Edge Xpert Demo:
Ship Engine Monitoring

Demo Overview

This end-to-end Edge Xpert demonstration performs the following tasks:

- Collecting and ingesting data from Modbus and OPC-UA simulated devices that represent ship engine equipment - in this case engine temperature & power (Modbus) and pressure (OPC-UA). Many other device connectors are supported but not demonstrated here
- Exporting the data to a time series database for presentation by a Grafana dashboard
- Streaming data northbound to AWS IoT Core

Demo Requirements

- Linux (this version tested on Ubuntu 20.04) with sudo permissions
- Docker, Docker-Compose and Java already installed
- Also requires librxtx-java (sudo apt-get install librxtx-java)

Demo Preparation

1. **Install Edge Xpert**

   Download link: https://www.iotechsys.com/resources/software-downloads/
   Installation instructions link: https://docs.iotechsys.com/
   Ensure that Docker and Docker-Compose are installed and working

2. **Install the Edge Xpert license file**

   $> edgexpert license install <EdgeXpertLicenseFile>.lic
   $> edgexpert license check
   Edge Xpert Licensing: Signature valid
   Edge Xpert Licensing: License valid

3. **Install the Prosys OPC-UA Simulation Server**

   Download link: https://downloads.prosysopc.com/opc-ua-simulation-server-downloads.php
   install command similar to:
   $> sh prosys-opc-ua-simulation-server-linux-5.0.4-284.sh

Running the Demo

1. **Start the Edge Xpert microservices using the edgexpert CLI:**

   $> edgexpert up device-modbus device-opc-ua xpert-manager influxdb grafana

```
$> edgexpert up device-modbus device-opc-ua xpert-manager influxdb grafana
Creating network "edgexpert_edgex-network" with driver "bridge"
Creating network "edgexpert_default" with the default driver
Creating volume "edgexpert_db-data" with default driver
Creating volume "edgexpert_consul-data" with default driver
Creating volume "edgexpert_consul-config" with default driver
Creating volume "edgexpert_consul-scripts" with default driver
Creating volume "edgexpert_export-data" with default driver
Creating volume "edgexpert_grafana-data" with default driver
Creating volume "edgexpert_nodered-data" with default driver
Creating volume "edgexpert_portainer-data" with default driver
Creating volume "edgexpert_xpert-manager-data" with default driver
Creating grafana ... done
Creating consul  ... done
Creating redis   ... done
Creating core-metadata ... done
Creating core-data     ... done
Creating core-command  ... done
Creating export-client ... done
Creating xpert-manager ... done
Creating device-opc-ua ... done
Creating device-modbus ... done
Creating export-distro ... done
Creating influxdb      ... done
$>
```

   Verify the services are up and running:

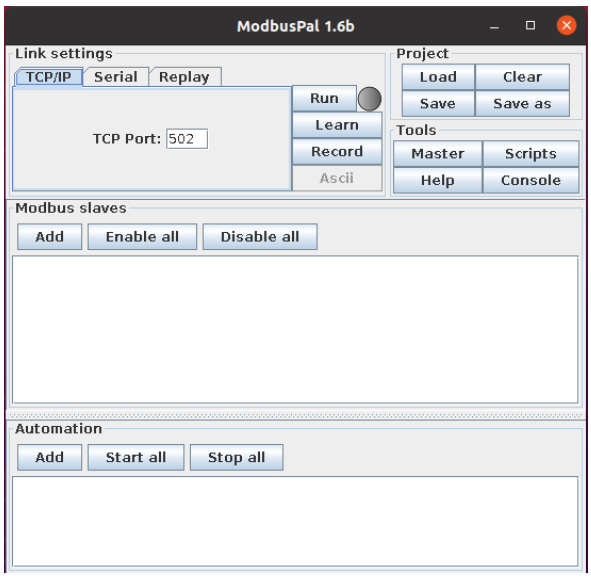
   $> edgexpert status

```
$> edgexpert status
NAMES           STATUS          CREATED          PORTS
influxdb        Up 26 seconds   28 seconds ago   0.0.0.0:8086->8086/tcp
export-distro   Up 28 seconds   31 seconds ago   0.0.0.0:48070->48070/tcp
device-modbus   Up 28 seconds   32 seconds ago   49988/tcp, 0.0.0.0:49991->49991/tcp
device-opc-ua   Up 28 seconds   33 seconds ago   0.0.0.0:49983->49983/tcp
xpert-manager   Up 28 seconds   33 seconds ago   0.0.0.0:8080->8080/tcp
export-client   Up 31 seconds   33 seconds ago   0.0.0.0:48071->48071/tcp
core-command    Up 33 seconds   34 seconds ago   0.0.0.0:48082->48082/tcp
core-data       Up 33 seconds   34 seconds ago   0.0.0.0:5563->5563/tcp, 0.0.0.0:48080->48080/tcp
core-metadata   Up 34 seconds   35 seconds ago   0.0.0.0:48081->48081/tcp
redis           Up 36 seconds   38 seconds ago   6379/tcp
consul          Up 35 seconds   38 seconds ago   8300-8302/tcp, 8400/tcp, 8301-8302/udp, 8600/tcp,
grafana         Up 35 seconds   38 seconds ago   0.0.0.0:3000->3000/tcp
$>
```

