AVEVAWORLD

SAN FRANCISCO 2022

AVEVAWorld 2022 Lab

PI Adapters – PI System Ingress

aveva.com

Table of contents

1. Introduction to PI Adapters

PI Adapters are our latest real-time data collection technology. They represent the continuing evolution of our software to span from edge to cloud, enabling completely new industrial IoT data patterns. In addition to being able to natively send data to an on-premises PI Server, PI Adapters can also write data up to OSIsoft Cloud Services or to Edge Data Store. Additionally, PI Adapters are supported on both Windows and Linux operating systems, with additional support for containers. Because of this, PI Adapters can be deployed on resource-constrained, ruggedized devices to monitor remote assets.

PI Adapters enable PI Systems to collect data that was previously stranded, or data that was not technically or commercially viable to measure. In summary, PI Adapters offer:

A lightweight installation footprint that can be deployed on both Windows and Linux operating systems, with support for container-based deployments (e.g., via Docker containers)
The ability to send data to PI Server, OSIsoft Cloud Services, and Edge Data Store
Different adapters support different source protocols, including:
PI Adapter for OPC UA
PI Adapter for Modbus TCP
PI Adapter for BACnet
PI Adapter for DNP3
… and more!

Regarding Windows and Linux support, PI Adapters are specifically supported on a variety of platforms and processors. Installation kits are currently available for the following platforms:

Operating System	Platform	Processor(s)
Windows 10 Enterprise Windows 10 IoT Enterprise	x64	Intel/AMD 64-bit processors
Debian 9, 10 Ubuntu 18.04, 20.04	x64	Intel/AMD 64-bit processors
Debian 9,10 Ubuntu 18.04, 20.04	ARM32	ARM 32-bit processors
Debian 9,10 Ubuntu 18.04, 20.04	ARM64	ARM 64-bit processors

In addition, custom Linux installers can also be built for PI Adapters by preparing your own tar.gz archives of the required PI Adapter binaries.

Some of you may have experience with other data collection technologies, such as PI Interfaces and PI Connectors. For comparison, before we move on to the next section, we'll walk through a quick table that highlights similarities and differences between those three product families.

	Can be set up via a GUI	Can perform data discovery on data sources	Supports Linux, ARM Linux, and Windows	Designed to run in containers	Allows local data access prior to egress
PI Interfaces	✔	✘	✘	✘	✘
PI Connectors	✔	✔	✘	✘	✘
PI Adapters	✘	✔	✔	✔	✘
Edge Data Store	✘	✔	✔	✔	✔

Data egress capabilities is a specific area where we can draw distinctions. Here are how those technologies stack up regarding sending data to a particular destination:

	Includes Buffering	Encrypts egressed data	Can write to PI Server	Can write to ADH	Egresses data via OMF	Allows local data access prior to egress
PI Interfaces	✔	✔	✔	✘	✘	✘
PI Connectors	✔	✔	✔	✘	✘	✘
PI Adapters	✔	✔	✔	✔	✔	✘
Edge Data Store	✔	✔	✔	✔	✔	✔

In brief summary, it's only PI Adapters and Edge Data Store that can write to OCS in addition to writing to PI Servers (and moreover, via the OSIsoft Message Format, or OMF for short). And, as its name implies, of all of the data ingress options discussed, only the Edge Data Store provides local, edge-accessible data storage for accessing data streams before they are egressed to their final destination.

1.1. PI Adapter Buffering

When it comes to the edge, buffering data can be essential. PI Adapters include built-in data buffering to minimize data loss due to network disruptions. Buffering is applied both when egressing data source data as well as health telemetry, using a first in, first out approach. Depending on the hardware or solution requirements, buffering can be configured as in-memory-only buffering or on-disk buffering.

1.2. PI Adapter Security

Outgoing data to the PI System is encrypted using the TLS protocol by default when HTTPS is used (which should always be the case, in production environments).   Data egress over HTTP is used when egressing data locally to an Edge Data Store, but at minimized risk, since the Edge Data Store only accepts data from local, same-host data sources.

1.3. PI Adapter Architecture

PI Adapters are designed to fully operate without requiring any third-party vendor software. Below is an example architecture for a PI Adapter, in this case, the PI Adapter for Modbus TCP. In the below architecture, like all PI Adapters, the OSIsoft Message Format (OMF) is used to send data to data and health endpoints:

PI Adapter for Modbus TCP architecture
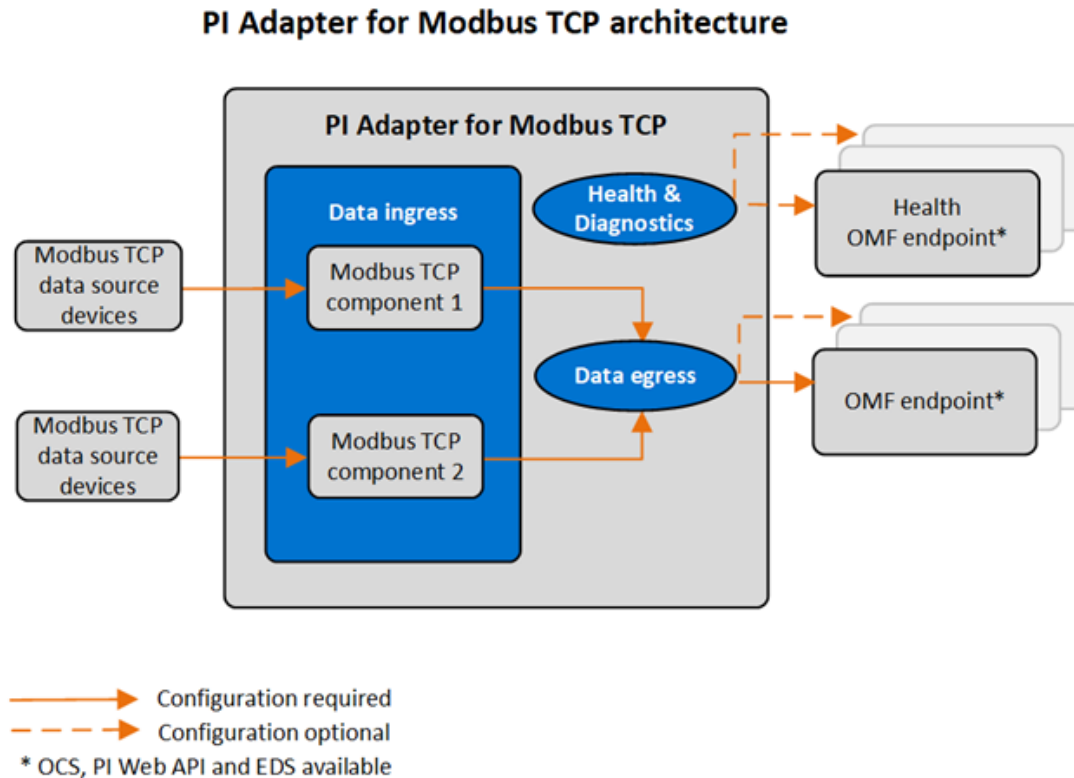
1.4. PI Adapter Configuration

Most commonly, PI Adapter configuration and administration is performed using EdgeCmd. EdgeCmd is a command line application for configuring PI Adapters and Edge Data Store, making it easy to configure a single adapter reading from a single data source.

