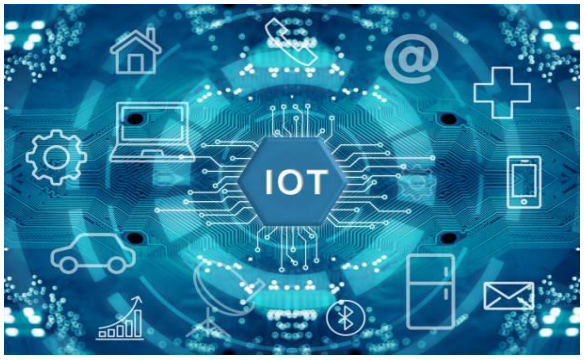
IOT MODELING WITH THINGWORX

INTERNET OF THINGS

STUDENT MANUAL

Internet of Things (IoT)

IOT is the networking of physical objects that contain electronics embedded within their architecture in order to communicate and sense interactions amongst each other or with respect to the external environment. In the upcoming years, IoT-based technology will offer advanced levels of services and practically change the way people lead their daily lives. Advancements in medicine, power, gene therapies, agriculture, smart cities, and smart homes are just a very few of the categorical examples where IoT is strongly established. Over 9 billion 'Things' (physical objects) are currently connected to the Internet, as of now. In the near future, this number is expected to rise to a whopping 20 billion.



Internet of Things (IoT) is a system of interconnected objects, usually called smart devices, through the Internet. The object can be a heart monitor, a remote, or an automobile with built-in sensors. That is objects that have been assigned an IP address and have the capability to collect and transfer data over a network.

S. No	Name of the Course	Duration
1	Fundamentals of IoT Development with Thingworx	50 Hours
2	IoT Modeling With Thingworx	40 Hours

IoT Modelling with ThingWorx

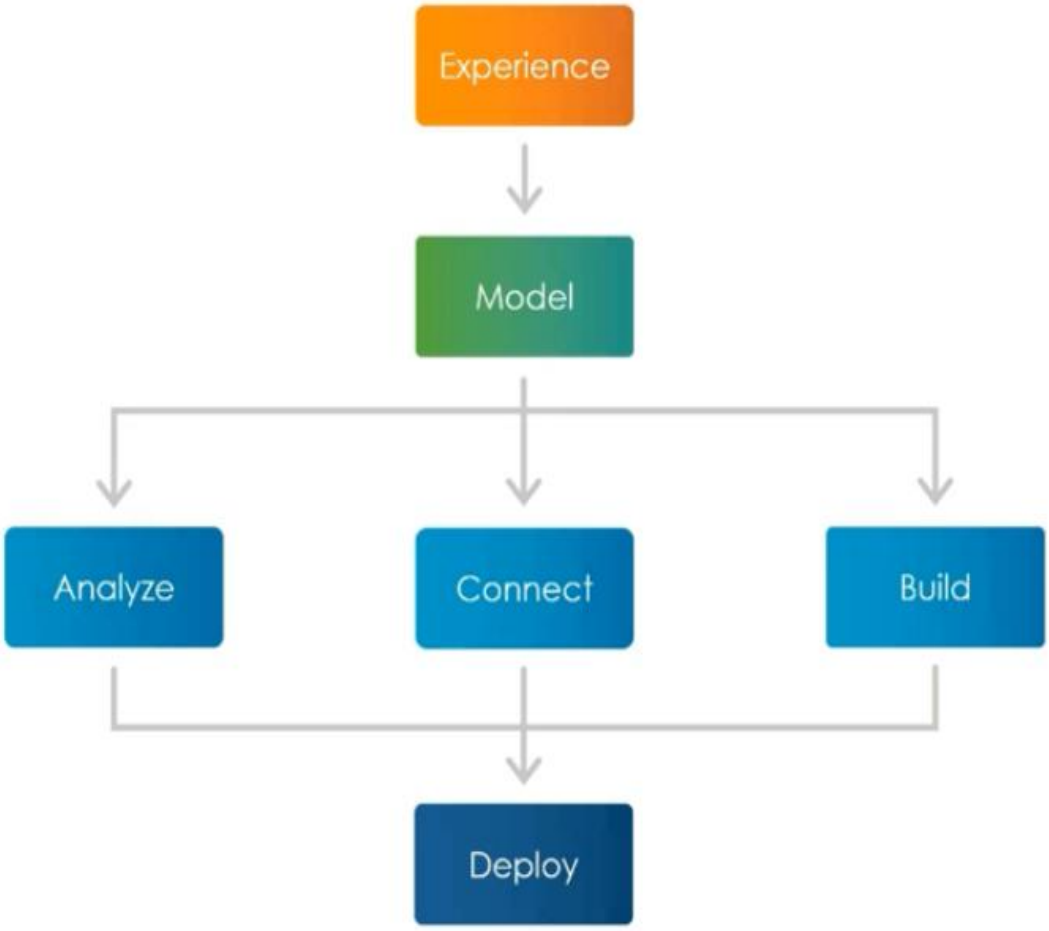
Deployments: - The ThingWorx Development Process is comprised of stages users go through beginning with conceptualization all the way through deployment.

