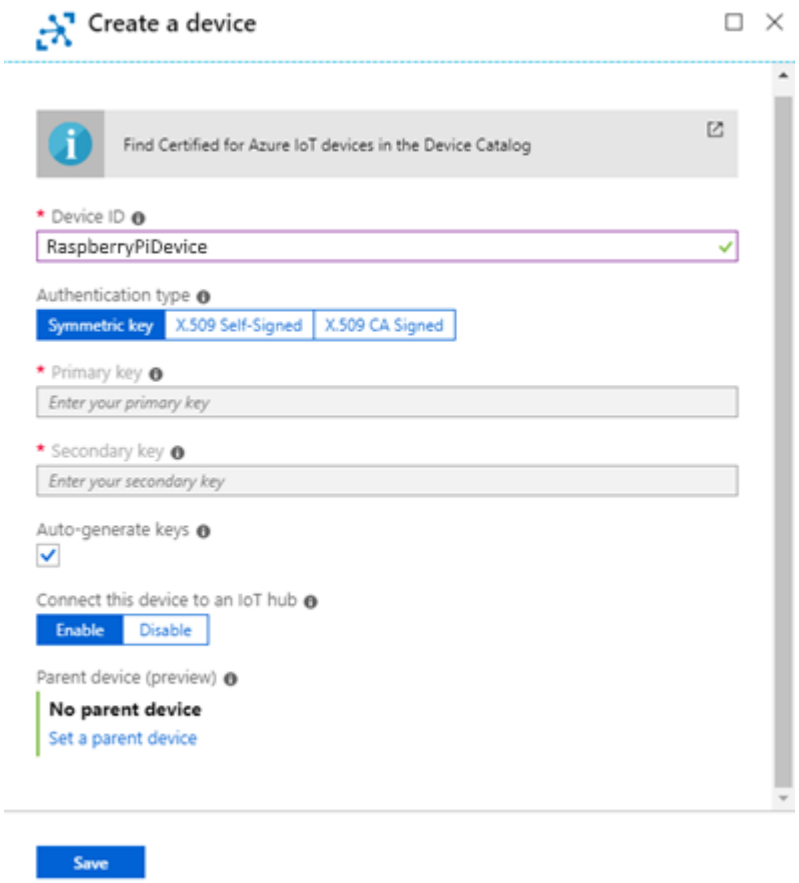
4. Go with the terminal inside the **refrigerated-sc-sample** code file (which can be get or pulled from here), then **IoT Devices** > **raspberry-pi-device**. Type *npm install* to install required packages for our application.



5. To test the program, we need to provide a Connection String which can be get from the IoT Hub, so we will create an identity for our Raspberry Pi on Azure AD. 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 **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.

7. 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.

8. Finally, we need to link your device ID to an existing user account in your *Azure AD tenant*. We are going to use the user *iot-device-4@<your tenant>.onmicrosoft.*com that we'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.

9. Click on it, and inside it, click on **Query Editor**.

10. Log 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'*.

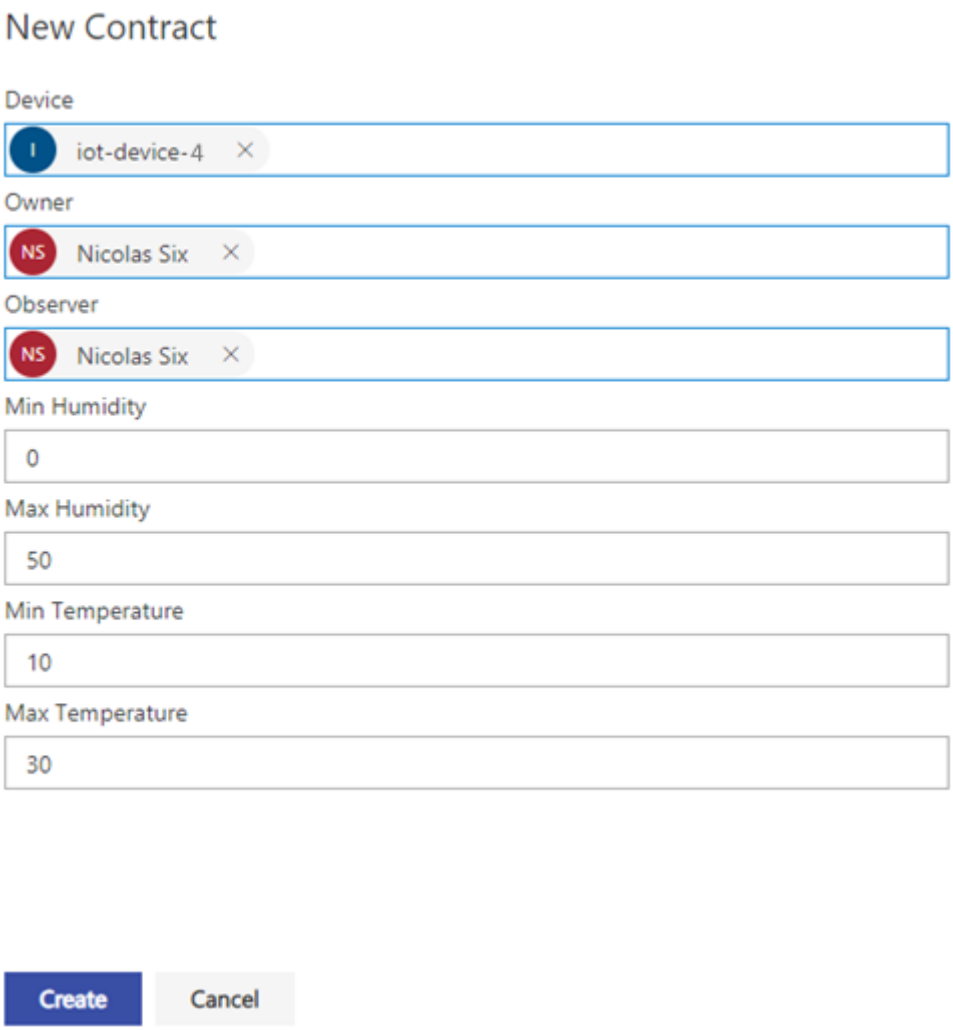
11. Check that the command has been correctly executed by looking into the messages tab below the *Query Editor:* it should be *Query succeeded: Affected rows: 1*.

Test the device

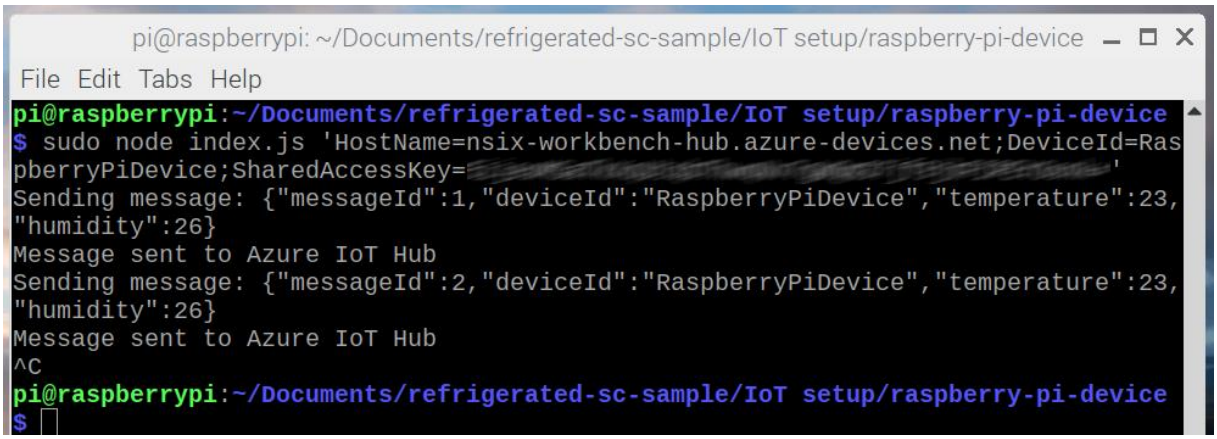
1. Go inside your Workbench application on your browser and click on your deployed **Refrigerated Transportation** application.

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!

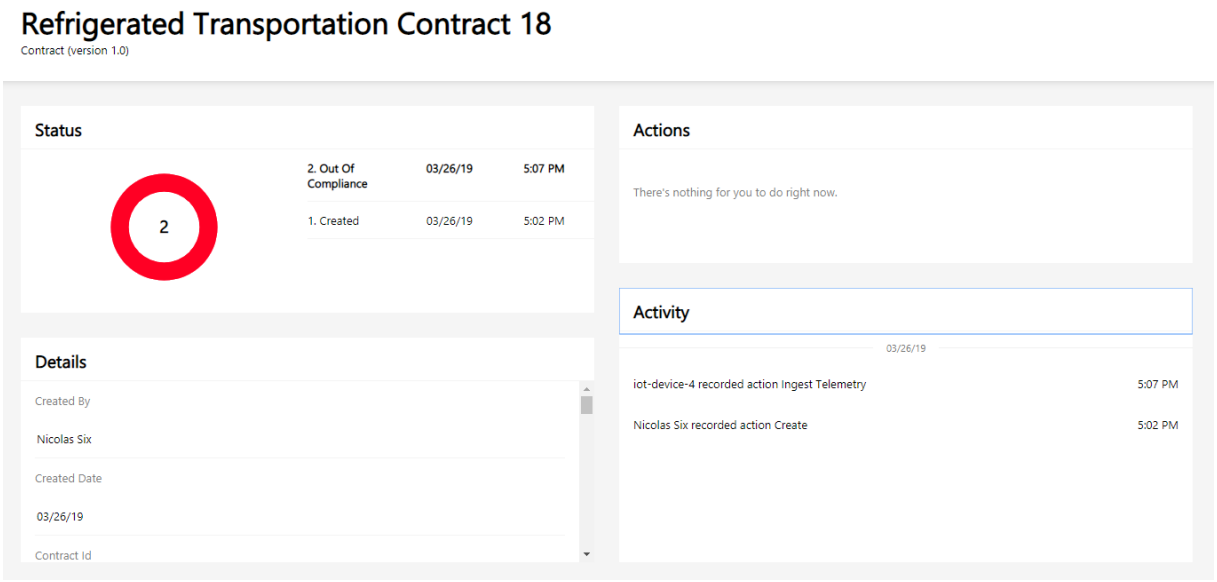


3. To test the device, you need to type in your *Raspberry Pi* terminal the command *sudo node.js index.js '<Your connection string>'*, by replacing the connection string with your own that you got from the Hub. If the connection is established, the program will send telemetry to your Hub and you will 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, you will see that the contract state will automatically change to *"Out-of-compliance"*.