Under the hood, the commands we'll run in this class with EdgeCmd are translated to HTTP queries against a REST API exposed by the adapter. It is possible to write your own custom HTTP queries to configure PI Adapters using the same queries that EdgeCmd uses allowing you to both edit any single component or facet of the system individually as well as configuring the entire PI Adapter with a single REST API HTTP request. We're not covering this in this class, but for more information on the REST API for adapters, see the *documentation* for the PI Adapter you are using.

Both EdgeCmd and REST configuration calls can only be made locally on the adapter machine. You **must** be connecting fro the local machine running the adapter – as long as you are, there is no additional authentication required when changing configuration.

All PI Adapters can be configured following a common workflow. First, before beginning data ingress, you must define a new PI Adapter component within the system components configuration, with one new component for each unique device from which the Adapter will collect data. The connection information for each of those devices, and the information for specifying which particular data items to collect, is next needed for configuring each Adapter component.

For data egress, you provide information that identifies the particular destination for the data—such as the address for the OMF endpoint of a PI Server, EDS, or OCS tenant—along with authorization details for the outgoing connection.

When applying authorization details, any passwords or secrets that are entered into a PI Adapter are stored internally in encrypted form, with cryptographic key material stored separately in a secure on-disk location.

Additional parameters can be specified to enable egressing health and diagnostics data, tailoring buffering configuration to protect against data loss, customizing logging information for troubleshooting purposes, and the sending of health data to a data egress.

2. Configuring an Adapter from Scratch: PI Adapter for OPC UA

We will be configuring the PI OPC UA Adapter step by step, using EdgeCmd.

Just as a preview, below is a quick whiteboard of what our completed architecture will look like:
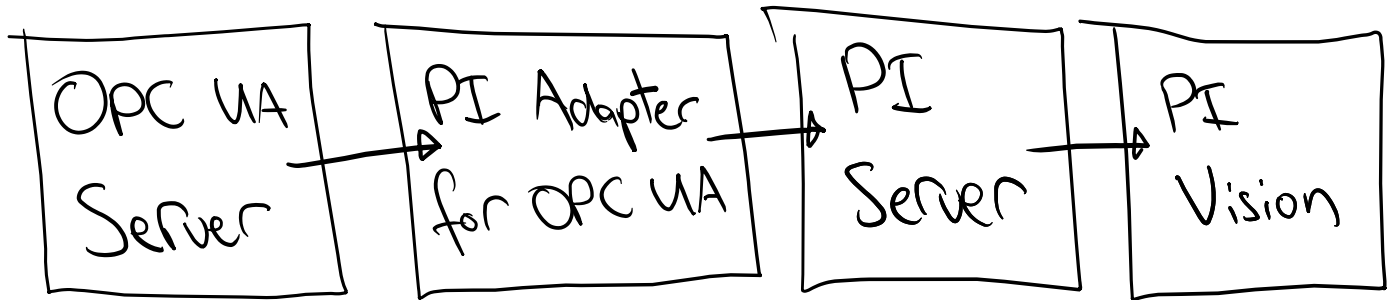
All these components will be configured on the same machine in this class. In the real world you'd split up the components into separate machines – but the configuration will be very similar to the configuration in this class.

During the configuration, if you need more information about any steps or terminology you'll find the documentation for the PI Adapter for OPC UA *here* (also bookmarked in the VM's web browser).

Our virtual environment has an OPC UA Server simulator, which will act as our data source. All other components are preinstalled, but the PI Adapter for OPC UA is not configured. We'll cover from-scratch configuration, performing the following actions:

- Step 1: Create a Component

We need to create a component for each source or destination of data to house its configuration. We'll create a component to hold the configuration for reading from our OPC UA Server simulator.

- Step 2: Configure a Data Source

We need to tell our new Adapter component where it should read data from.

- Step 3: Discover data items

We'll ask our adapter to scan for data items on our data source.

- Step 4: Configure Data Filtering

We'll define deadband rules for data in effort to reduce network bandwidth taken by our Adapter by avoiding sending unnecessary data.

- Step 5: Configure Data Selection

We'll combine the work we did with discovery and filtering, and tell our Adapter which data items to collect, and how.

- Step 6: Configure Egress to the PI System

Next, we'll configure the endpoint in which we'd like to send the data. In our case this will be a PI Web API server, which will forward the data on to a PI Data Archive.

- Step 7: Confirm Data Flow

Lastly, we'll make sure we can see data coming into our PI Data Archive from our Adapter and look at some health tags.

2.1 Introduction to EdgeCmd

Let's go ahead and use our first command to start getting more comfortable with the EdgeCmd configuration utility.

1. On your PISRV01 machine, open a new command prompt. In the command prompt, type the following command, then press ENTER:

   `EdgeCmd help`
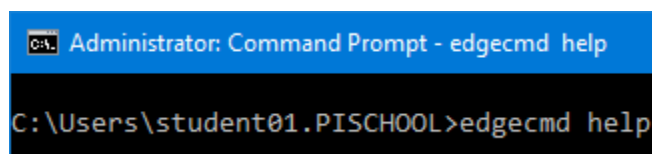
You should see the help overview for the utility appear, including details about the command pattern syntax, available operations, arguments, targets, component IDs, and several example commands. More detailed help can be found in the EdgeCmd documentation (you'll find a bookmark for it in your lab machine's browser).

If we wanted to, we could do the whole configuration in this cmd prompt window. That would involve a lot of typing though, and it's difficult to reuse our work afterwards.

Instead of typing everything into a command prompt, we'll make use of the Windows PowerShell ISE to keep track of our commands. This way we'll get more of an idea of the steps to configure an adapter and end up with a script to configure an adapter from scratch that we can reuse later if we need.

2. Open the "Class Files" folder on the desktop of your VM, then double click on OpcUaAdapterConfiguration.ps1.

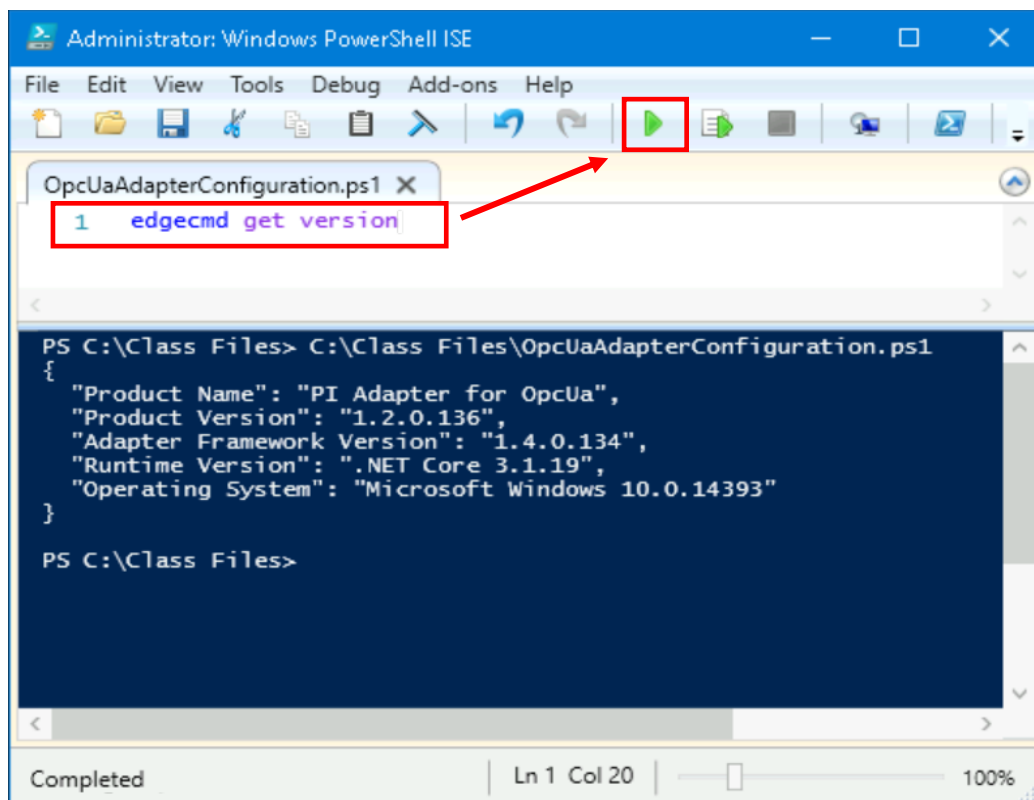This will open the Windows PowerShell ISE into a blank file.