Experience stage: -why we have to emphasize the Experience Stage so heavily? It's because the Experience Stage is where you determine why you are building an IoT application in the first place and what's going to be included. This is the critical point in the process where you define your scope – what are you building, why you're building it, and what value it should ultimately deliver. Every phase you subsequently proceed through refers back to the Experience Stage. It is the point where you will ask yourself which specific goals you are furthering by developing your project.

S. No	Name of the Course	Duration
1	IoT Modelling With ThingWorx	40 Hours

Table of Contents

1)ThingWorx development process: -



Deployments: - The ThingWorx Development Process is comprised of stages users go through beginning with conceptualization all the way through deployment.

Experience stage: -why we have to emphasize the Experience Stage so heavily? It's because the Experience Stage is where you determine why you are building an IoT application in the first place and what's going to be included. This is the critical point in the process where you define your scope – what are you building, why you're building it, and what value it should ultimately deliver. Every phase you subsequently proceed through refers back to the Experience Stage. It is the point where you will ask yourself which specific goals you are furthering by developing your project.

Within the Experience Stage, you'll ask yourself "why" questions. These "why questions" are a series of use cases or problem statements that define why you're building the "Thing,"

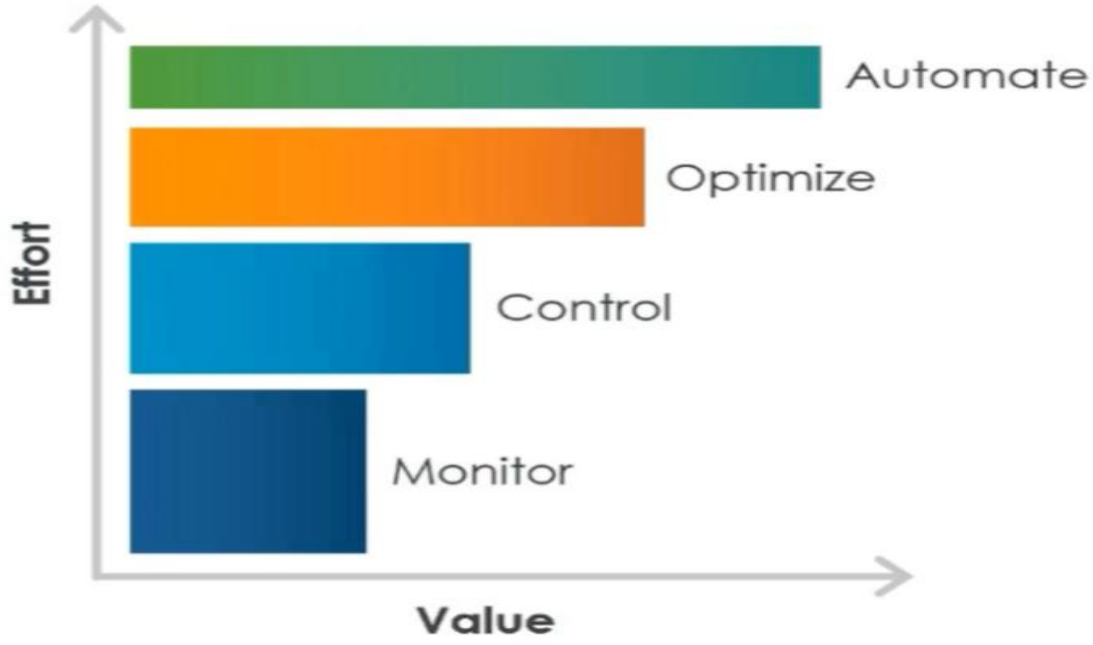
who you are building it for, and what problems it solves. Below are some common examples of "why" questions you should be asking throughout the Experience Stage:

- What is the problem that the IoT application needs to solve?
- What business value does the IoT solution offer?
- Who will use the IoT application, and how does it help them?

Next, you'll ask yourself "how" questions. "How" questions cover what capabilities the application will have to further those goals. Generally, in IoT development, there are four capabilities in escalating stages of complexity:

1. **Monitor:** How do I monitor IoT devices and dashboards that retrieve information?
2. **Control:** Once I can monitor a device, how can I control it? This involves sending instructions to the device that will be able to change its behaviour.
3. **Optimize:** Now that I can control it, how can I provide it with information to consistently generating a particular outcome? At this point you'll find it useful to leverage data analytics.
4. **Automate:** How can I ensure that my desired outcomes happen automatically without user intervention?

Review the Four Capabilities of IoT



Note that not all applications need all four capabilities. Some applications simply monitor whereas others may monitor and control. However, you cannot achieve more complex capabilities without the ones underneath it. Optimizing a device that you can't monitor is not possible.

Model Stage: - In which we develop cloud-based digital representations of our items and users and provide the functionality that allows IoT to function.

Analyse stage: - In which we give our data context, means some data analytics.

Connect stage: -Which involves connecting our gadgets to the mode

Built Stage: - User interface so that our users can interact with the model during the build stage.

Thing: -

Things are representations of physical devices, assets, products, systems, people, or processes that have properties and business logic. All Things are based on Thing Templates (inheritance) and can implement one or more Thing Shapes (composition). It is a best practice to create a Thing Template to describe a Thing, and then create an instance of that Thing Template as a Thing. This practice leverages inheritance in your model and lowers the amount of time you spend maintaining and updating your model.

Thing Templates: -

Thing Templates provide base functionality with properties, services, events, and subscriptions that Thing instances use in their execution. Every Thing is created from a Thing Template.

A Thing Template can derive one or more additional characteristics by implementing Thing Shapes. When you make a change to the Thing Template, the change is propagated to the Things that implement that Thing Template, simplifying the model maintenance.

Thing Shapes: -

Thing Shapes provide a set of characteristics represented as properties, services, events, and subscriptions that are shared across a group of physical assets. A Thing Shape is best used for composition to describe relationships between objects in your model. Thing Shapes promote reuse of contained properties and business logic that can be inherited by one or more Thing Templates. In ThingWorx, the model allows a Thing Template to implement one or more Thing Shapes.

When you make a change to the Thing Shape, the change is propagated to the Thing Templates and Things that implement that Thing Shape, simplifying the model maintenance.

Data shapes: -

Data Shapes represent the data in your model. A Data Shape is a named set of field definitions and related metadata. Each field in a Data Shape has a data type. ThingWorx has a defined set of base types.

Data Shapes help you to create applications, because when an application consumes data, with the Data Shape definition, the application has built-in knowledge of how to represent the data set. For example, if you are putting data into a grid on a mashup, the grid knows what the data is like because of the Data Shape definition. The grid knows which fields are numbers, strings, or dates. When you configure how the grid should render the data, this knowledge of the data set makes the configuration much easier.