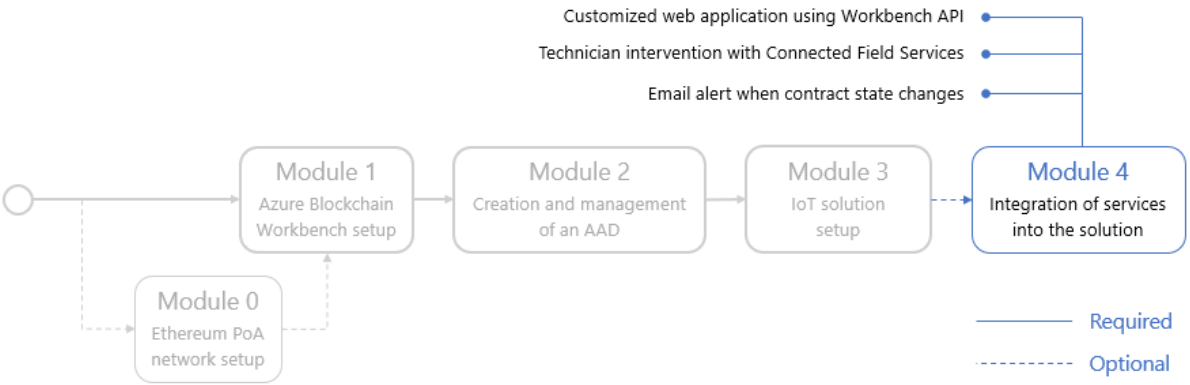
Module 4 : Integration of services into the solution

Overview

Since the beginning, we've built a working solution, with an operational blockchain and an additional layer which is the *Workbench*.

We've also connected multiple IoT devices, which shows that any connected device can be used as a temperature sensor if they can establish a connection with our *IoT Hub*, and we saw how to manage *Workbench* users and user groups.

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



In this module, we 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, a Dynamics 365 module

Email alert when contract state changes

A good idea would be to alert counterparties of the contract if 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.

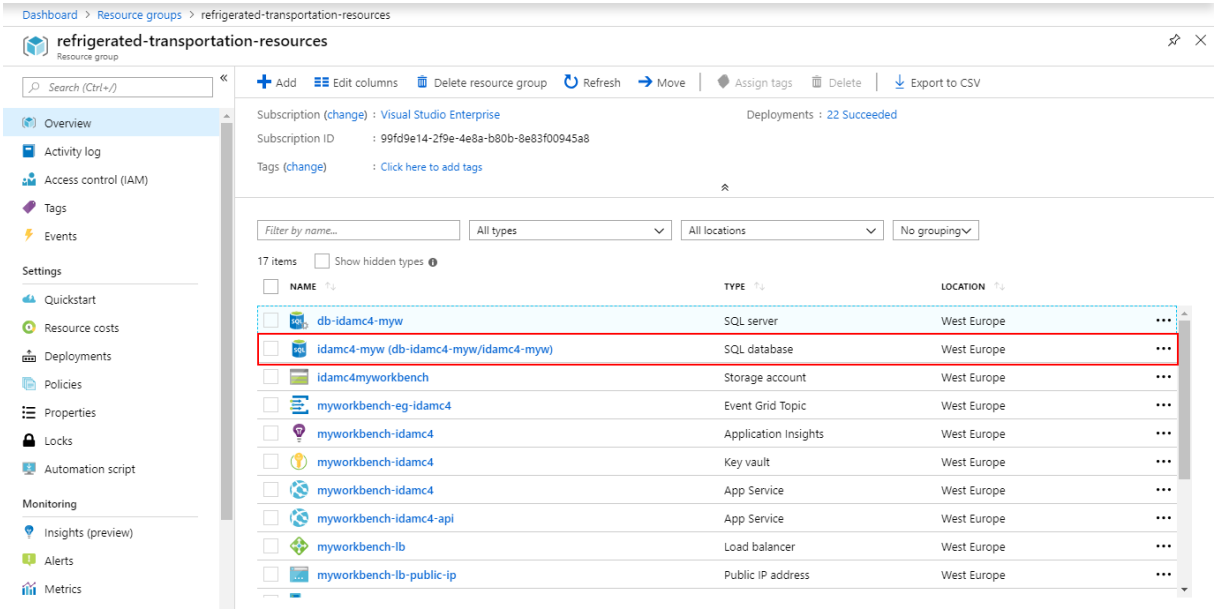
The best way to do that is to create a new *Logic App*, which takes *Workbench* messages as an input from its *Egress Topic*, and check if a state changes and if yes, which one. Then, the app send an email to alert if something important happened.

Also, sending an email to contract parties when the counterparty in charge of the shipment changes could be a great fit for our guide. We're going to implement both of those alerts inside our Logic App.

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

Getting contract parties emails

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. Log 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 before (or grab it here if you don't have it yet), then open the **getContractParties.sql** file which is inside **Integration** directory, 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 call a step "Get contract parties" instead of the default "Execute a stored procedure".

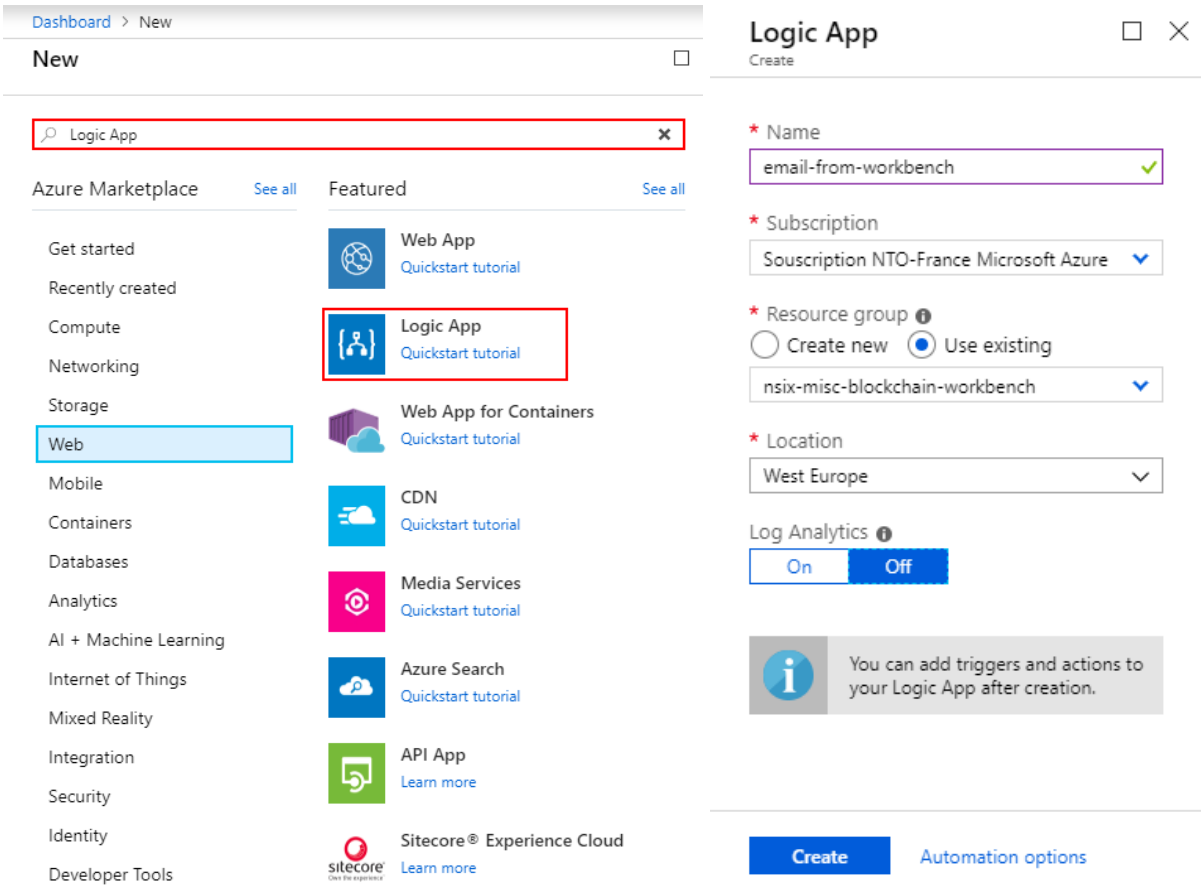
If you want to do that, keep in mind that you should do it when creating the step, and not at the end of the Logic App creation, because it's impossible to rename a step if some data from the step is used somewhere else. In this guide, we will keep default step names.

1. Log in to the Azure portal.

2. Choose +**Create a resource**, then choose **Web**.

3. Click **Logic App** from the list on the right.

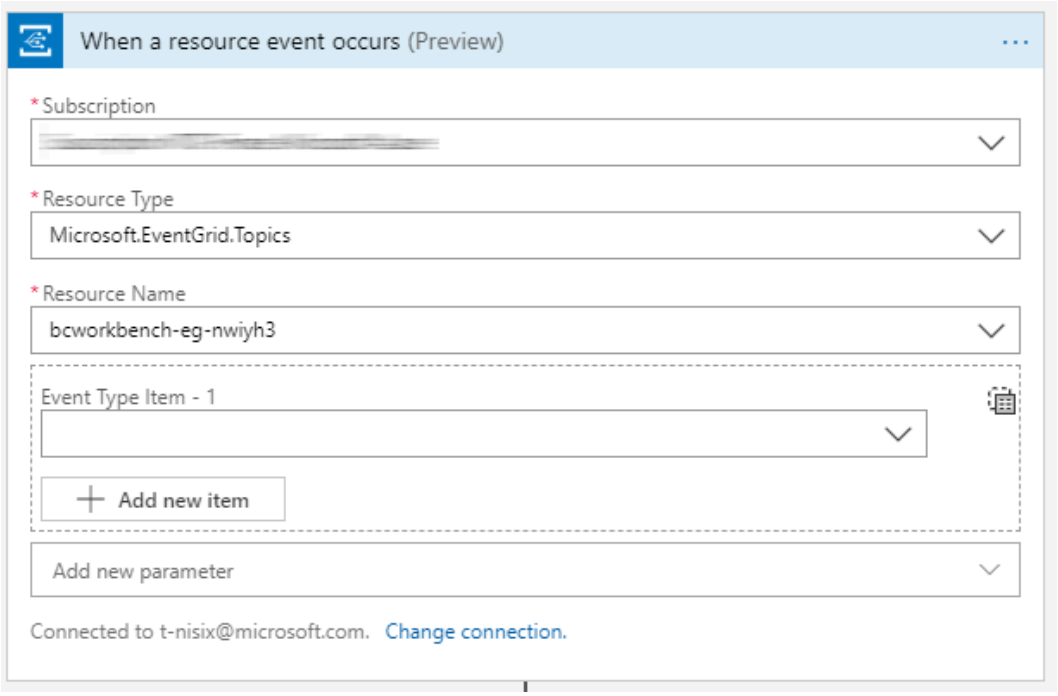
4. Fill the configuration tab.