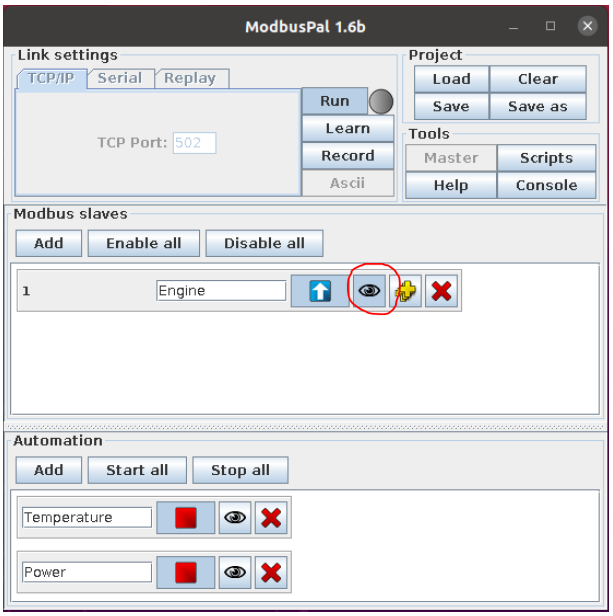
2. Start the ModbusPal simulator provided within the project:

Note it is important to use the ModbusPal.jar provided in the project. Other versions have been found to not work correctly with loaded simulations.

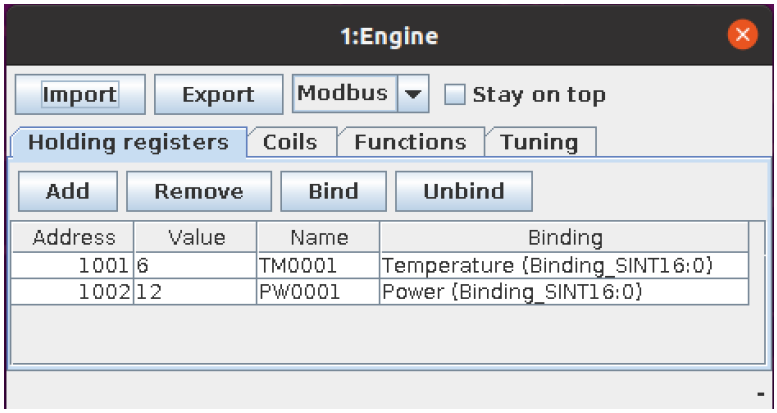
$> sudo java -jar ModbusPal.jar



Project -> Load -> ModbusSimulation.xmpp file
Automation -> Start all
Run



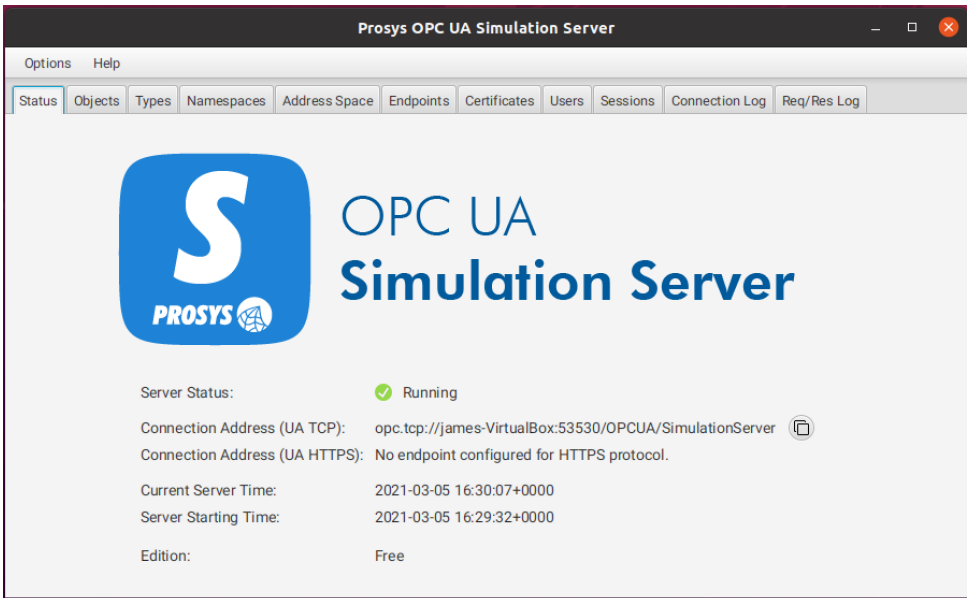
Clicking the “eye” icon shows the changing values in the simulator:



3. Start the OPC-UA Simulation Server (the actual command depends on where it was installed)

$> ./prosys-opc-ua-simulation-server/UaSimulationServer

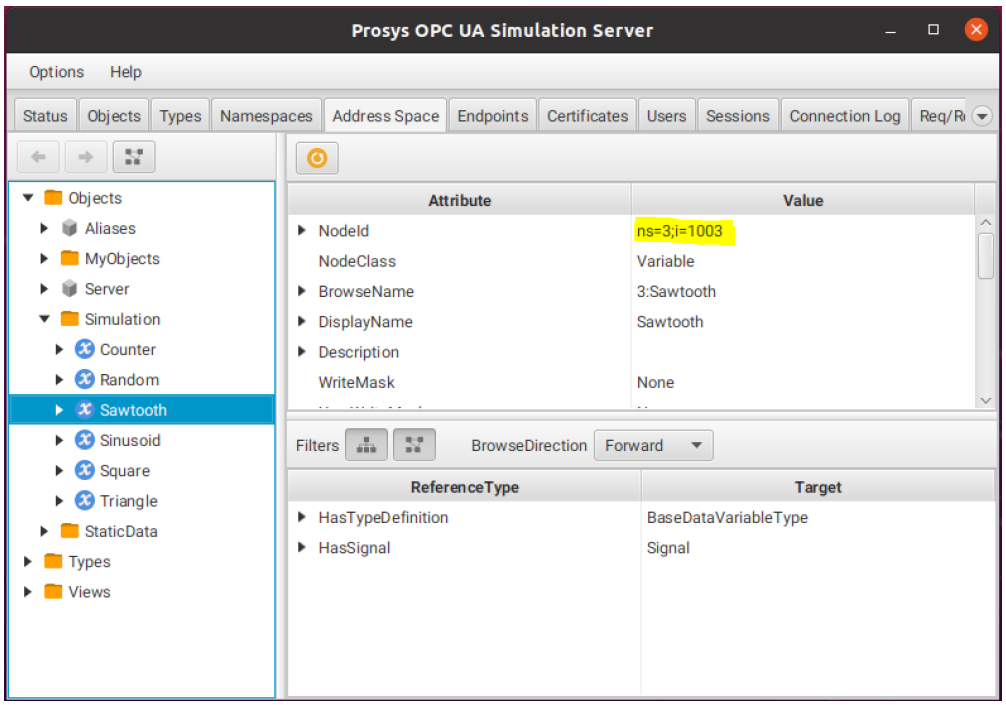
Switch to expert mode (Options -> Switch to Expert Mode)



Note that IPV6 may need to be temporarily disabled to help the Prosys simulator start.

Verify that the following attribute exists (it is used in the EngineOPCUA.yaml):

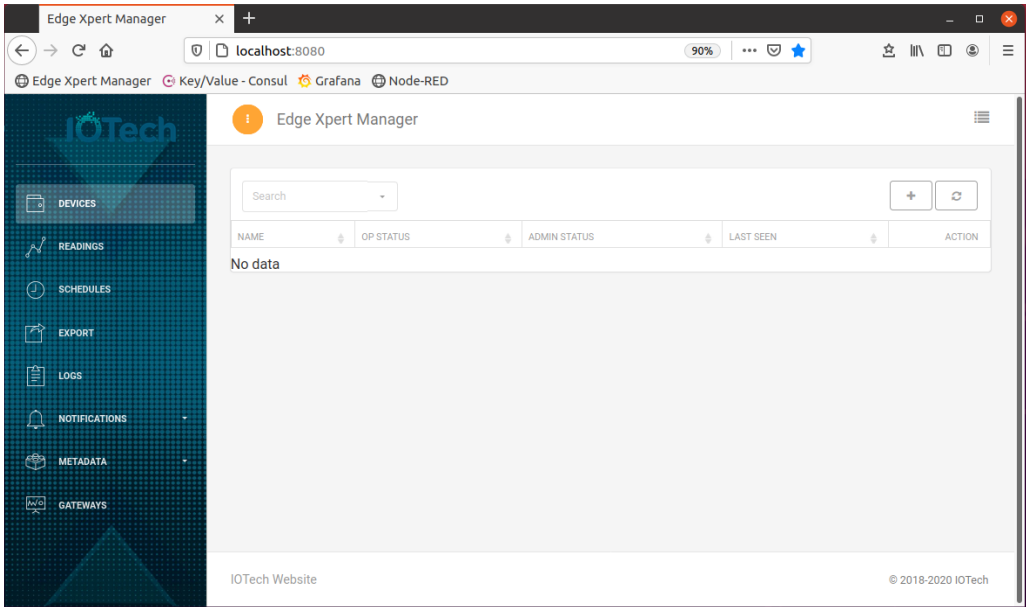
Address Space -> Objects -> Simulation -> Sawtooth (ns=3, i=1003)