Next, let's run a "get" command to get the current PI Adapter version using this new user interface.

3. On the first line of the script, type the following command:

   `edgecmd get version`

4. Press the "Run Script" button.

This button will run every line of script we've typed into the window so far.

You should see a response like the above showing the current PI Adapter version information. We'll only use this "Run Script" button this one time. For all upcoming steps we'll be using the next button along to only run our script a single line at a time.

> **Note:** During this class we use PowerShell to run these commands. The same commands (with very minor syntax changes for multiline commands) will work in Windows Command Prompt, or a Unix Shell, once EdgeCmd is installed on the machine.
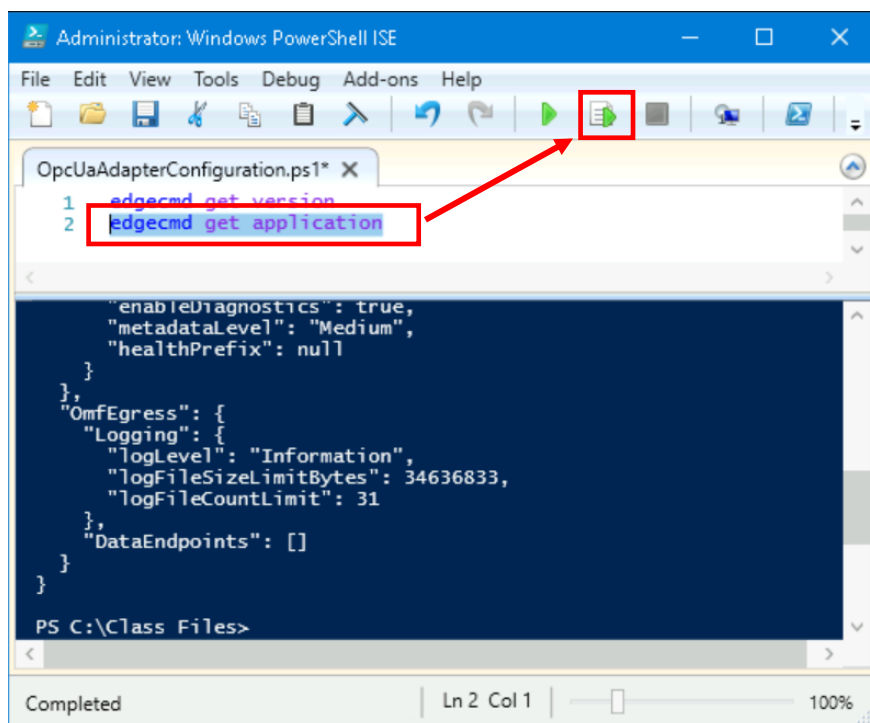
Now, let's use another "get" command to get the current PI Adapter application configuration.

5. Type the following command underneath the "edgecmd get version" command.

   `edgecmd get application`

6. Use your cursor to select the line containing your newly typed text

7. Click the "Run Selection" button.

This button will only run the lines of script we currently have selected.

You will see the default configuration for a newly installed PI Adapter for OPC UA. In the coming steps, we are going to update this configuration so it will begin collecting and streaming real-time data.

> **Note:** From this point on, when this document states that you should "Run a command", you should add the command to your script, then run it by selecting the line and clicking the "Run Selection" button.

2.2 Create a Component

Components are used to house configuration, either to read from a data source, or write to a data egress. Let's see what components we already have configured on our default installation.

> 8. Add the following command to your script, and run the single line:
>
> ```
> edgecmd get components
> ```

You should see the output below:

```
[
  {
    "componentId": "OmfEgress",
    "componentType": "OmfEgress"
  }
]
```

The only component we have right now is the default one to house our egress configuration. It doesn't do anything right now, but we'll use it later.

Let's create a new component to house the configuration for our data source – our OPC UA Server Simulator.

> 9. Add this line to your script and run it:
>
> ```
> edgecmd add components -id simulator -type OpcUa
> ```

This adds a new component called "simulator" of the type "OpcUa". After running the command, you should see a "Operation has been completed successfully" message.

> 10. Highlight your "edgecmd get components" line again and run it. You should now see two components listed.

2.3 Configure a Data Source

We've now got a component created to house our configuration. Let's tell it where to get data from.

> 11. Add these lines to your script file, select them **all,** then run them with the "Run Selection" button:
>
> ```
> edgecmd set datasource `
>   -cid simulator `
>   -endpointurl opc.tcp://127.0.0.1:49320 `
>   -usesecureconnection true
> ```

Note: This is a single command split out into multiple lines for better readability. We'll use these multi-line commands frequently throughout this class.

In this command we are setting parameters for the datasource property in our component called simulator. We set the endpointurl to opc.tcp://127.0.0.1:49320 (which is the address of our OPC UA simulator server) and tell it to use a secure connection.

> 12. Run the following command to look at the current configuration of your Data Source:
>
> ```
> edgecmd get datasource -cid simulator
> ```

You should see the details we entered in the command. Now our Adapter knows about our data source, let's see if it was successful in connecting. To do this, we check the log files:

> 13. Open Windows file explorer

> 14. Click the "Adapters" pinned folder on the left pane. This will bring you to C:\ProgramData\OSIsoft\Adapters

15. Drill down to C:\ProgramData\OSIsoft\Adapters\OpcUa\Logs, and double click on the file beginning with "simulator-".

This is the log file or our newly created component. The file will open in Atom text editor and will automatically update when new log messages are written to it. Look at the last couple of message in the log:

```
2022-04-11 07:44:04.965 +00:00 [Warning] Rejected OPC UA server certificate:
Subject name: CN=KEPServerEX/UA Server, O=Unknown, C=US, DC=PISRV01.PISCHOOL.INT, Thumbprint:
BE3C7C5FF2DC4AA9EEB9D4885EBEEE6F3ED073CD.

2022-04-11 07:44:05.573 +00:00 [Error] "Server certificate isn't trusted. Please verify the
certificate and then move it to the trusted folder." "Exception type: StatusException.
Message: Certificate is not trusted. SubjectName=CN=KEPServerEX/UA Server, O=Unknown, C=US,
DC=PISRV01.PISCHOOL.INT Thumbprint=BE3C7C5FF2DC4AA9EEB9D4885EBEEE6F3ED073CD Inner Exception:
StatusException Message: Certificate is not trusted. SubjectName=CN=KEPServerEX/UA Server,
O=Unknown, C=US, DC=PISRV01.PISCHOOL.INT Thumbprint=BE3C7C5FF2DC4AA9EEB9D4885EBEEE6F3ED073CD"

2022-04-11 07:44:05.579 +00:00 [Information] Connection failed. The adapter will retry every
10 seconds...
```

These messages state that the Adapter con't connect because the certificate of the server isn't yet trusted by the adapter. The next few steps will set up security on both sides so the Adapter trusts the server, and the server trusts the Adapter. We'll configure the Adapter side first. Keep the log open – we'll need it later.