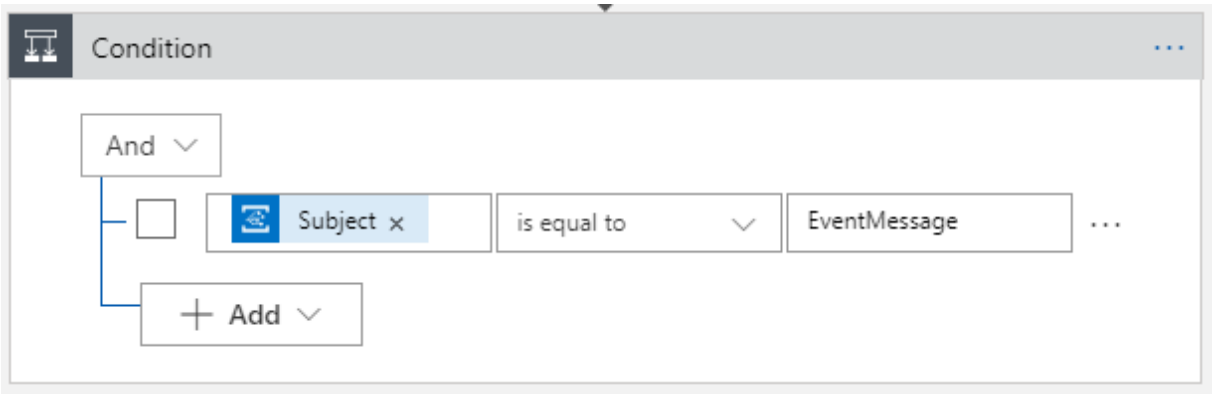
Setting	Description
Name	This is the name of your future app.
Subscription	Select the subscription to use for your *Logic App*.
Resource group	You can create a new resource group or use an existing one. To create a new one, click **Create** new and fill in the name you want to use. To use an existing 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 **Service Bus** and search for **When a resource event occurs,** then click on it.

6. Fill the form by selecting your subscription, the *Resource Type* called **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**. Click on the left value field, and select **Subject** from the *egress queue*, and in the right field type **EventMessage**. It means that we're going to check every message from the *Workbench* and if the subject is *EventMessage*, something happened inside the *Workbench* and we need to check if it requires an email.



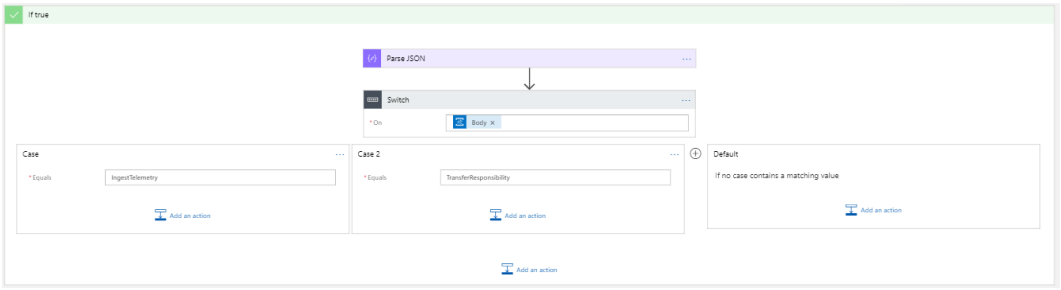
8. Two tabs should have appeared, *True* and *False*. Let the *False* one by default, as we're not going to take an action for messages which are not *Event Messages*. In the True tab, click on **Add an Action**. Click on **Data Operations** then **Parse JSON**. Click on the field **Content** and select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

9. 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 2 equal fields type respectively *IngestTelemetry* and *TransferResponsibility*.



   Those 2 cases represent the 2 possible paths that our 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.

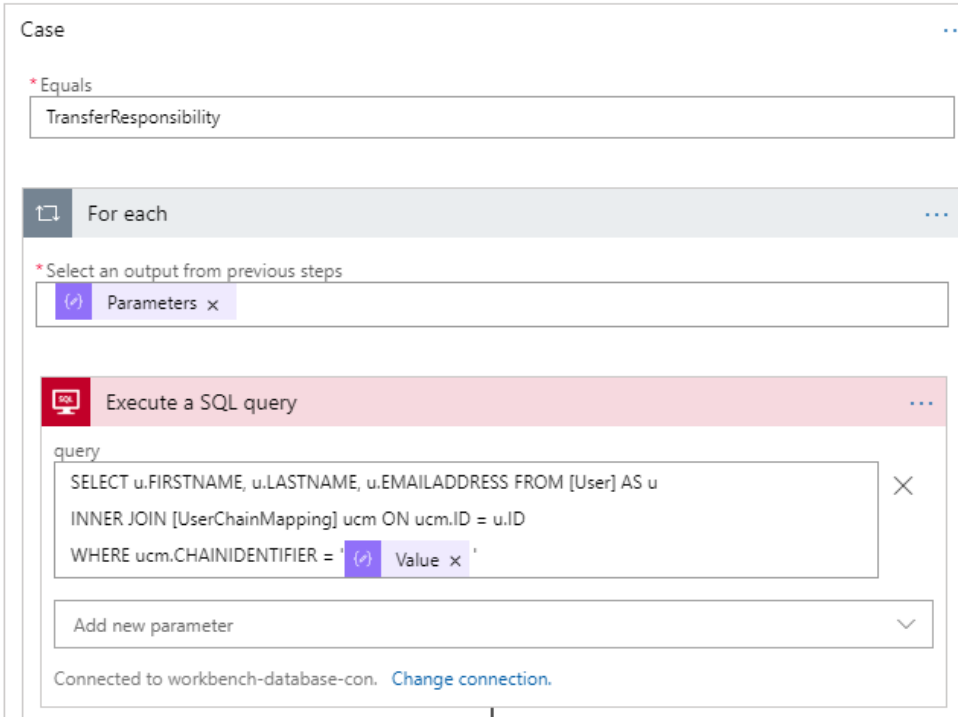
   We will complete both of the cases, but for now we will start with the *TransferResponsibility* case.

10. 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.

11. 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:

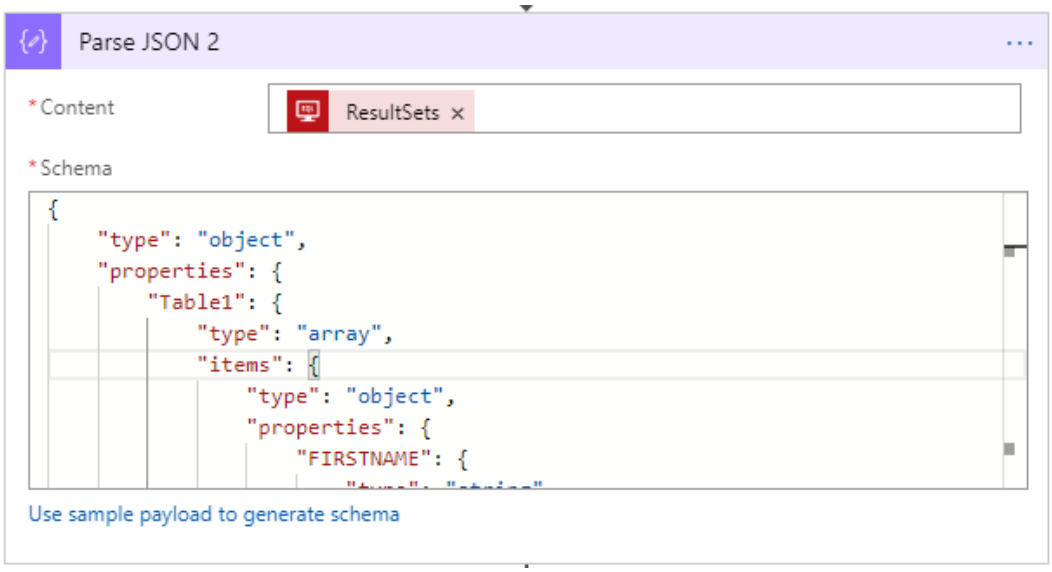
    SELECT u.FIRSTNAME, u.LASTNAME, u.EMAILADDRESS FROM [User] AS u INNER JOIN [UserChainMapping] ucm ON ucm.ID = u.ID WHERE ucm.CHAINIDENTIFIER = '';

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

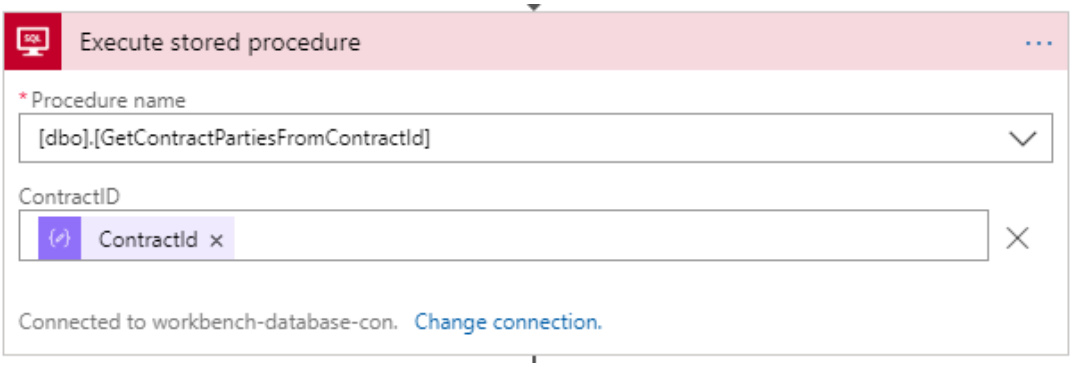


12. Click on **Add an Action**. Click on **Data Operation** then **Parse JSON**. Click on the **Content** field and select **ResultSets**. Paste the following content into Schema:

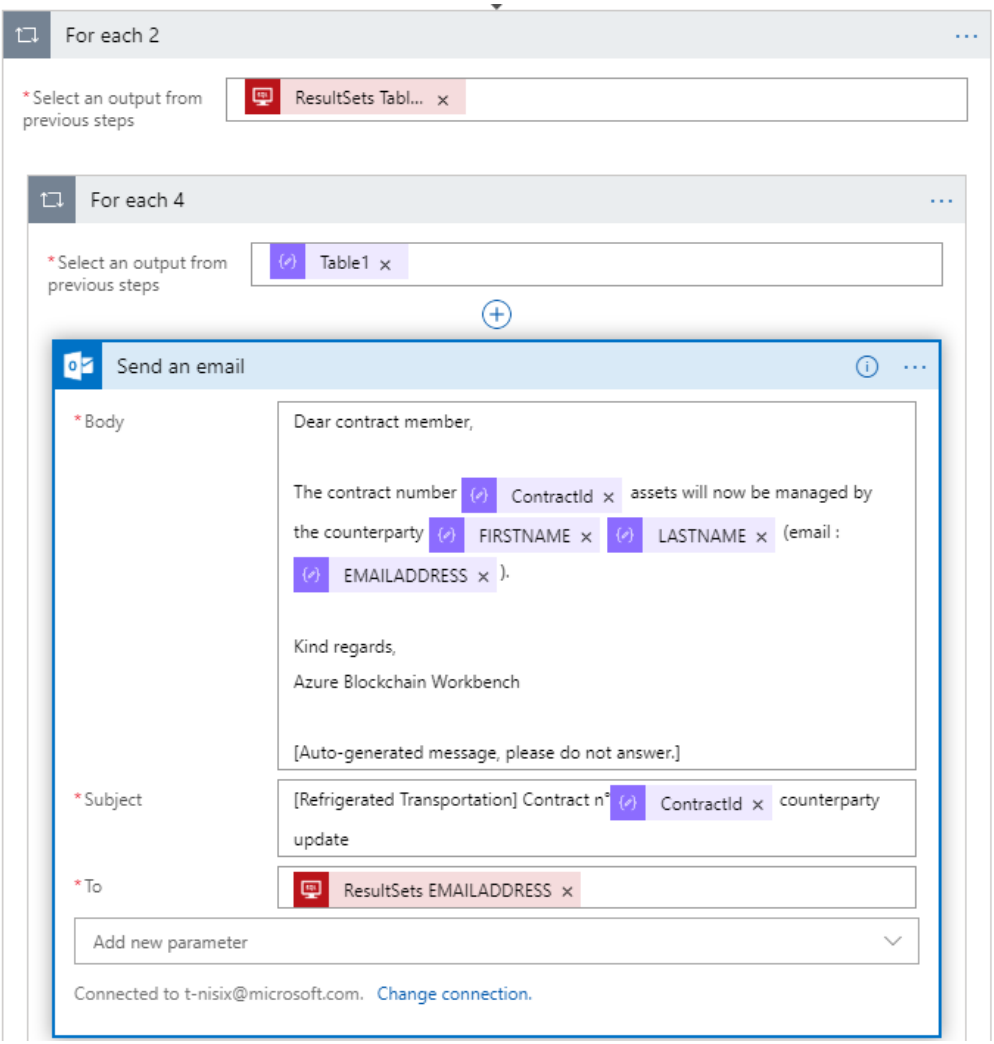
```
{
    "type": "object",
    "properties": {
        "Table1": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "FIRSTNAME": {
                        "type": "string"
                    },
               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



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



14. 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.

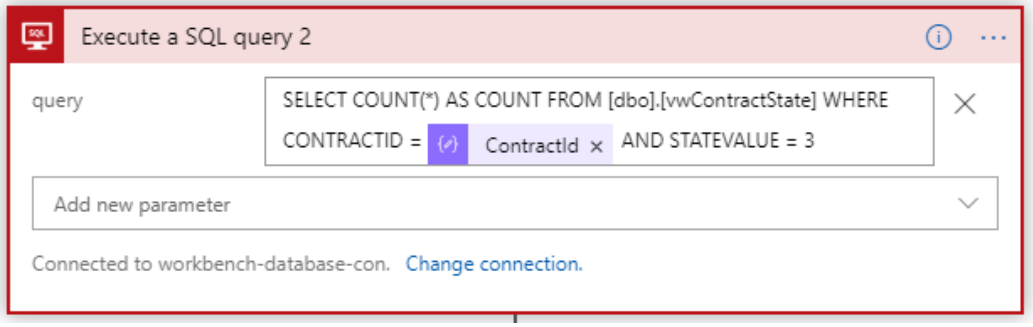


15. Go back to the **IngestTelemetry** case. Click on **Add an action**, choose **SQL Server** then **Execute a SQL query**. Add a new parameter **query** and inside the new query field, type this request:

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

    Replace the <> by the dynamic value ContractId as we 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), we’re sure that the contract is “Out-of-Compliance” as his state can’t change anymore.

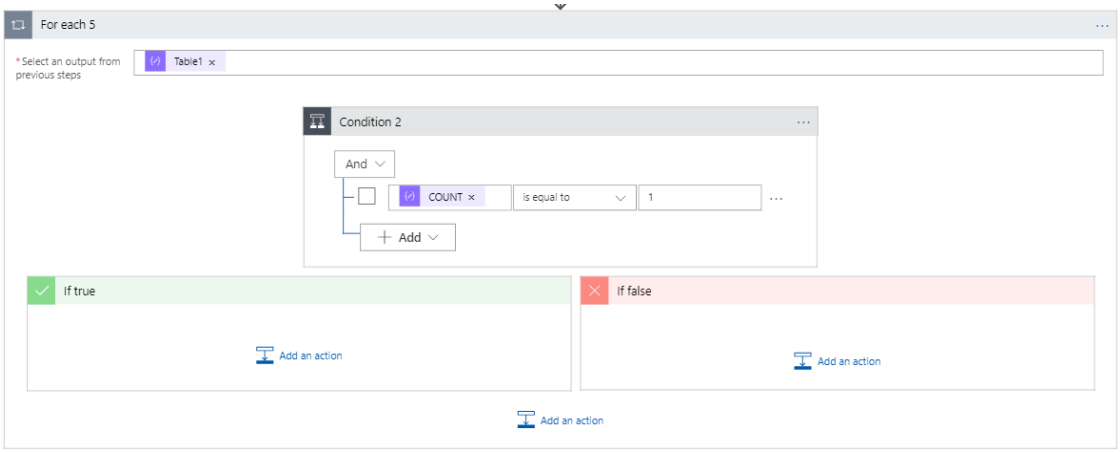
    So, the request returns 0 (if the contract is not “Out-of-Compliance” or 1.



16. 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**:

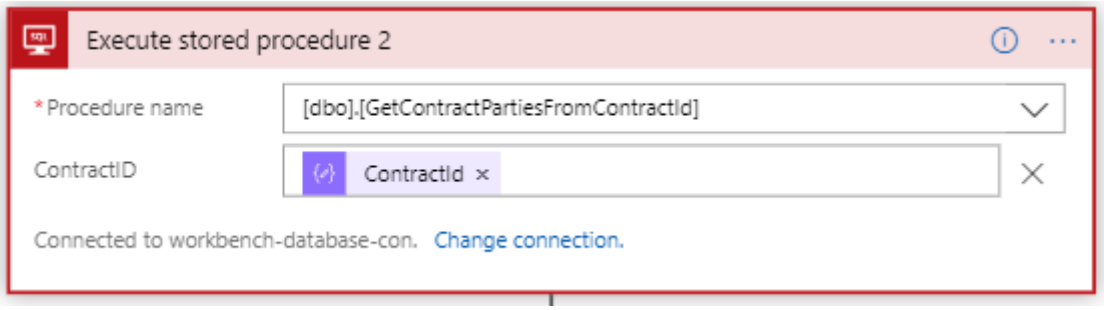
```
{
    "properties": {
        "Table1": {
            "items": {
                "properties": {
                    "COUNT": {
                        "type": "integer"
                    }
                },
                "required": [
                    "COUNT"
                ],
        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

17. 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).

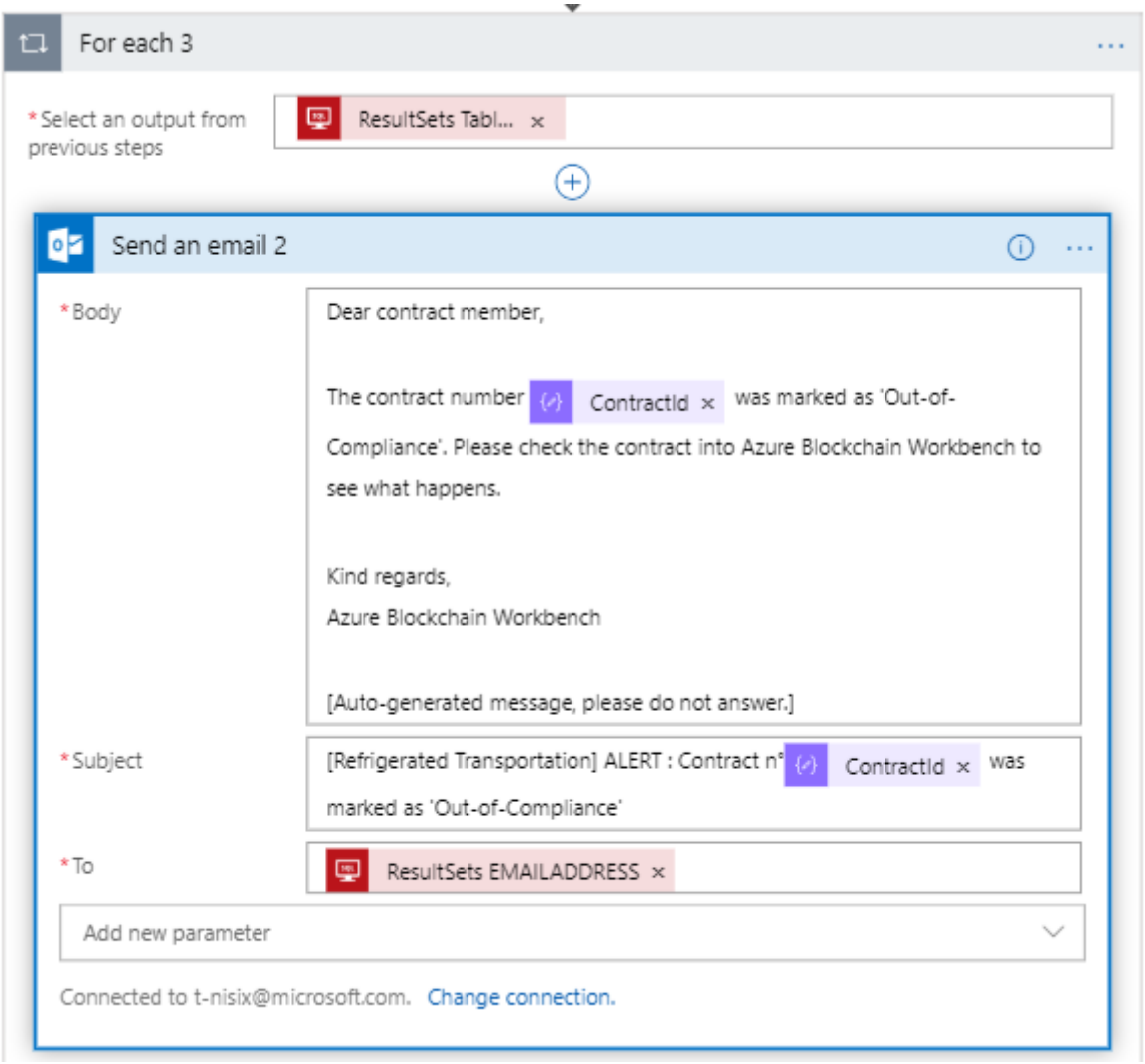


18. In the True block, click on **Add an Action** and choose **SQL Server** then **Execute a stored procedure**. Select the procedure **[dbo].[GetContractPartiesFromContractId]** previously

created, then add a parameter called **ContractID**, click on the field and select the **ContractID** dynamic value.

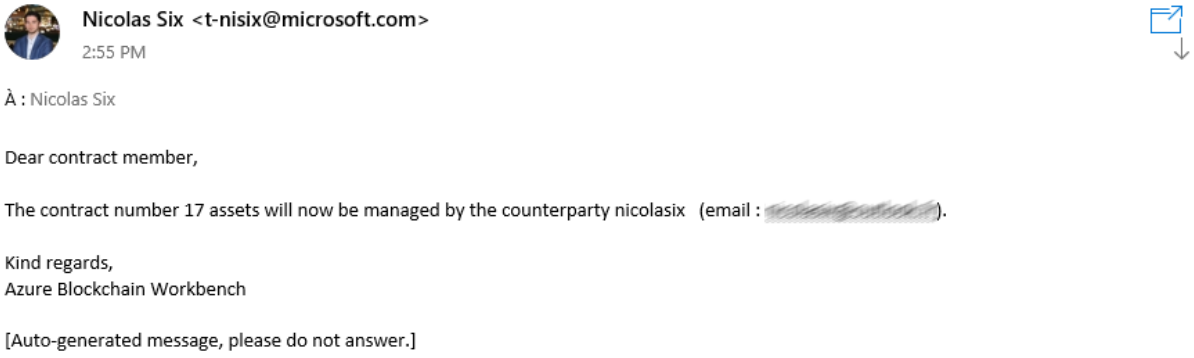


19. The final block will be the email sending. Click on **Add an Action, Office 365 Outlook** then **Send an Email**. Fill the form as the screenshot below, and keep in mind that adding the dynamic values will automatically create a *For each* which will go through the contract parties to send emails.