Why IOT is complex: -

When we are going to design a IOT application there are so many doubts in our mind What are the smart device and they will have transferred data to cloud?

What are medium being there?

What will be logic and analytics with data?

Is it possible to communicate with other clouds services?

Who are our users?

We have to think about all of this questions and then we have to find the solution for that.

That's why IOT application designing is complex task.

IOT Modelling process: -

This is the Guide to our IOT Modelling Process.

This are the different IOT modelling process steps to build an IOT application.

IoT Modelling with ThingWorx

Project Introduction:

Projects are used to organize entities within ThingWorx. Entities that are collected in projects can be exported and imported together. An entity can belong to only one project.

Creating Projects:

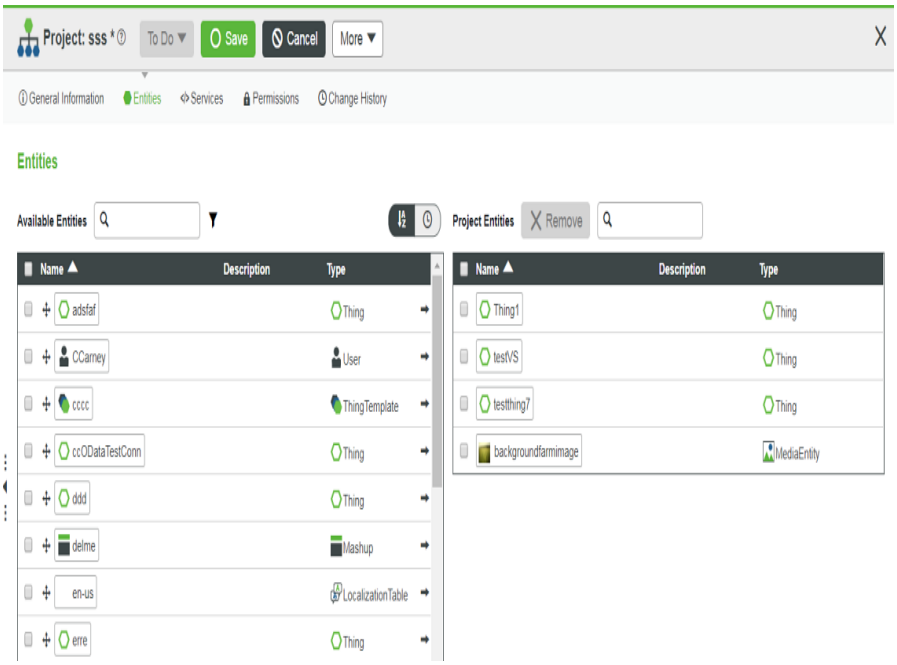
1. From **Composer**, browse **Projects**, and then click the + icon to create a new project.
2. Enter a name and description for the project.
3. Optionally, add appropriate tags and select a home mashup.
4. Specify a list of other projects that the project depends on using the **Dependencies** field.

It is not recommended to assign an empty project to another project using the **Dependencies** field. If the empty project is deleted after it has been assigned to a project, errors may occur when importing or exporting the existing project.

5. To add entities to a project, in the **Entities** area, select or search for entities from the **Available Entities** list, and then drag and drop them to the **Project Entities** area.

For a new project, in the **Available Entities** list, a list of entities that have already been assigned to another project are displayed. If you add them to a new project, then they are re-assigned and removed from the previous project.

Using the filter ( ) icon, you may also filter the entities based on the type, project, tags, Thing Template, Thing Shape, or descriptions.