16. Open Windows file explorer and navigate to

C:\ProgramData\OSIsoft\Adapters\OpcUa\Certificates

Note that this folder has several different subfolders. We're most interested in the "rejected" folder, which stores certificates that the adapter attempted to connect to that are unknown to it, and the "trusted" folder, which stores certificates of servers that are known.

When we added the discovery, a new certificate would have appeared in the "rejected" folder. We're going to move that certificate to the "trusted" folder.

17. Navigate to:

C:\ProgramData\OSIsoft\Adapters\OpcUa\Certificates**rejected**\certs

18. Right click and "cut" the certificate file you find inside this folder.

19. Navigate to:

    C:\ProgramData\OSIsoft\Adapters\OpcUa\Certificates**trusted**\certs

20. Right click inside the folder and "paste" the certificate inside.

21. Switch back to your Atom window again and look at the logs.

You may see some certificate warnings (those can be ignored in this lab), but you'll also see a message like the following:

```
2022-04-11 07:52:41.329 +00:00 [Error] "Connection failed. Status Code:
BadCertificateInvalid." "Exception type: StatusException. Message: [BadCertificateInvalid]"
```

this indicates that the adapter attempted to connect to the server but couldn't because the server doesn't trust the adapter yet. We need to configure security on the server side.

Note: the upcoming steps configure security for the simulator we are using in this lab, KEPServerEx. Every OPC UA Server uses a different workflow to trust a client certificate. You should consult your OPC UA Server vendor's documentation if proceeding through these steps in your own system.

22.	Expand the system tray in the bottom right of your window.

23.	Right click on the KEPServerEx icon

24.	Open "OPC UA Configuration"

25.	Navigate to the "Trusted Clients" tab

26.	Right click on "PI Adapter for OPC UA"

27.	Select "Trust"

28.	Switch back to your Atom window and look at the logs.
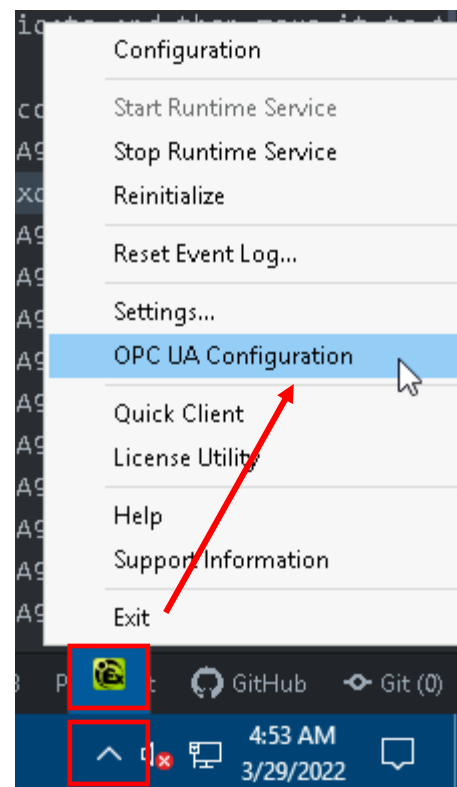
You should see a message like:

```
2022-04-11 07:54:22.367 +00:00 [Information] Successfully connected to the source endpoint:
opc.tcp://127.0.0.1:49320.
```

this message indicates that the adapter connected to the server successfully.

2.4 Discover Data Items

Once we have our data source details entered and our security configured, our next step is to ask our adapter to discover tags on our OPC UA Server.

> **Note:** Data discovery is a special step for the PI Adapter for OPC UA specifically, but some other adapters have data discovery functionality as well. Check the documentation for the adapter you are using to see if the feature is available.

29. Switch back to your PowerShell ISE and run the following command:

```
edgecmd add discoveries `
  -cid simulator `
  -id pumps `
  -query "RootNodeIds=ns=2;s=Simulation Examples.Pumps"
```

This command adds a discovery to our component. A discovery is a set of instructions for the adapter to reach out to the Data Source and discover tags that are available, based on a query.

In our case we called this discovery "pumps" and told it to search for tags under the namespace with id 2, starting at the point in the hierarchy under Simulation Examples.Pumps. These details about namespaces and OPC UA queries are beyond the scope of this course, in a real-world situation you would look at the OPC UA Server with an OPC UA Client supplied by your vendor and decide on which tags to read from. For more informamiton see the *documentation* on data scource discovery.

30. Switch to your Atom window again and look at your log file.

You'll see some log messages appear at the bottom of the file like the below:

```
2022-04-11 07:56:00.855 +00:00 [Information] Starting discovery with ID pumps and query
string: RootNodeIds=ns=2;s=Simulation Examples.Pumps.

2022-04-11 07:56:01.151 +00:00 [Information] Discovery with ID pumps - Successfully connected
to the source endpoint: opc.tcp://127.0.0.1:49320.

2022-04-11 07:56:01.231 +00:00 [Information] Discovery with ID pumps has been completed.
```

Let's check the result of our discovery.

31. Switch to your PowerShell ISE window and run the following command:

```
edgecmd get discoveries `
  -cid simulator `
  -id pumps `
  -result
```

This command retrieves the results of our discovery, which should be a whole lot of tags on our OPC UA Server. You should see a whole bunch of them listed out in your PowerShell console:

```
[
  {
    "nodeId": "ns=2;s=Simulation Examples.Pumps.Pump1.PumpSpeed",
    "selected": false,
    "name": null,
    "streamId": "2.Simulation Examples.Pumps.Pump1.PumpSpeed",
    "dataFilterId": null
  },
  {
    "nodeId": "ns=2;s=Simulation Examples.Pumps.Pump3.OutputFlowRate",
    "selected": false,
    "name": null,
    "streamId": "2.Simulation Examples.Pumps.Pump3.OutputFlowRate",
    "dataFilterId": null
  },
  {
    "nodeId": "ns=2;s=Simulation Examples.Pumps.Pump1.OilPressure",
    "selected": false,
    ...
```

This is **NOT** the configuration of the tags currently being read by the Adapter, it is only a list of tags it has discovered on the OPC Server after we told it to go and look for them. Before we apply it and start collecting data, we'd like to make some changes to the list.

> **Note:** In the future if you're configuring this on your own and want to automatically apply the data selection that your discovery found and just start reading data straight away, you can add an "-autoselect true" parameter to the end of the "edgecmd add discovery" command. Doing this will allow you to skip most of the upcoming steps but give you much less flexibility on data filtering and selecting specific tags for loading.

2.5 Configure Data Selection

We're about to configure data selection, but first let's talk about data filters. Once we tell our Adapter to start collecting data, it doesn't *have* to send *all* the data it collects. A lot of the time we don't want it to – if a valve is "open" for 3 days straight, do we really need a reading from it every second? We can configure our Adapter to only send data when it changes or set a deadband for analog data.