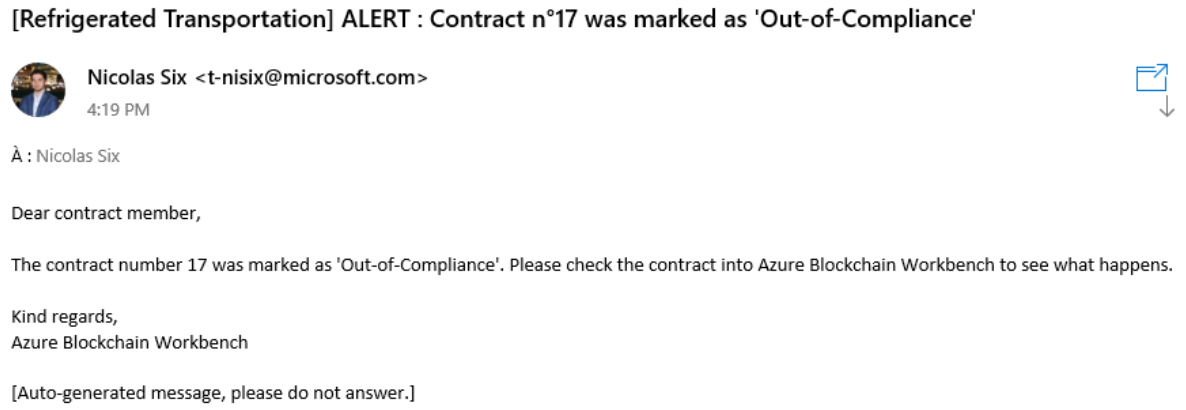


Test the application

Our 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 operational. Those 2 functionnalities work really well with the *Refrigerated Transportation scenario*, but you're free to go in depth through the Logic App documentation (available here) to develop your own Logic Apps around the Workbench or not !

Technician 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 our scenario, this is a problem because a broken device is not able to send telemetry and therefore, insure that the shipment stay at correct temperature and humidity.

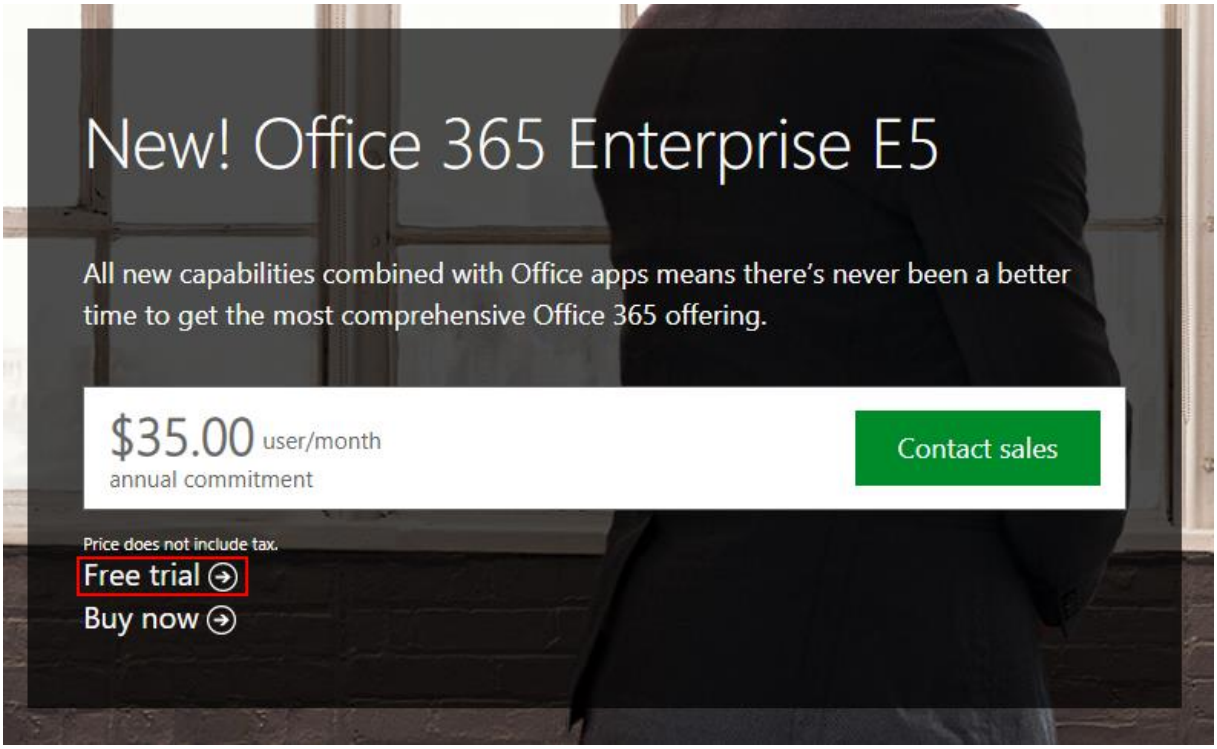
A solution to this problem could be a technical intervention, by checking if supply chain devices are correctly sending telemetry using Workbench database, and if not send a technician to repair/change the device.

Setting up Office 365 & Dynamics 365

To begin, we need to have an *Office 365* subscription, including *Dynamics 365*. 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 part is to create an *Office 365* tenant using the trial version, and do the same for *Dynamics 365* and *Connected Field Services*.

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



2. Fill the form to create your tenant and your admin user.

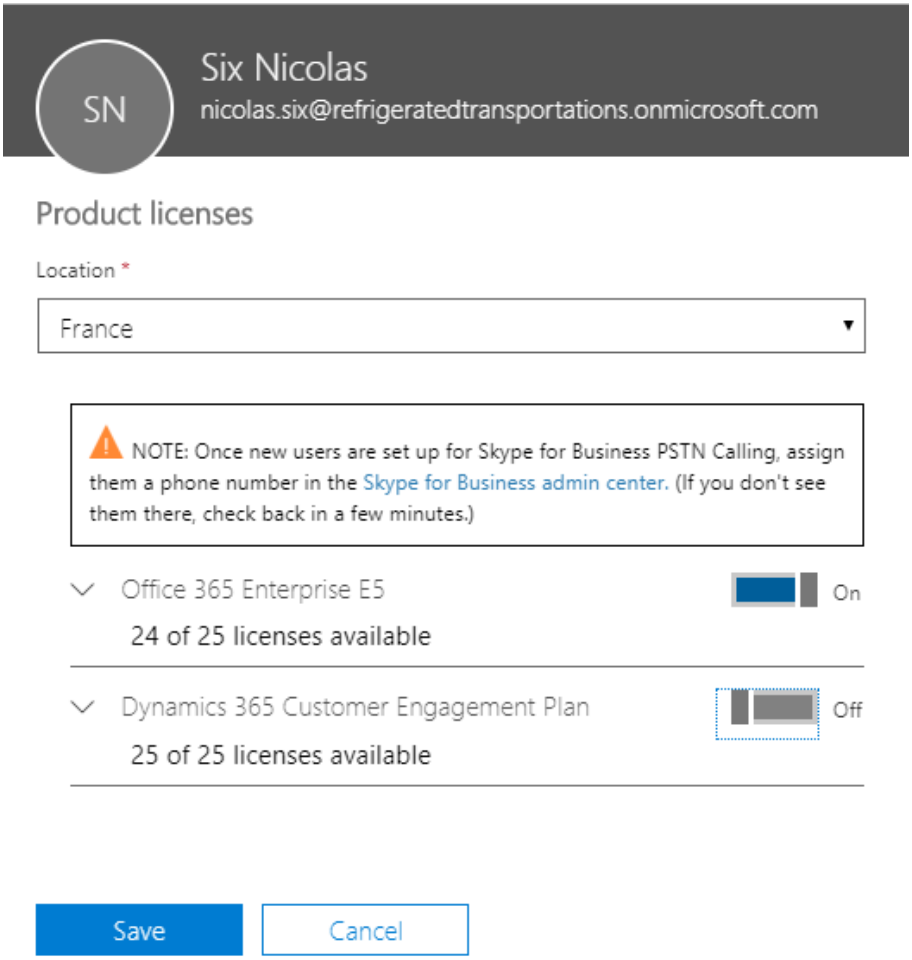
3. Go on the admin panel, and on the left navbar click on **Billing** > **Subscriptions**, then click on **+Add subscriptions**.

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.

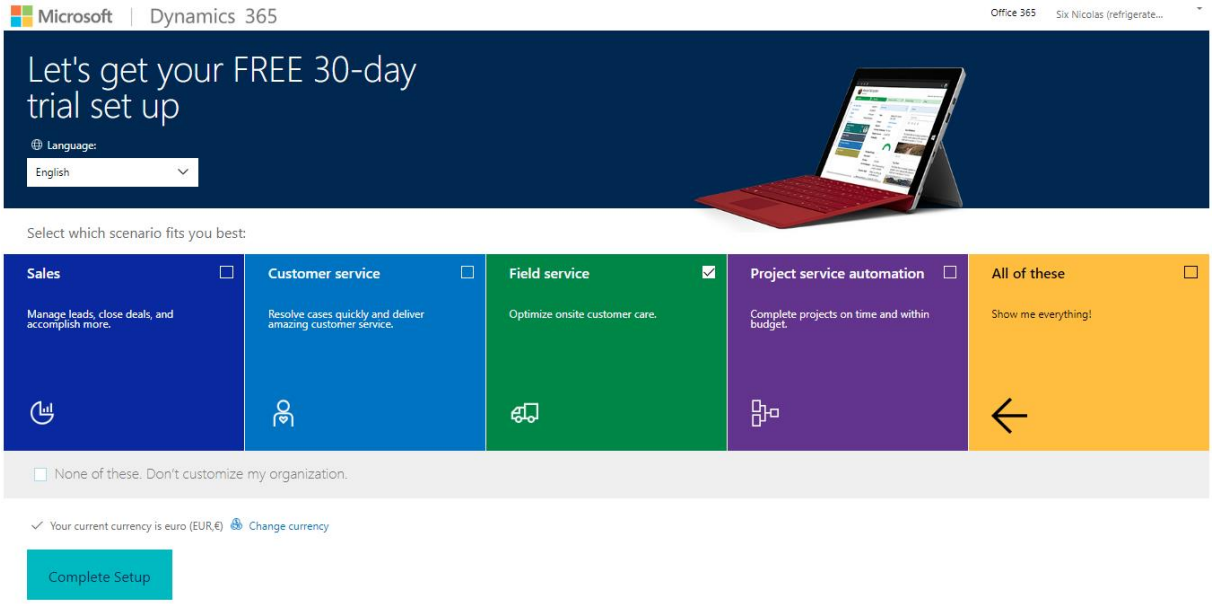
5. Click on **Users** > **Active users**, then click on yourself and in your profile, on **Edit** next to the License section.