4. Onboard the Modbus device to Edge Xpert:

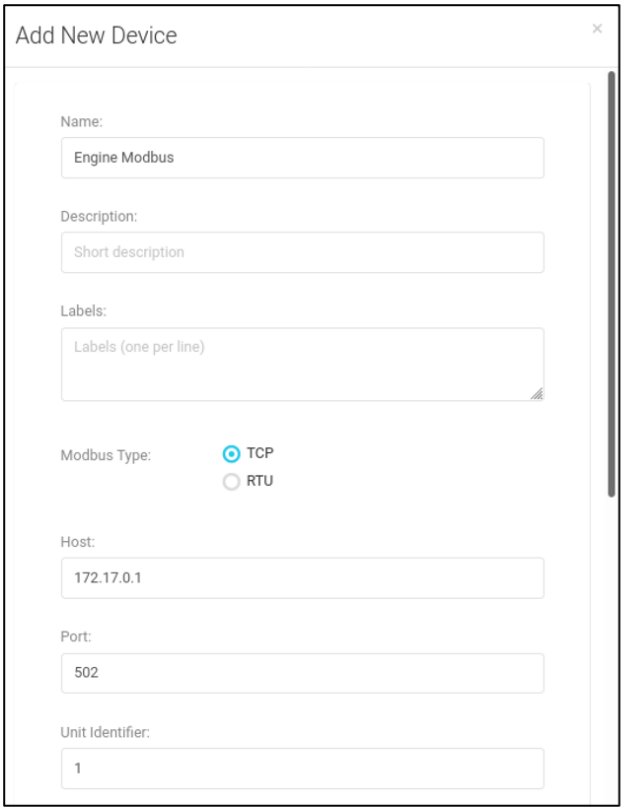
Use the Edge Xpert Manager UI. In a browser navigate to http://localhost:8080

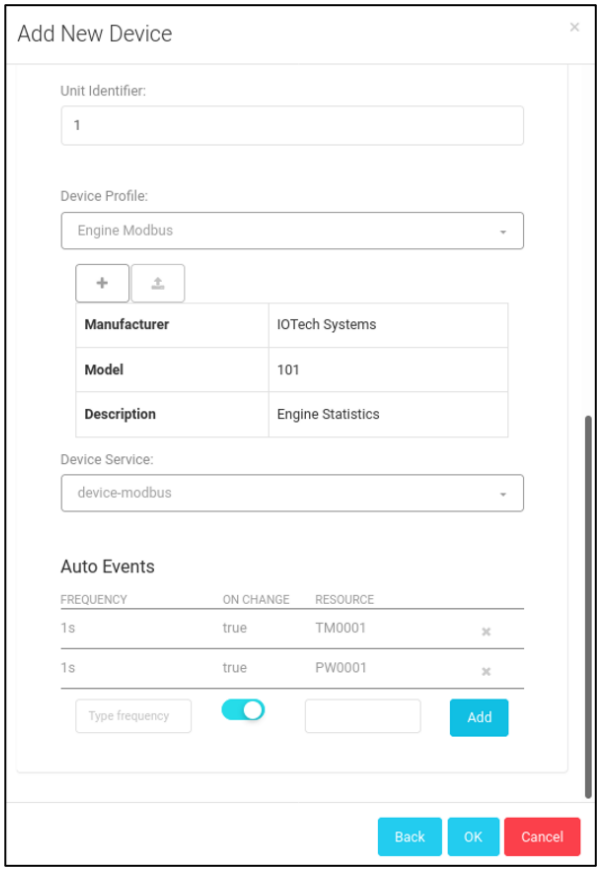
Initial login is admin/admin



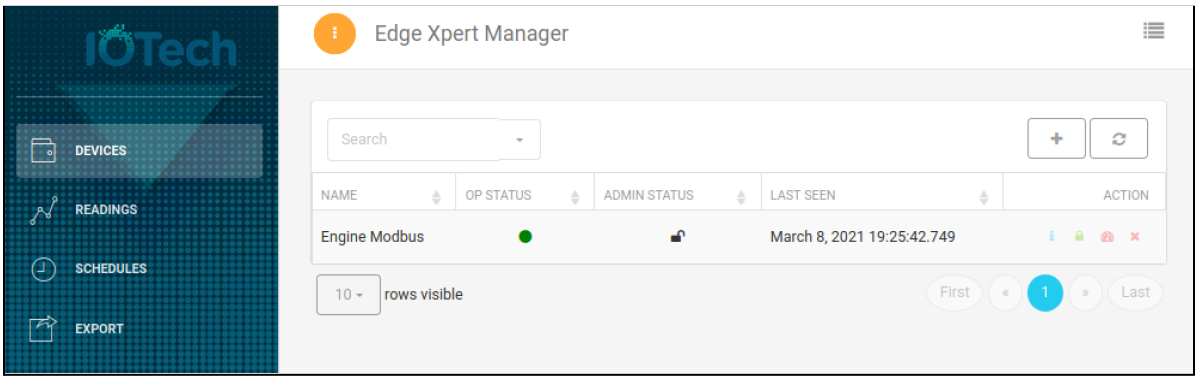
Use the Device Onboarding Wizard to add the Modbus device:

Devices -> Add Device (“+” icon) -> Modbus
Name: Engine Modbus
Description: <can be blank>
Labels: <can be blank>
Modbus Type: TCP
Host: 172.17.0.1
Port: 502
Unit Identifier: 1
Device Profile: Select EngineModbus.yaml with the + and then upload it
Device Service: device-modbus
AutoEvents:
Frequency: 1s Resource: TM0001
Frequency: 1s Resource: PW0001

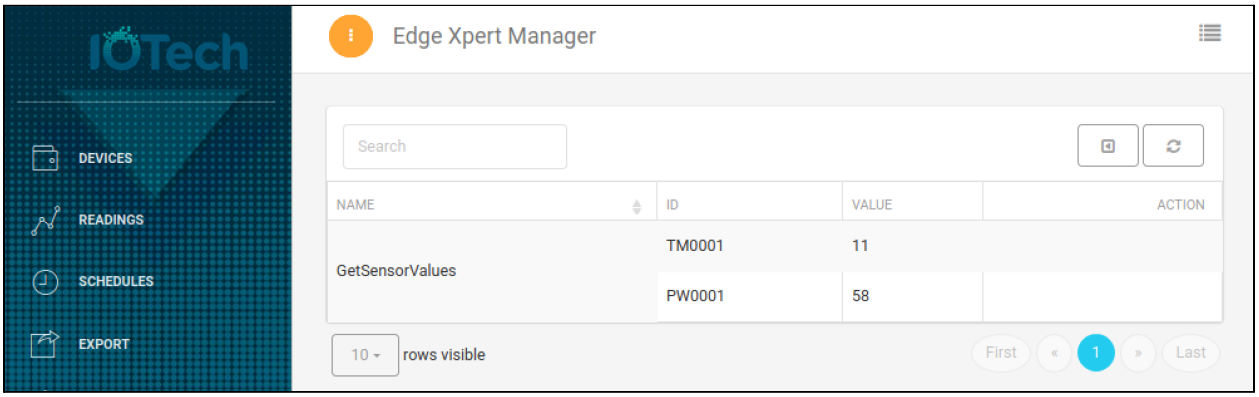




Click “OK” and the device should be onboarded:



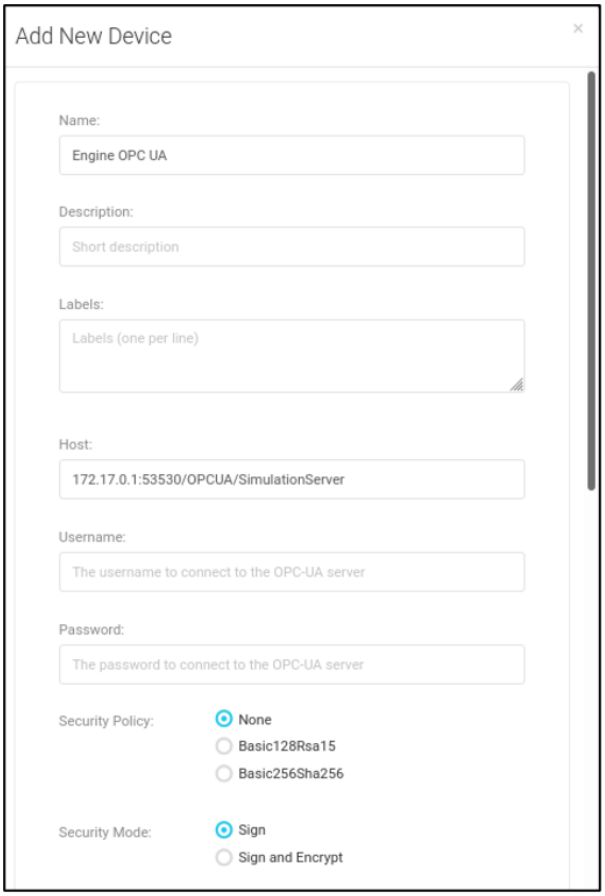
Click the “Control Device” (red dial) button to see the current values. These should track the values shown on the Modbus simulator. You need to press refresh to see the values change:

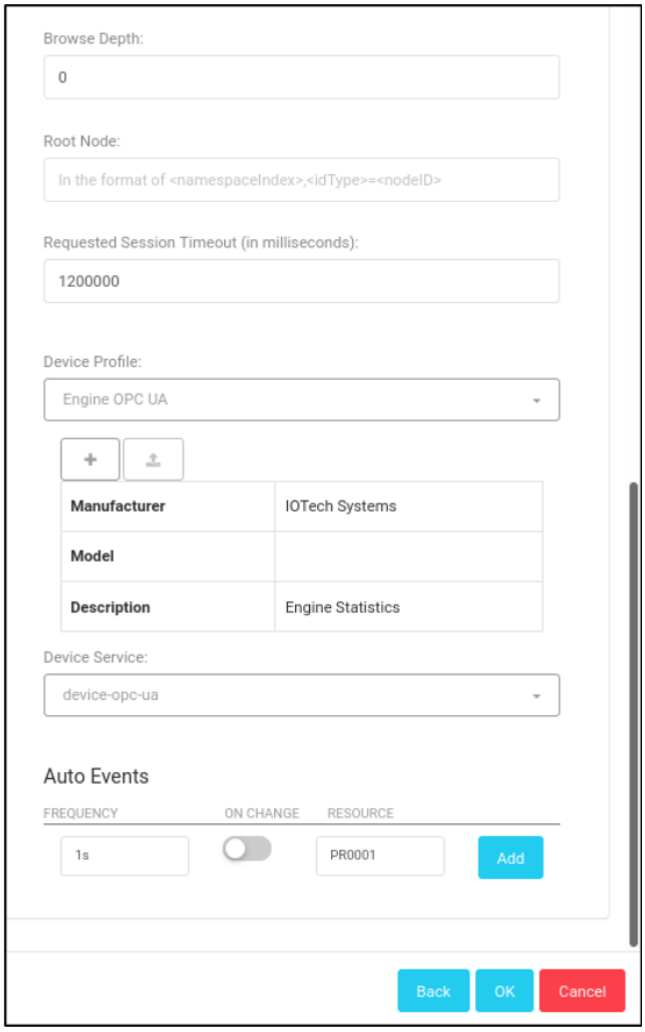


5. Onboard the OPC-UA device to Edge Xpert:

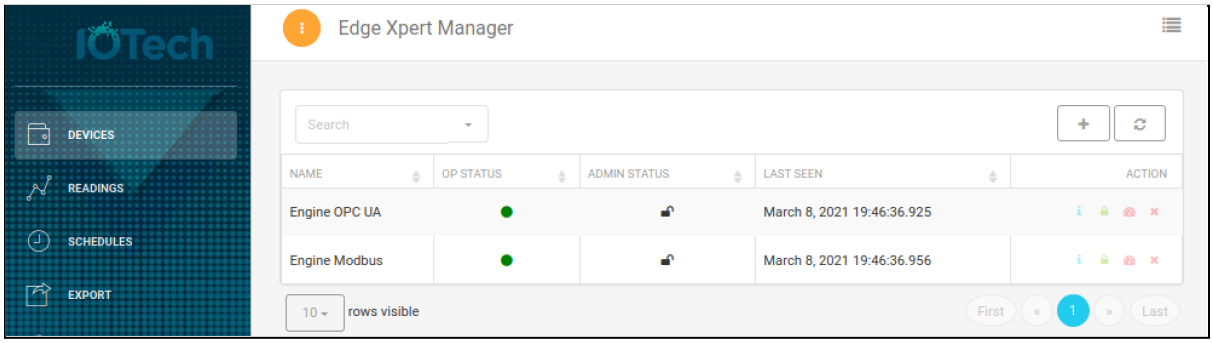
Follow the same process as above to onboard the OPC-UA device, again using the Device Onboarding Wizard:

Devices -> Add Device (“+” icon) -> OPC-UA
Name: Engine OPC UA
Description: <can be blank>
Labels: <can be blank>
Host: 172.17.0.1:53530/OPCUA/SimulationServer
Username: <blank>
Password: <blank>
Security Policy: None <default>
Security Mode: Sign <default>
Browse Depth: 0 <default>
Root Node: <blank>
Requested Session Timeout: 1200000 <default>
Device Profile: Upload and select EngineOPCUA.yaml
Device Service: device-opc-ua
AutoEvents:
Frequency: 1s Resource: PR0001





Click “OK” and now both devices should be connected:



6. Export the data to a Time-Series database (InfluxDB):

Again use the Edge Xpert Manager UI:

Export-> Add Export (“+” icon)
Destination: InfluxDB
Name: Influx
Enable: TRUE <default>
Encryption Method: None <default>
Compression Method: None <default>
Address: influxdb (this is the name of the Edge Xpert Influx microservice)
Port: 8086
Database name: edgex
Measurement name: <leave blank>
User: core
Password: password
Device filter <leave blank>
Reading filter <leave blank>

Click “OK” to create the export.