> 32. Run this command to see the current rules we have defined:
>
> ```
> edgecmd get datafilters -cid simulator
> ```

You should see a single datafilter listed called "DuplicateData":

```json
[
  {
    "id": "DuplicateData",
    "absoluteDeadband": 0,
    "percentChange": null,
    "expirationPeriod": "1:00:00"
  }
]
```

This datafilter is present on configurations by default. If applied, it will ensure the adapter will only send data if it has changed from its previous value but will resend the duplicate value if an hour has passed.

We'd like to edit this default filter, so users see updated timestamps for values even if they've stayed the same, at least once every 10 minutes. We want to change the expirationPeriod parameter to 10 minutes.

33. Run the following command:

```
edgecmd edit datafilters `
  -cid simulator `
  -id DuplicateData `
  -expirationperiod "00:10:00"
```

This command edits the expiration period to 10 minutes and leaves all other parameters the same. We want one additional data filter.

34. Run the following command:

```
edgecmd add datafilters `
  -cid simulator `
  -id 1UnitDeadband `
  -absolutedeadband 1 `
  -expirationperiod 600
```

This command adds a data filter that will act as a deadband with a width of 1 engineering unit and an expiration of 600s.

Note: The -expirationperiod parameter accepts a time or an integer. If you use an integer, the Adapter assumes it is a number of seconds. In the above case we could have also used -expirationperiod 00:10:00.

In other words, if a sensor reads 10 degrees on one reading of a data item with this filter selected, and 10.5 degrees on the next, the 10.5 value will not be sent to the egress server. If on the next reading it reads 12 degrees, the 12 value will be sent as it is outside the 1-unit deadband centered around 10 degrees.

Now we have a successful discovery and rules defined for data filtering, let's combine them into a data selection. We'll extract the discovery result, edit a few fields, then import them back into our adapter as a data selection.

35. Run the command:

```
edgecmd get discoveries `
  -cid simulator `
  -id pumps `
  -result `
  -file "C:\Class Files\OpcUaPumpsDataSelection.csv" `
  -csv
```

36. Open C:\Class Files\OpcUaPumpsDataSelection.csv in Excel

37. Make the following changes to the spreadsheet:

I.   Sort the spreadsheet by the "streamId" column
II.   For all rows except the streams involving the word _System, change the values in the "selected" column from **FALSE** to **TRUE**
III.   Select the "streamId" column, and use the Excel find and replace feature to remove all mentions of "2.Simulation Examples." In that column
IV.   Set the values in the "dataFilterId" column for all streams to "1UnitDeadband"

Note: The **streamId** column contains how the items will be named on the egress side (in our case, the tag names on the Data Archive), and the **nodeId** column must correspond to the name of the item on the data source.

38. Save and close the spreadsheet

39. Run the following command to import and apply your configuration:

```
edgecmd set dataselection `
  -cid simulator `
  -file "C:\Class Files\OpcUaPumpsDataSelection.csv" `
  -csv
```

40. Navigate back to your Atom Window to watch the log file.

You should see messages like the following:

```
2022-04-04 05:46:56.978 +00:00 [Information] DataSelection has been received.
2022-04-04 05:47:13.905 +00:00 [Information] Attempting to add items to the subscription.
2022-04-04 05:47:13.966 +00:00 [Information] Subscription Id: 1 contains: 30 items. Publishing
interval: 1000 ms, Sampling interval: 500 ms, Queue size: 2
2022-04-04 05:47:24.842 +00:00 [Information] Automatic history recovery - Data selection
configuration does not contain any items with user access level of HistoryRead defined..
Automatic history recovery operation for 03/31/2022 05:47:14 - 04/04/2022 05:47:14 will not
start.
2022-04-04 05:47:24.843 +00:00 [Information] Finished performing history recovery for interval
from 03/31/2022 05:47:14 to 04/04/2022 05:47:14.
```

Our configuration has been applied, and a history recovery would have been attempted – although none of the tags on the OPC UA Server support this so it didn't start. It looks the Adapter isn't complaining so we're probably reading data successfully. We'll confirm this later.

2.6 Configure Egress

There are a few options regarding where we can egress data to, but in this class, we'll configure egress to a PI Web API endpoint on a PI Data Archive.

41. Run the following command:

```
edgecmd add dataendpoints `
  -id PISRV01 `
  -endpoint https://pisrv01/piwebapi/omf/ `
  -username "pischool\\piadapter" `
  -password "P1Adapter!"
```

```
edgecmd add dataendpoints -id PISRV01 -endpoint
https://pisrv01/piwebapi/omf/ -username "pischool\\piadapter" -password
"P1Adapter!"
```

> **Note:** the account username/password combination above refers to is a service account on this domain that has already been created for us. The stored password is encrypted on the adapter itself – but is in plaintext here, and when we save our script. When running a script like this in production, the password should be parameterized and not written out in plaintext. We'll ignore this for now, that's something to think about later if applying this to your own system.

After running the above command, the adapter starts sending data and automatically creates our items on our egress server, there's no management on that side at all.

We have one more type of egress to configure, a health endpoint. The configuration is very similar to a data egress, but instead of sending data read from the source through, the Adapter instead sends health and diagnostic data about it's own components.

42. Run the following command:

```
edgecmd add healthendpoints `
  -id PISRV01 `
  -endpoint https://pisrv01/piwebapi/omf/ `
  -username "pischool\\piadapter" `
  -password "P1Adapter!"
```

We should now be up and running, sending data to the Data Archive.

2.7 Confirm Data Flow

Now that the PI Adapter is fully configured, let's check on our destination PI Server to see if new PI Tags were created and if they are updating with new data from the PI Adapter.
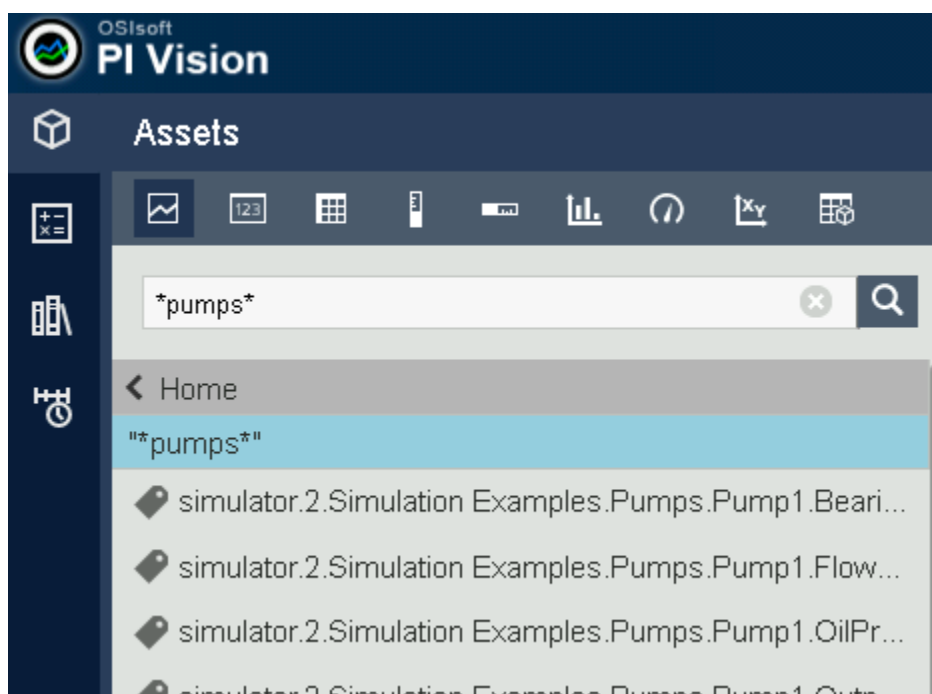
43. On your VM, open a web browser, and visit the PI Vision webpage (you'll find a bookmark for it):

   https://localhost/pivision

44. Near the top-right of the PI Vision home page, click the "New Display" button to create a new display
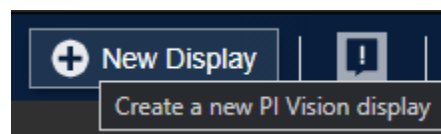
45. Using the left-hand search pane, search the VM PI System for the new Modbus tags (that were created by the PI Adapter) by searching for *pumps*

You should see the newly created tags. These tags were created by the adapter, through the OMF endpoint we set up. For adapters, tag management is not done on the PI Data Archive – each adapter creates its own tags.