6. Turn on **Dynamics 365** for your account and click on **Save**.

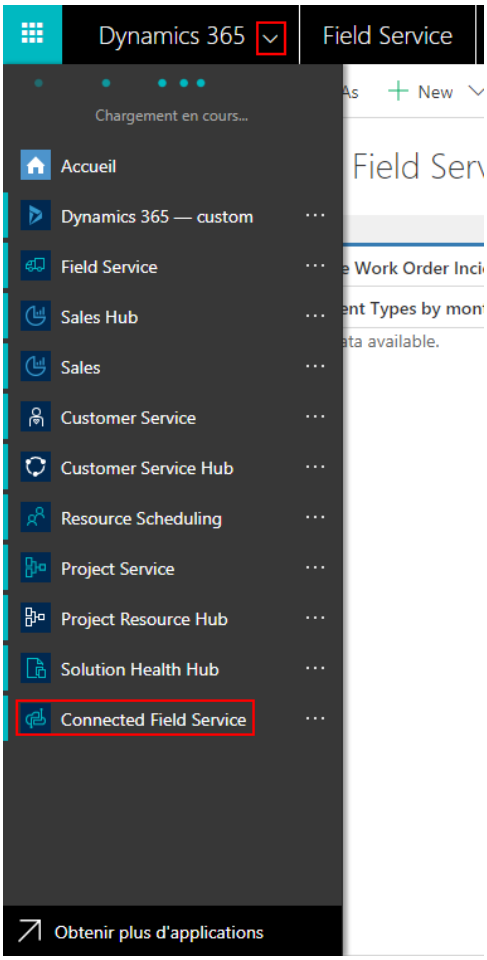


7. Go into *Admin centers section*, click on **All admin centers** and choose **Dynamics 365**.

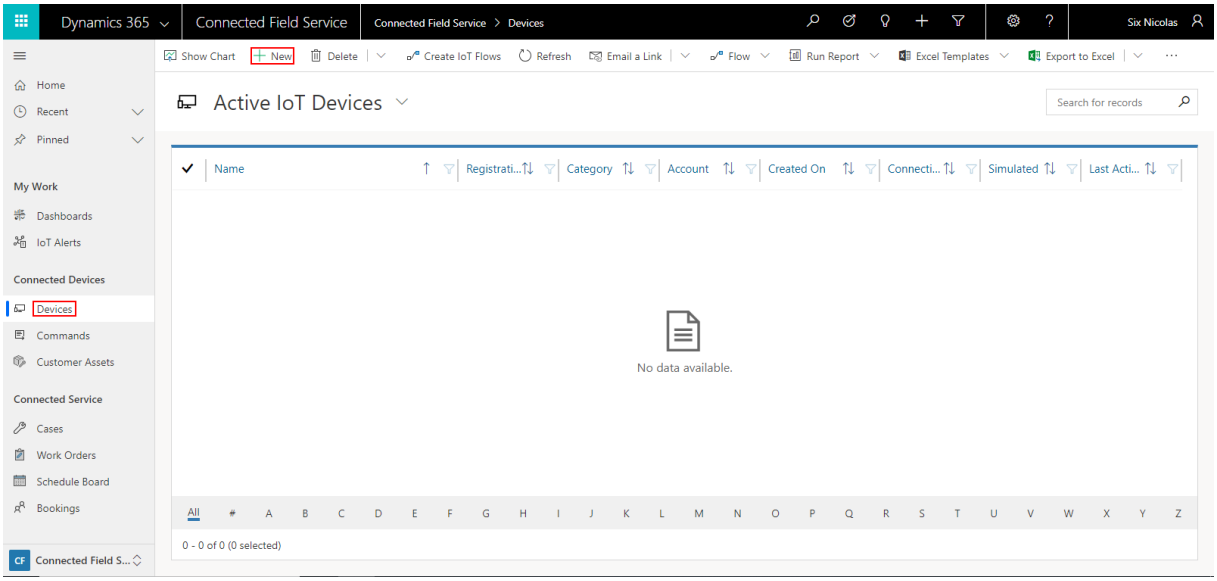
8. You will arrive on a page which invites you to choose your wanted *Dynamics 365 scenario*. Check at least **Field service** (if you want to try other services, feel free to check other boxes), and click on **Complete Setup**.



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, we are able to send IoT alerts, and it will be implemented in the next part of this guide.

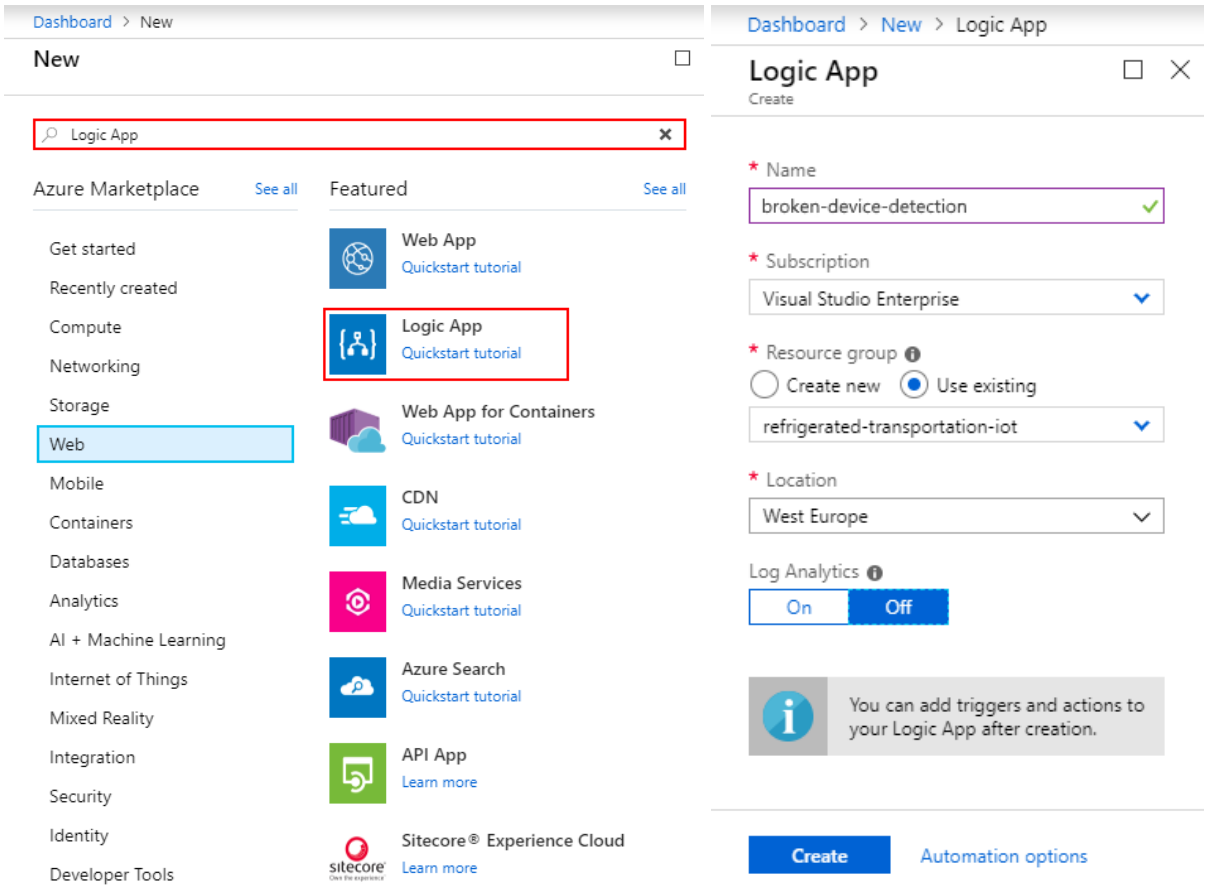
Setting up the Logic App

To create a link between our *Workbench* and *Dynamics 365*, we'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 call a step "Get contract parties" instead of the default "Execute a stored procedure".

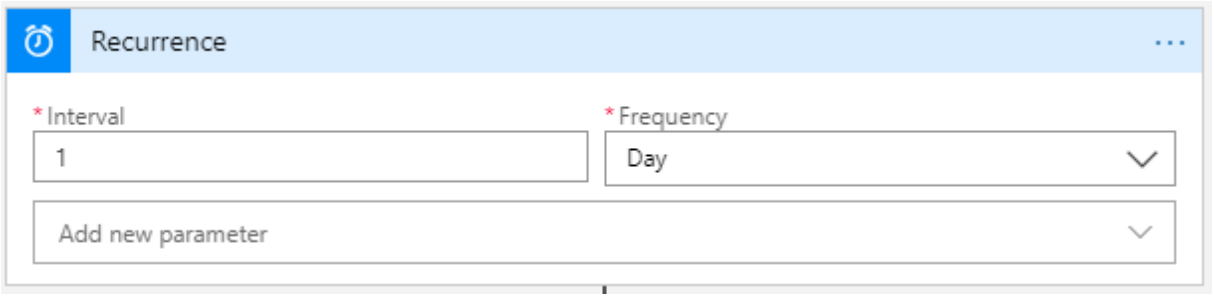
If you want to do that, keep in mind that you should do it when creating the step, and not at the end of the Logic App creation, because it's impossible to rename a step if some data from the step is used somewhere else. In this guide, we will keep default step names.

1. Log 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.



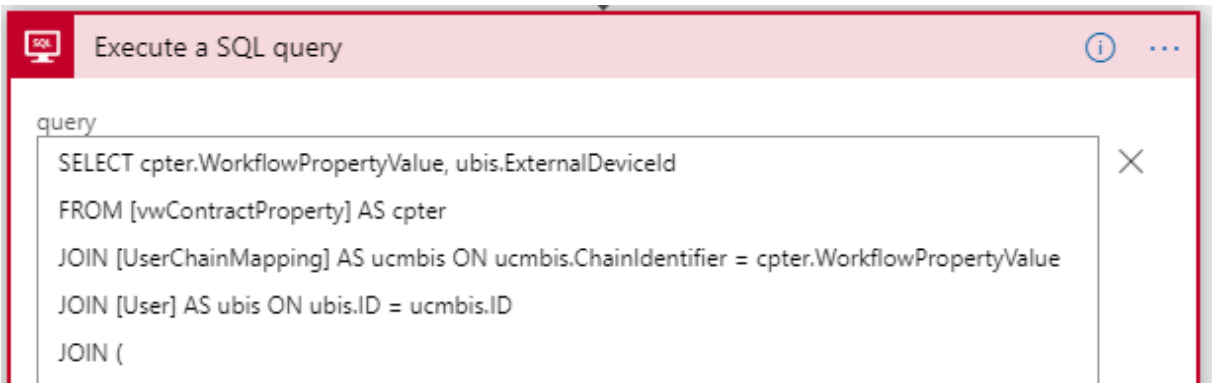
Setting	Description
Name	This is the name of your future app.
Subscription	Select the subscription to use for your *Logic App*.
Resource group	You can create a new resource group or use an existing one. To create a new one, click **Create** new and fill in the name you want to use. To use an existing 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.