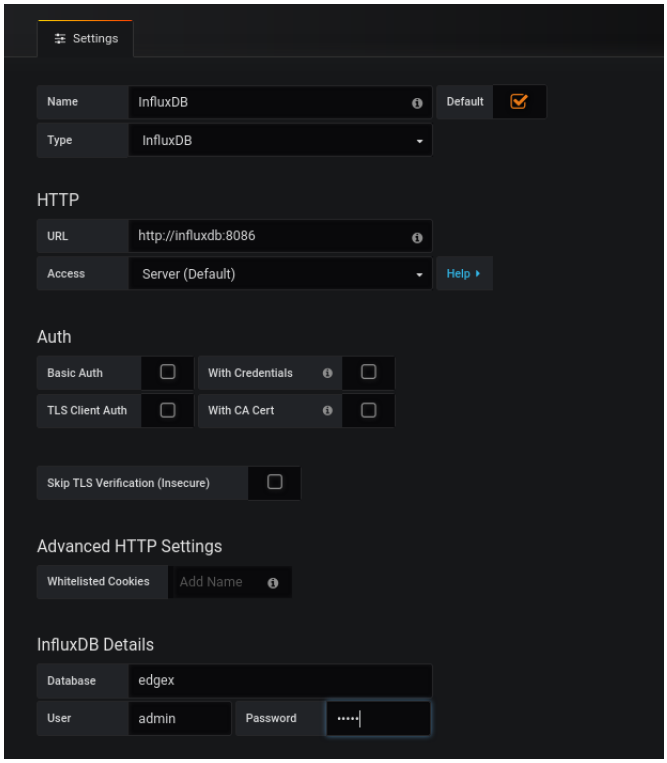
Optional check: You can see the data in influx by executing into the influxdb container as follows:

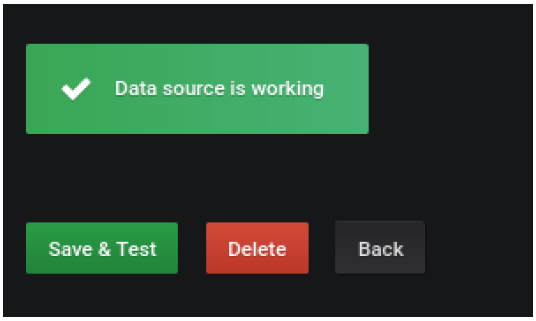
```
$>docker exec -it influxdb /usr/bin/influx
Connected to http://localhost:8086 version 1.7.10
InfluxDB shell version: 1.7.10
> auth admin admin
> use edgex
Using database edgex
> select * from readings
name: readings
time                device         event_id                             resource_name value
----                ------         --------                             ------------- -----
1615233400108684234 Engine OPC UA  1bf7b358-598b-41a8-8f65-2df2fd62824b PR0001        0
1615233400180348248 Engine Modbus  d63921e1-0da1-435e-a99e-6fe0be7cd4b4 TM0001        71
1615233400558350875 Engine OPC UA  bbb2d69e-6f21-425a-b0bc-51c9417900f9 PR0001        0
1615233401110922325 Engine OPC UA  1aab0802-213d-4cbf-9633-78c06843f239 PR0001        0.4
1615233401187920898 Engine Modbus  41044dc6-3d6f-422a-a4c6-d6a99e912688 TM0001        73
```

7. Setup the Grafana dashboard:

Use a browser and navigate to localhost:3000

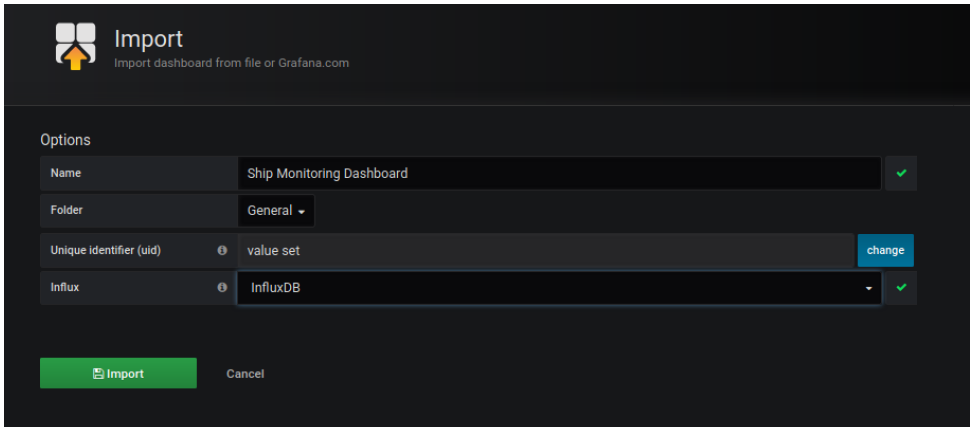
Initial login is admin/admin
Add data source:
Name: InfluxDB
Type: InfluxDB
HTTP URL: http://influxdb:8086
InfluxDB Details:
    Database: edgex
    User: admin
    Password: admin
Click “Save & Test”. It should say "Data source is working"





Once the data source is marked as working, view the Grafana dashboard as follows:

Go to Create (“+” icon) -> Import -> Upload .json file -> select ’ShipMonitoring Dashboard.json' -> Influx: InfluxDB -> Import



The dashboard should look similar to as follows:

8. Stream the data to AWS:

Use the Edge Xpert Application Services to stream the data to AWS IoT Core.

Edit the configuration-pemblock-aws.toml file to match your AWS credentials for **AWSIoTMQTTBrokerAddress**, **AWSIoTMQTTTopic** and **AWSIoTThingName**. The existing values match the IOTech test environment and will need to be changed.

Also paste the contents of your AWS certificate and private key files to the **clientcert** and **clientkey** blocks in the toml file. The entries delivered in this example are not valid and must be changed to match your own credentials.