46. Hold the ctrl key and select some of the tags. Drag them over to your canvas to create a trend and confirm that you are seeing data come through. Try playing with the display start time and end time at the bottom of the screen and navigate forward and backward in time.
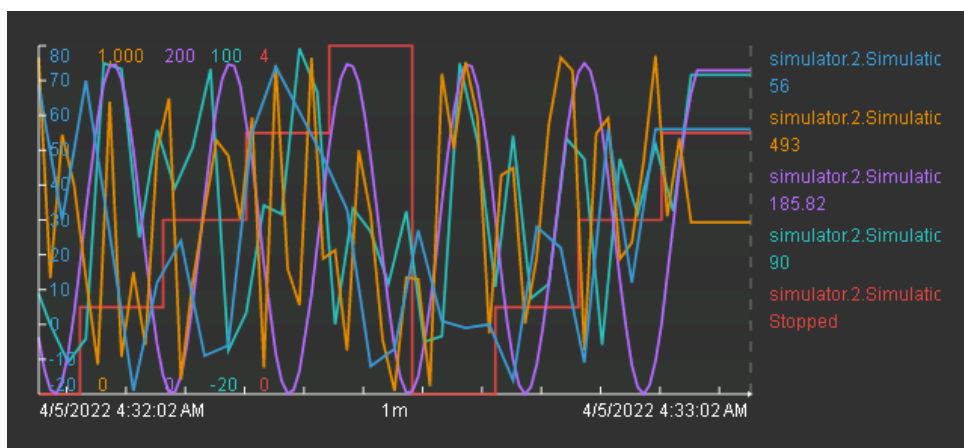
Next, let's look at our health tags.

47. Search for *opcua* in the search pane. You should find all health tags
     associated with your new Adapter listed. Drag out the following tags:
       - PISRV01.OpcUa.simulator.DeviceStatus
       - PISRV01.OpcUa.simulator.IORate
       - PISRV01.OpcUa.simulator.NextHealthMessageExpected

These three tags give us a pretty good idea of how our Adapter is running. DeviceStatus tells us if the Adapter is connected to its data endpoint, IORate tells us how many data events are flowing through every second, and NextHealthMessageExpected tells us when the next health update will be – if this doesn't change when it is expected to change, then something is wrong.

That seemed like a lot of steps to get up and running – but – now we have a script to automate most of the process, and you have the knowledge you need to modify the script for your own system. The huge benefit to using a command line tool to configure an Adapter is that we can keep a complete configuration record and reuse it on other systems.

Our script is also long and performs a lot of actions. Wouldn't it be nice it we had a configuration file we could run with a single line to apply it to another system?

48. Add the following command to your script and run it:

```
edgecmd get application `
  -file  "C:\Class Files\OpcUaAdapterConfig.json"
```

Our entire Adapter configuration is now contained in this file. We'll explore this more in the next section, where we use a pre-built configuration file for a different type of Adapter.

3. Configuring an Adapter from a Config File: PI Adapter for Modbus TCP

In this section, we are going to further explore Adapter configuration automation while we look at the PI Adapter or Modbus TCP. We'll configure the new Adapter with just a single command! … assuming we already have a full configuration file for it. We'll look at this configuration file, which should give you a good idea on how to get started implementing one yourself.

Here's a quick whiteboard on what our completed architecture will look like.
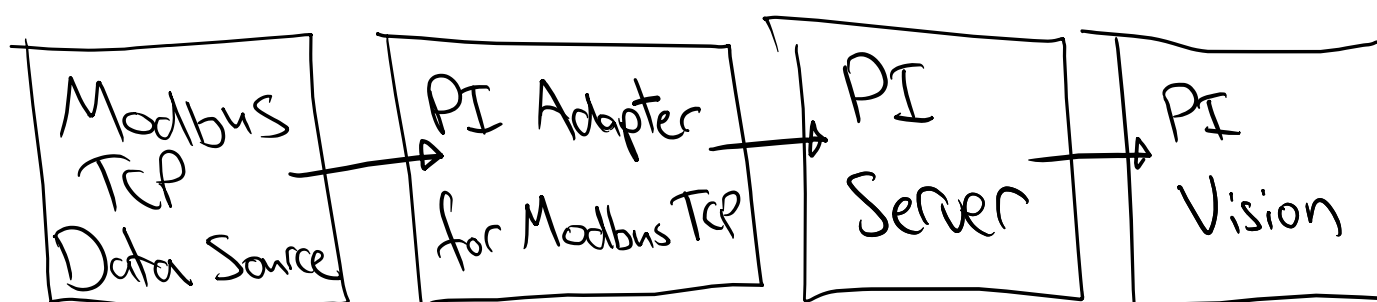
We are going to do this by using the EdgeCmd configuration utility to pass a JSON configuration file to a pre-installed PI Adapter for Modbus TCP.

The Modbus TCP communication protocol is a commonly available industrial communication protocol used for connecting and transmitting information between industrial electronic devices. The PI Adapter for Modbus TCP can connect to any device that uses the Modbus TCP communication protocol.

Before we get started configuring the PI Adapter for Modbus TCP, we'll need a Modbus TCP data source. In this case, we have a Modbus TCP data simulator available on the PISRV01 machine.

> 49. Open the "Class Files" folder on the desktop, then open
>   ModbusAdapterConfiguration.ps1

Let's look at our current configuration for the adapter.

> 50. Add this line to your script file and run it:
>
>   `edgecmd get application -port 6000`

Your adapter should have only the default configuration, with only the OmfEgress component present, but not configured.

You may have noticed we define a port here, which wasn't present on any of our OPC UA configuration. This is because the PI Adapter for Modbus TCP on this machine isn't installed on the default port.

When installing an Adapter, you are asked which port you'd like the Adapter to expose. The default port for PI Adapters is 5590, and this is the port that EdgeCmd will connect to by default. On our environment the PI Adapter for OPC UA was installed on port 5590, so we had to choose another for Modbus – port 6000. All commands we send to the PI Adapter for Modbus TCP will have this -port parameter defined.

3.1   Reading a Configuration File

Before we configure our Adapter, let's look at the pre-prepared configuration we'll use.

---

51. Open the "Class Files" folder on the desktop, then open ModbusAdapterConfig.json

---

This will open the file in Atom text editor.

We'll take a quick look through this file and get familiar with the different configuration options for the adapter. If you need additional help, you can access the PI Adapter for Modbus TCP documentation *here*, which is bookmarked in the VM's web browser).