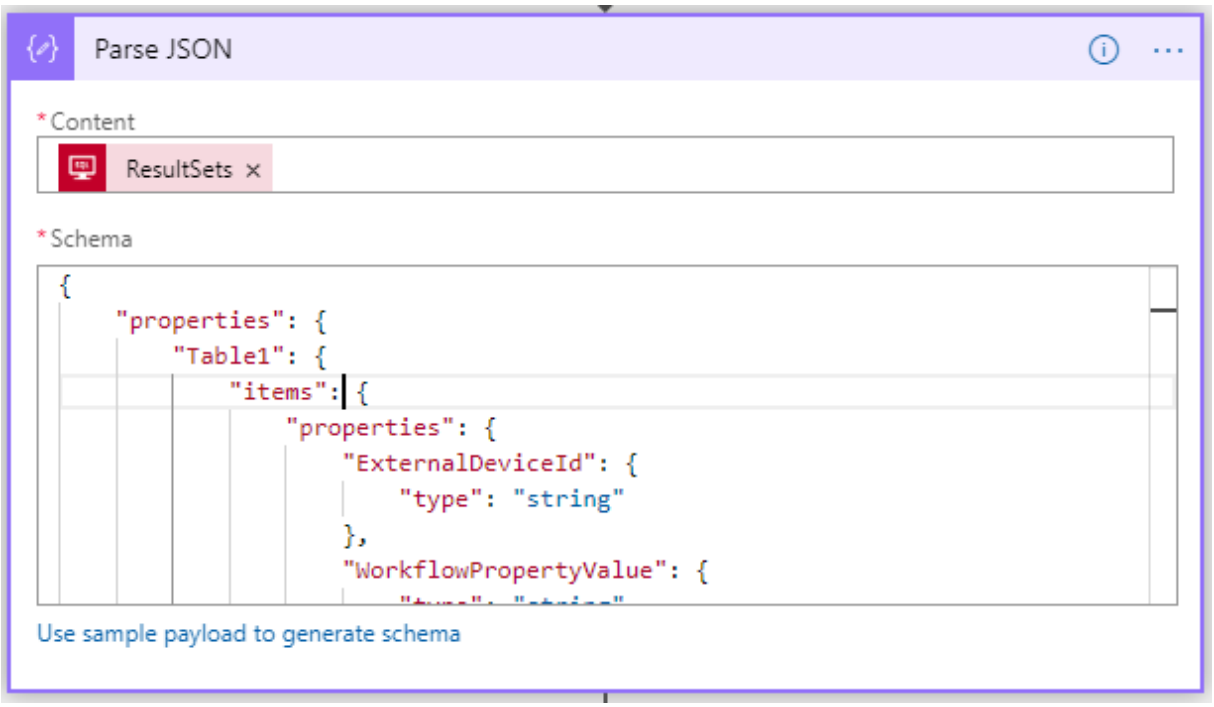
6. Click on **+New Step**. Choose **SQL Server > Execute a SQL query**, and paste the following query inside the *Query* field:

```
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
```

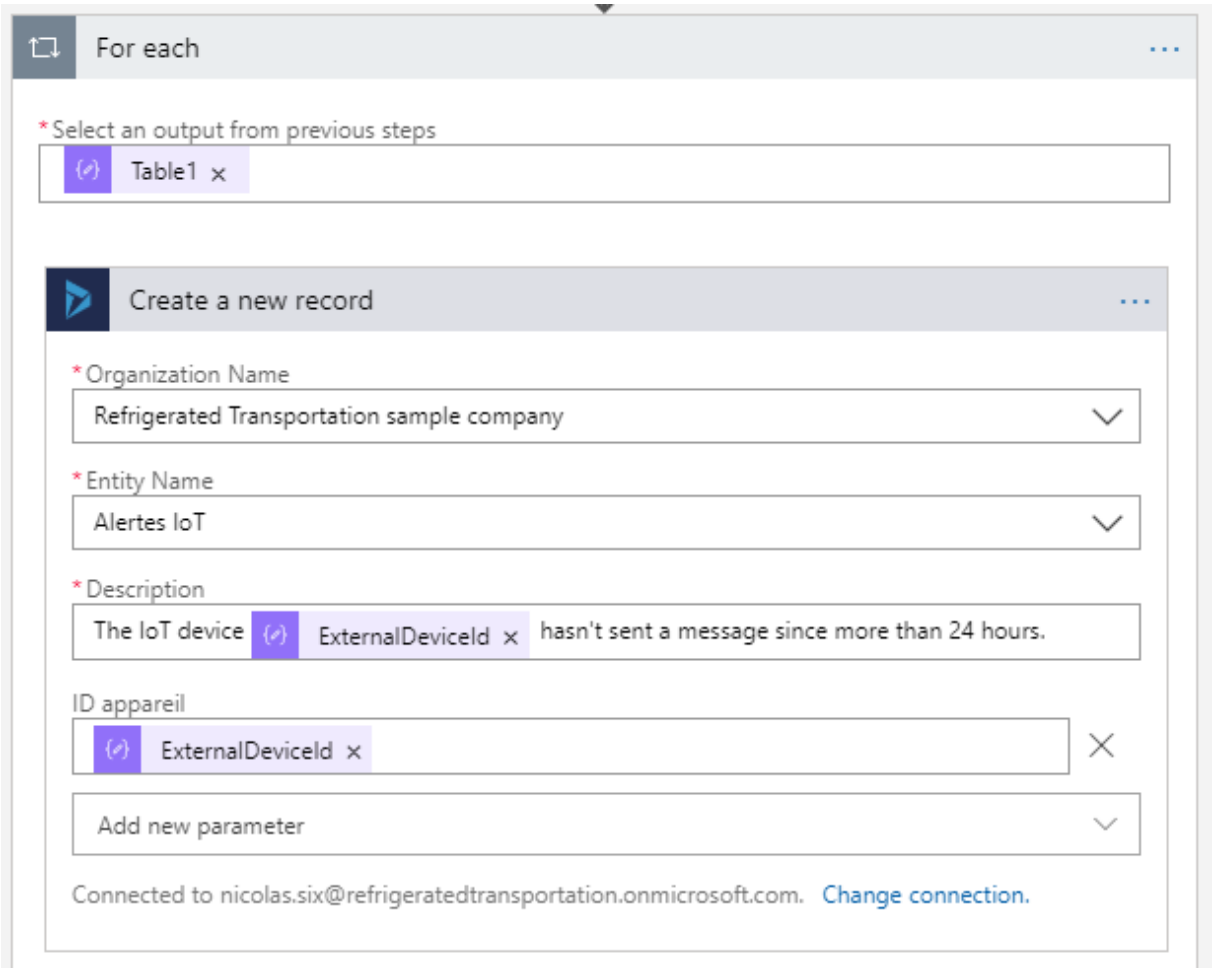


7. Click on **+New Step**. Choose **Data Operations** > **Parse JSON**. Click on the *Content* field, and choose **ResultSet**. Paste this following JSON into the Schema field:

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

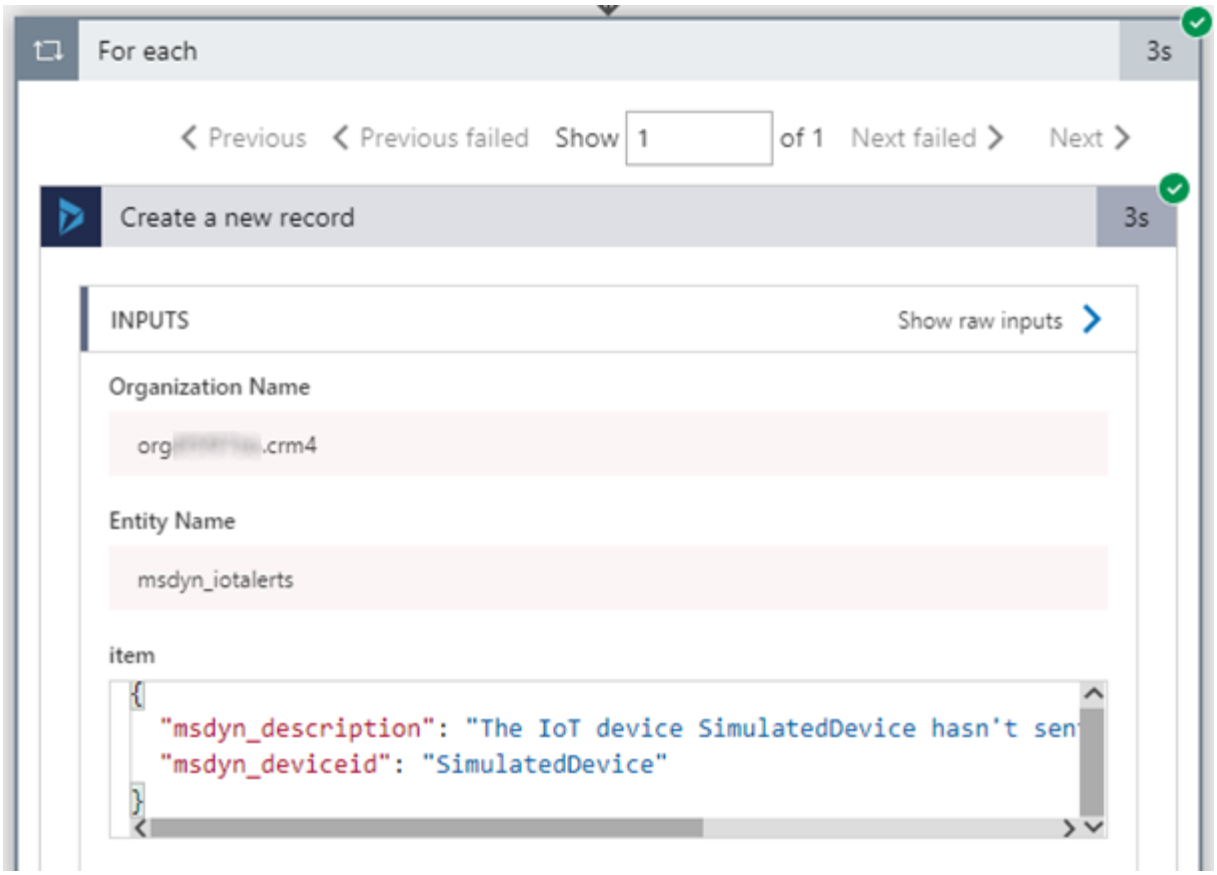


8. Click on **+New Step**. Select **Dynamics 365 > Create a new record**. You will have to log yourself on Dynamics 365, use the account where *Dynamics* is set up. Fill the form as below:

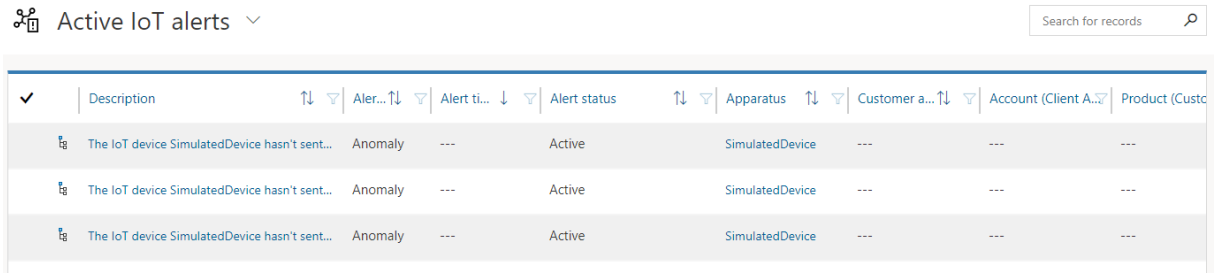


Test the application

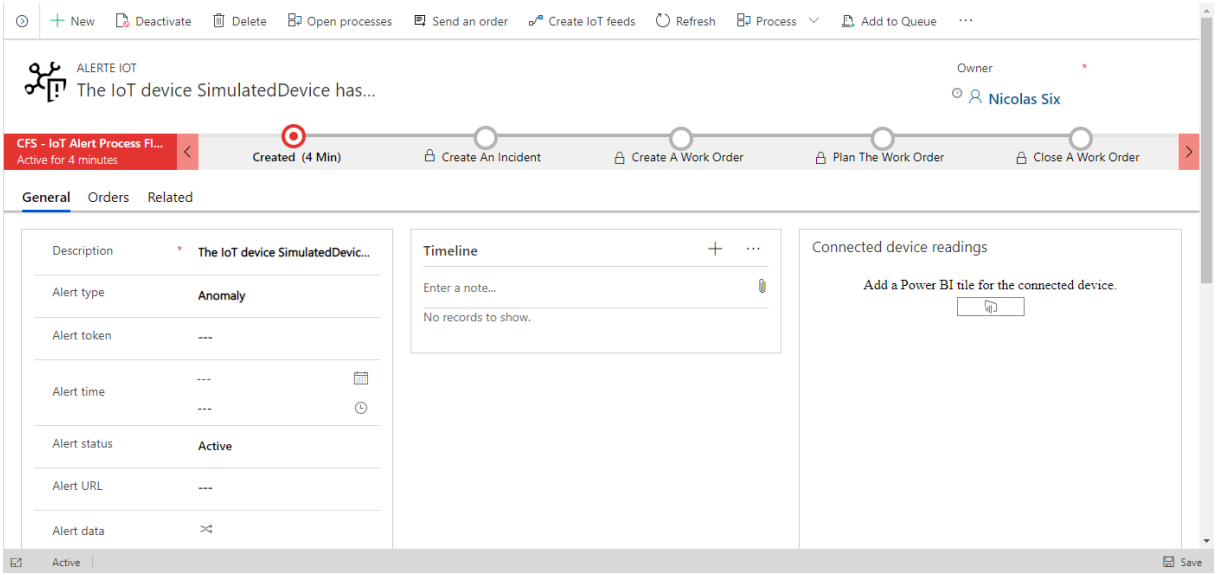
To test the *Logic App*, just click on the **Run** button to trigger the starting action. If it worked, you should see all steps marked as "*Validated*" and, by clicking on **Create a new record** step, the message sent to *Connected Field Services*.



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.

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 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 : we can use the API to integrate the blockchain inside an external solution, which can be a web application, a software, ...

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.**

```
import { AuthenticationContext, adalFetch, withAdalLogin } from 'react-adal';

// App Registration ID
const workbenchApiID = '4083c3c9-[redacted]';

// API Configuration
export const adalConfigApi = {
  cacheLocation: 'localStorage',
  clientId: workbenchApiID,
  endpoints: {
      api: workbenchApiID
  },
  tenant: 'microsoft.onmicrosoft.com',
  redirectUri: window.location.origin + '/',
  postLogoutRedirectUri: window.location.origin + '/',
};

// Authentication Context definition (contains cookies, tokens ...)
export const authContextApi = new AuthenticationContext(adalConfigApi);

// API Fetch function definition
export const adalApiFetch = (fetch, url, options) =>
  adalFetch(authContextApi, adalConfigApi.endpoints.api, fetch, url, options);

// HOC Login if needed when accessing to a restricted page
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*.

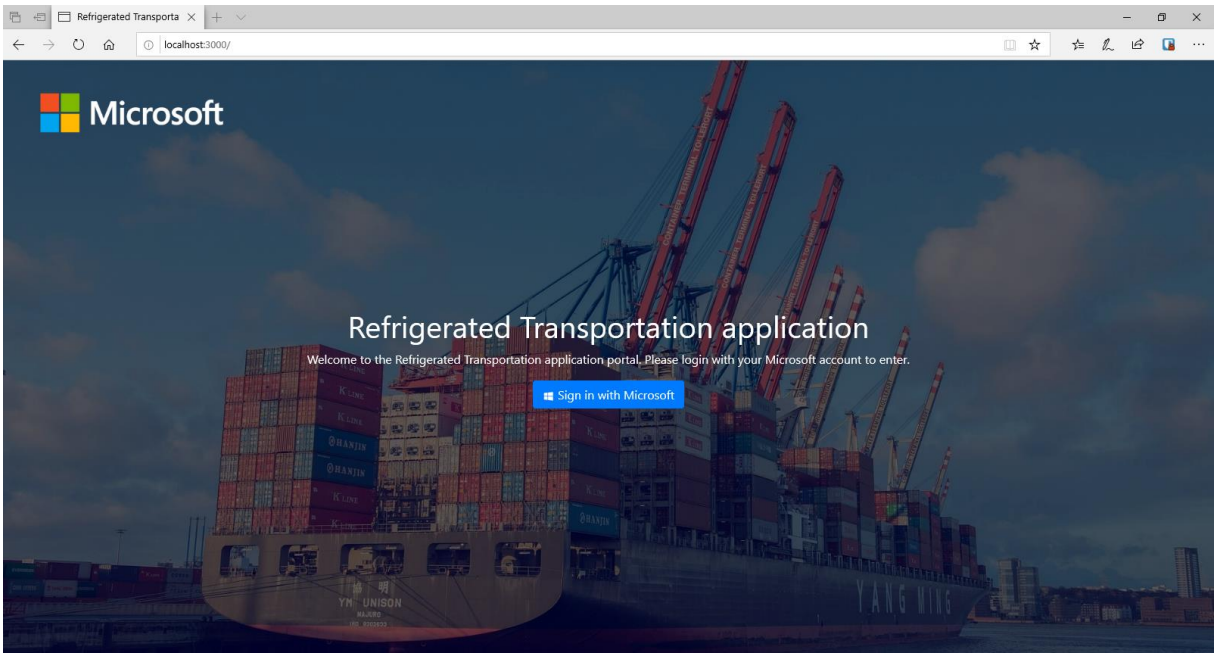
You're now ready to start the development server by typing this command into your terminal:

```
npm start
```

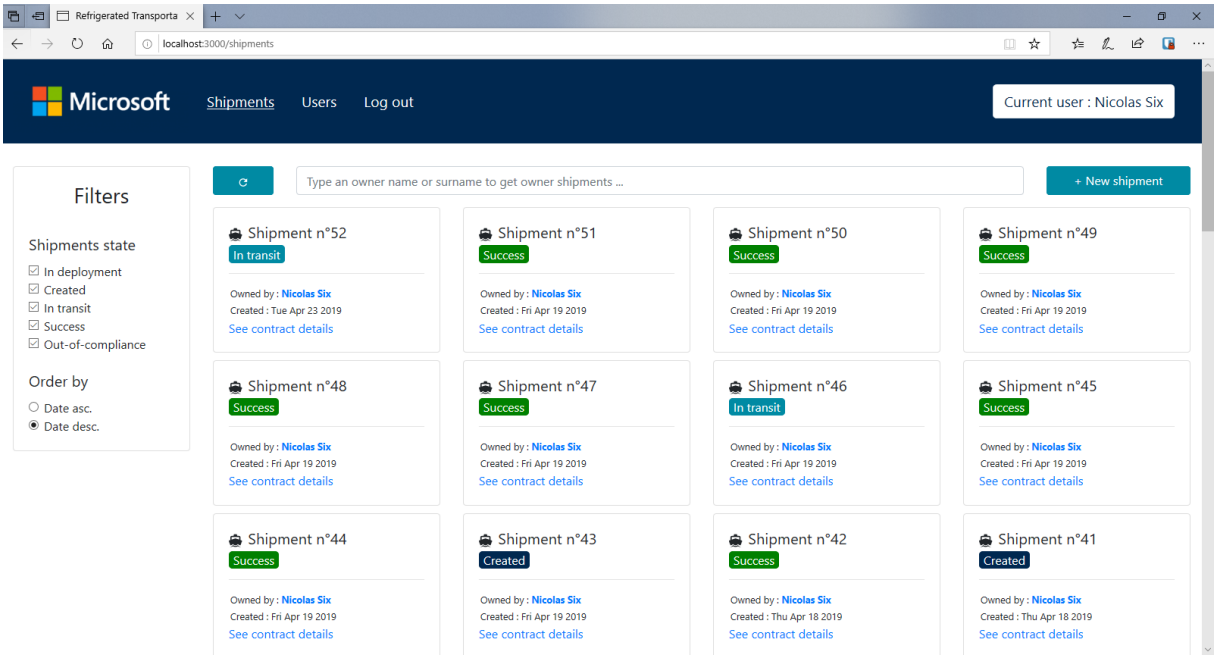
Test the application

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.



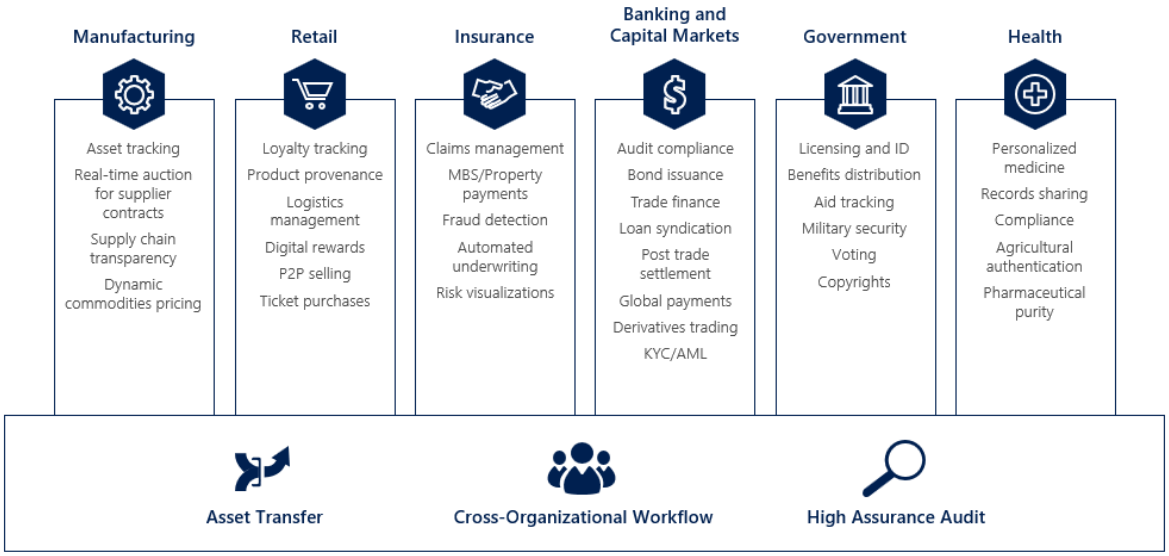
Once inside the application, you'll be able to see your application contracts displayed as shipments. The UI is really tailored on the *Refrigerated Transportation scenario,* so you will have the same view as a supply chain party could have.



Conclusion

By following this guide, you 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 ...).

Now, you should have a great overview of what can Azure Blockchain Workbench do, and how the Workbench can be applied to a wide range of applications in multiple domains. We 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 just capable of rebuilding the solution introduced in this guide, but you can think about how Azure Blockchain Workbench could be integrated into your own business workflows to improve them and save additional costs !