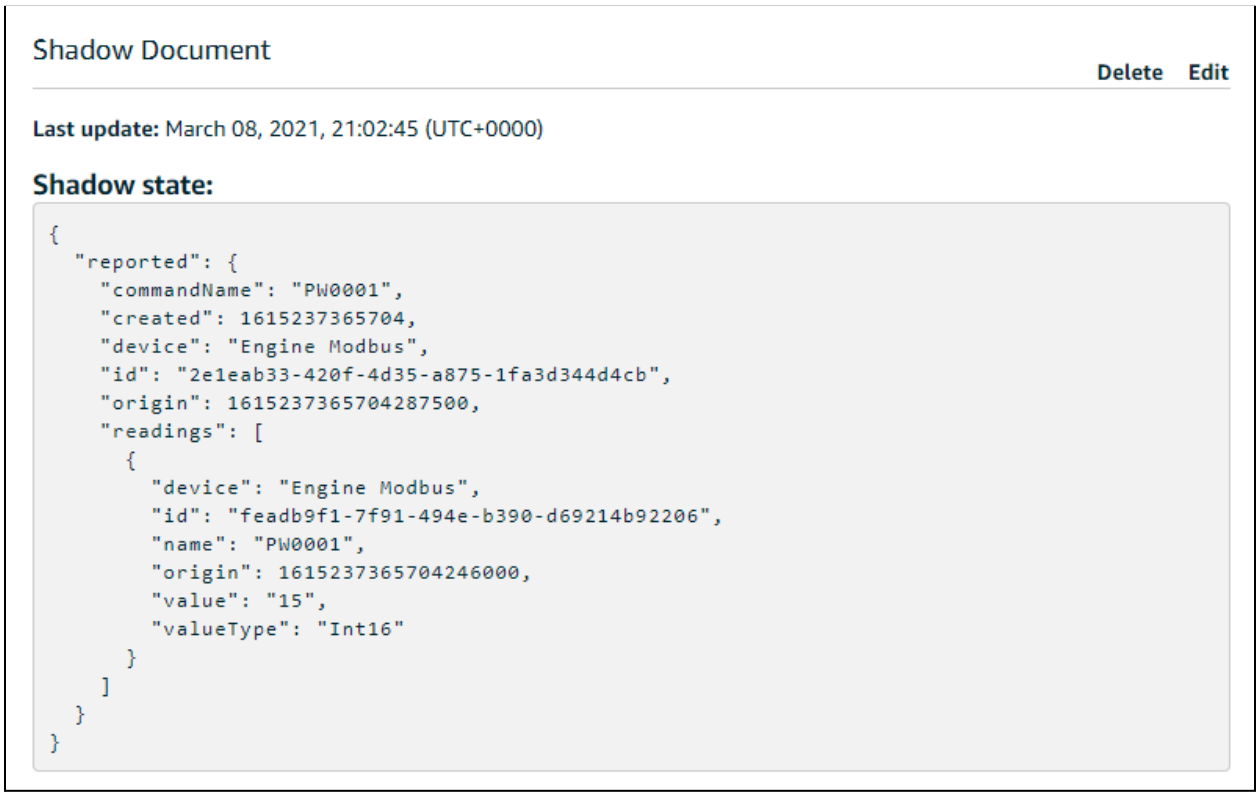
Start the Edge Xpert Application Service to connect to and stream data to AWS:

$> edgexpert up app-service --path=./configuration-pemblock-aws.toml

In a browser, navigate to your AWS account and check the data is arriving, e.g. based on the existing settings in the configuration-pemblock-aws.toml:

Internet of Things -> IoT Core -> Manage -> Things -> Engine -> Shadow

You should see the shadow data updating inline with the data from the edge simulators:



The application can be stopped as follows:

$> edgexpert down app-service --path=./configuration-pemblock-aws.toml

Optional Filtering:

Note that filtering can be applied by editing the configuration-pemblock-aws-filtering.toml in a similar way as described above. The specific functions can be enabled or disabled by editing the ExecutionOrder function list in the toml file. Then start and stop the filtering application as follows:

$> edgexpert up app-service --path=./configuration-pemblock-aws-filtering.toml

$> edgexpert down app-service --path=./configuration-pemblock-aws-filtering.toml

9. Stop Edge Xpert

The stop command stops all the running Edge Xpert microservices but leaves the containers (and therefore associated state) for later restart.

$> edgexpert down

This means that the Edge Xpert microservices and saved state such as the configured devices, exports and dashboard configurations will be maintained, so that an edge xpert up command will cause the microservices to pick up where they were stopped. No reconfiguration is required.

Warning: The edgexpert clean command stops all the running Edge Xpert microservices, deletes all the containers and deletes any volumes or networks used by Edge Xpert. So all Edge Xpert state is lost when this operation is conducted. The next time you run an 'edgexpert up', all of the configuration defined above will need to be recreated

$> edgexpert clean