In this configuration file, you'll see the specification of a new Modbus component with the ID of "ModbusSimulator":

```
"Components": [
        {
            "componentId": "OmfEgress",
            "componentType": "OmfEgress"
        },
        {
            "componentId": "ModbusSimulator",
            "componentType": "Modbus"
        }
    ]
```

For this new "ModbusSimulator" component, you'll see a corresponding Modbus TCP data source with a source IP address of "127.0.0.1" and port "502":

```
"ModbusSimulator": {
        "Logging": {
            "logLevel": "Information",
            "logFileSizeLimitBytes": 34636833,
            "logFileCountLimit": 31
        },
        "DataSource": {
            "streamIdPrefix": "ModbusSimulator.",
            "devices": [
                {
                    "id": "Device1",
                    "ipAddress": "127.0.0.1",
                    "port": 502
                }
            ],
            "simultaneousRequests": 1,
            "maxResponseDataLength": 250,
            "connectTimeout": "0:00:05",
            "reconnectInterval": "0:00:01",
            "requestTimeout": "0:00:10",
            "delayBetweenRequests": "0:00:00"
        }
}
```

Further down, you'll see a definition for DataFilters. We just have one data filter here, the default one:

```
"DataFilters": [
        {
            "id": "DuplicateData",
            "absoluteDeadband": 0,
            "percentChange": null,
            "expirationPeriod": "00:10:00"
        }
    ]
```

You'll also see a data selection configuration, specifying five different specific Modbus data streams to begin polling from the Modbus data source:

```
"DataSelection": [
        {
            "selected": true,
            "name": null,
            "streamId": "InboardTemperature",
            "dataFilterId": "DuplicateData",
            "deviceId": "Device1",
            "scheduleId": "schedule1",
            "unitId": 1,
            "registerType": "Holding16",
            "registerOffset": 1,
            "dataTypeCode": 20,
            "bitMap": "",
            "conversionFactor": null,
            "conversionOffset": null
        },
        {
            "selected": true,
            "name": null,
            "streamId": "OutboardTemperature",
        ……………….
```

Next, you'll see a data collection schedule specified with the ID "schedule1", with a data collection period of 10 seconds. Each item in the Modbus Adapter's data selection must be associated with a schedule. Schedules handle how often the Adapter will scan for data.

```
"Schedules": [
        {
            "id": "schedule1",
            "period": "0:00:10",
            "offset": "0:00:00"
        }
    ]
```

Finally, so that data from those five data items will end up in PI Points on the lab VM PI Server, you'll see the OMF egress configuration, with a data endpoint PI Web API target, including the endpoint address and the authentication username and password.

```
"OmfEgress": {
     "Logging": {
         "logLevel": "Information",
         "logFileSizeLimitBytes": 34636833,
         "logFileCountLimit": 31
     },
     "DataEndpoints": [
         {
             "id": "PI Web API",
             "endpoint": "https://localhost/piwebapi/omf",
             "userName": "pischool\\piadapter",
             "password": "P1Adapter!",
             "clientId": null,
             "clientSecret": null,
             "tokenEndpoint": null,
             "validateEndpointCertificate": false
         }
     ]
 }
```

3.2   Applying Configuration from a File

Now that we looked through the configuration file, let's go ahead and apply the details contained within this configuration file to the PI Adapter application configuration by using a "set" command. The command we'll run effectively applies all configuration that was in our configuration file to the currently installed Adapter.

---

52. Add the following lines to your ModbusAdapterConfiguration.ps1 file and run it:

```
edgecmd set application `
  -port 6000 `
  -file "C:\Class Files\ModbusAdapterConfig.json"
```

You should see an "Operation has been completed successfully" message. Let's see if our configuration was applied.

---

53. Run the following command:

```
edgecmd get application -port 6000
```

You should now see that the original PI Adapter application configuration has been replaced with the settings of the configuration JSON file that we just used.

Now that the PI Adapter is fully configured, let's check on our destination PI Server to see if new PI Tags were created and if they are updating with new data from the PI Adapter.

---

54. On your VM, open a web browser, and visit the PI Vision webpage (you'll find a bookmark for it):

    https://localhost/pivision

---

55. Create a new display, then search for tags with the search term *Modbus*. Drag some out and confirm data is flowing.

You should see new data updates that arrive every 10 seconds. This was defined in the "schedules" section of the configuration file we looked at. We only had one schedule defined called "schedule1" and this was the schedule defined for all our data items.
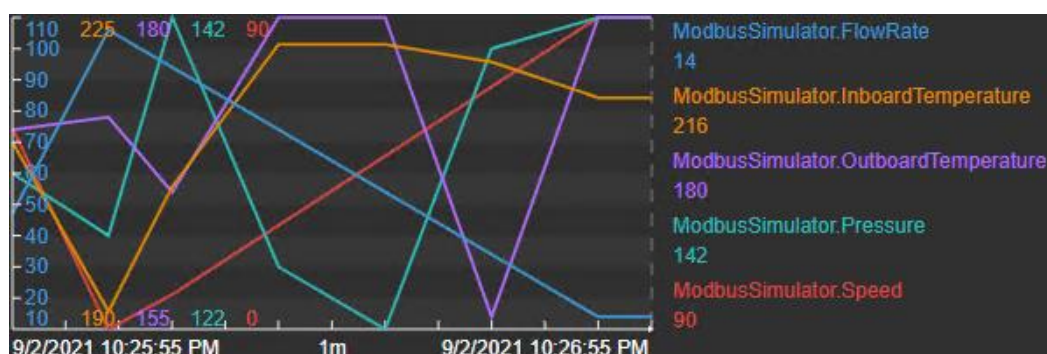
This doesn't seem nearly frequently enough for important data like this…

3.3 Editing Schedules

Let's update it to make them poll more frequently.

---

56. Add the following line to your script and run it:

```
EdgeCmd edit Schedules `
  -port 6000 `
  -cid Modbussimulator `
  -id schedule1 `
  -Period 00:00:01
```

This command edits the schedule for the "ModbusSimulator" PI Adapter component and "schedule1" schedule ID from the previously configured value of 10 seconds to a new value of 1 second.

After making this configuration change, you should next see that data with values every 1 second now appear in your PI Vision display:

We have now got a script that applies a set configuration and makes a minor change to it. If you're familiar with deployment or DevOps tooling like Azure DevOps, AWS DevOps, or Jenkins, you can imagine the simplicity of pushing out Adapter configurations in bulk. We've just done all the groundwork for a first deployment, so pushing it out to many machines or devices is a case of paramaterising sections of our script or config files, and then using deployment tools to do the deployment work for you.

4. Managing an Existing Adapter: PI Adapter for MQTT

The beauty of all Adapters sharing a common framework is that it makes it incredibly easy and straightforward to set up additional data collection to new protocols. Once you've got experience with a few adapters, they all look very similar and getting used to new ones is very quick. Let's examine just what that can look like, reviewing everything that we've learned about PI Adapters thus far and performing actions on an **already-configured** installation. Our new Adapter reads data following the MQTT (Message Queuing Telemetry Transport) protocol, common to many IIoT (Industrial Internet of Things) implementations. Below is a quick whiteboard of our current architecture:
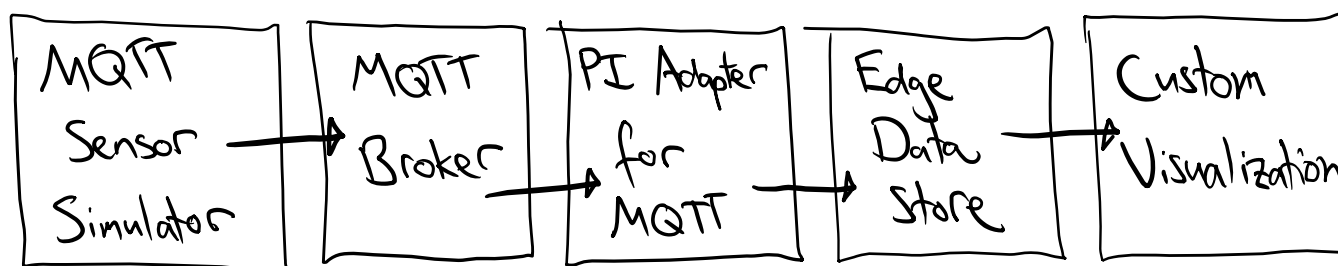
Here's we're pretending our training environment VM is out in the field – "on the edge" – and we'd like to collect data from an MQTT broker and store it locally in an Edge Data Store, for access by custom visualization software running locally on our device. If you're not familiar with Edge Data Store, you can read the documentation *here*. The PI Adapter for MQTT is bound to port 6010 on our class VM.

4.1 Exercise: Show Your Skills

Now you know all about EdgeCmd, use your new skills to write a new script to answer the following questions about the currently installed MQTT Adapter. A useful part of the documentation to reference is the article on *EdgeCmd commands*. You may also find the *PI Adapter for MQTT documentation* helpful.

Before you answer the below questions:

> 57. Open the "Class Files" folder on the desktop, then open
>   MQTTAdapterConfiguration.ps1

If you're really stuck, you can find a script to get the solutions to the below questions in the "Class Files\Solutions" folder. Remember, The PI Adapter for MQTT is bound to **port 6010** on our class VM. Add lines to this PowerShell script file to answer the following. Write down the answer in the boxes below.

- What is the ID of the MQTT Adapter component?

- At what IP Address and Port is our MQTT Broker data source located?

- How many data items is our Adapter reading from?

- The Adapter is configured to send data to Edge Data Store (EDS). At what address and port does this installation of EDS run on?

After careful consideration, you'd like the data coming from your MQTT broker to also be sent to the PI Server.
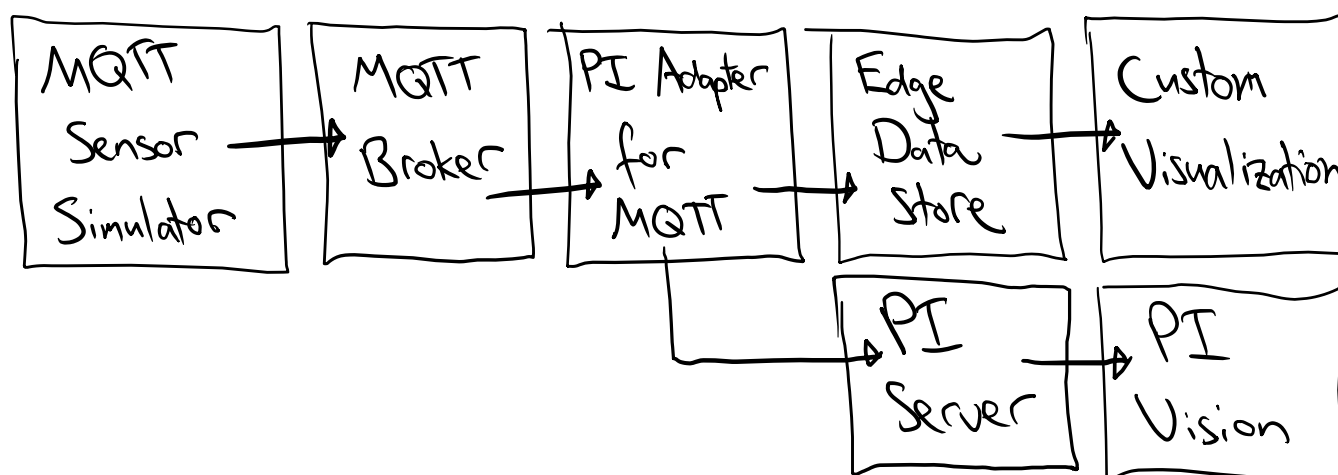

- **Challenge:** send all MQTT data to the PI Server (use the same address and credentials for the PI Server we used in the previous chapter). Confirm that you can see your new tag in PI Vision.

- **Challenge:** Configure the Adapter to send health data to the PI Data Archive **and** the Edge Data Store. Confirm that you can see your health tags in PI Vision.

And there you have it, you're now familiar enough with Adapters to go in and look at the configuration of a completely new adapter, and even change its configuration.

4.2   Other Architectures and Conclusion

We had two choices on architecture for those last two challenges. The Edge Data Store can forward data to a PI Data Archive through an OMF endpoint just like an Adapter. Like this:
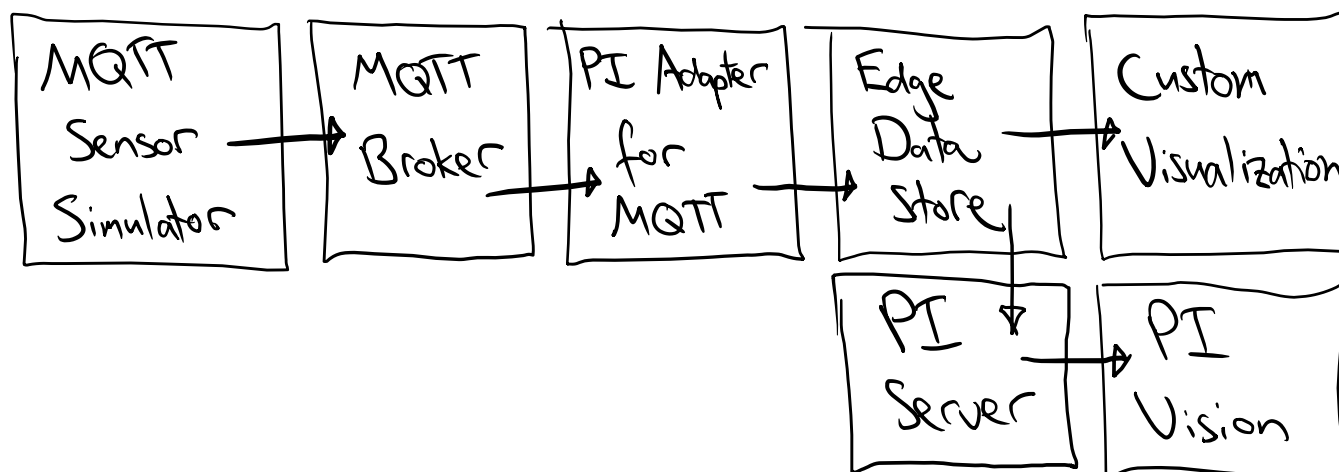
We could have used EdgeCmd to configure EDS, and the commands would have been very similar to the commands we used to configure the adapter. In essence, this architecture uses EDS as a buffer between our Adapter and the rest of our PI System. In a real-word situation, we would make a choice depending on the overall architecture of our system. Seeing as though this class focused on adapters, we did things on the adapter side, but it's possible to do it on the EDS side too.

If you still have time at the end of the lab, try to remove the configuration for the PI Server data endpoint you set up on the adapter, and try to accomplish the same thing with EDS.

Your opinion is important to us, please help us improve by openly providing feedback on your experience in the labs at AVEVA World.