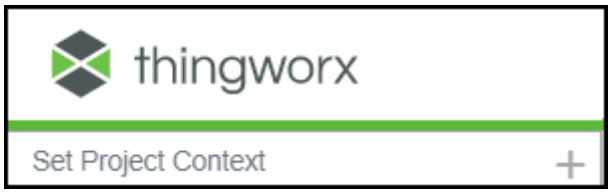


6. Optionally, click the sort ( ) icon to sort the entities or click the timer ( ) icon to view recent entities.

- To remove an entity from the project entities, select an entity and then click **Remove**. You may select and remove multiple entities at a time. The removed entity again appears in the available entities list.
- You can delete or export a project by clicking the **More** button.

7.Click **Save**.

Project Context

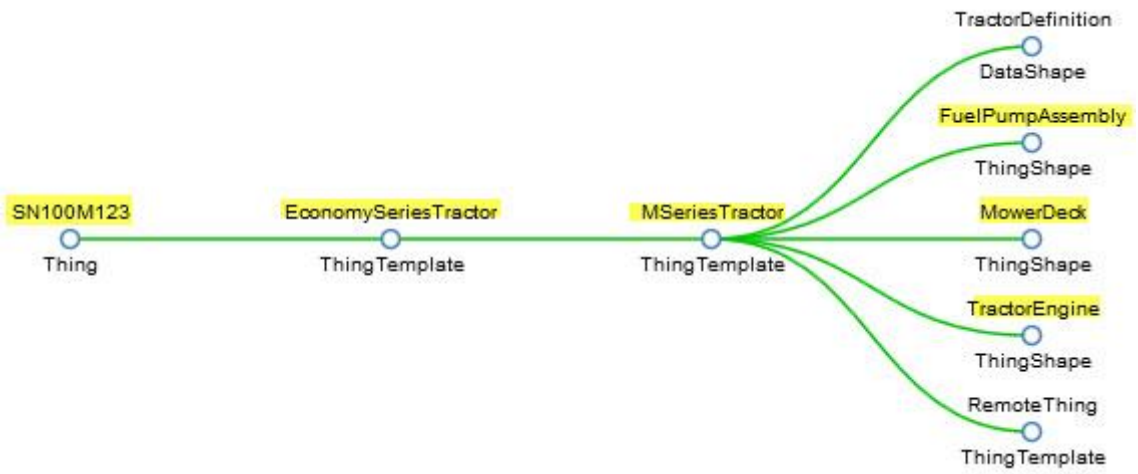


You can use the **Set Project Context** option at the top left of Composer to set a default project. This is helpful to manage any subsequent entities created in your model. If a project is set via **Set Project Context** and you then create entities, they will automatically be added to this project.

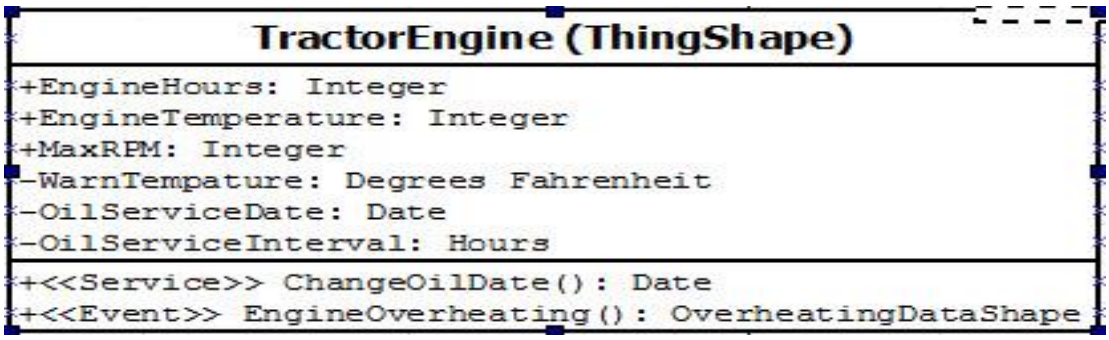
Model Example

Consider a hypothetical company named Acme Mowers that sells and services residential riding lawn tractors. Acme Mowers is introducing a new line of smart and connected tractors and is using ThingWorx to build solutions for their customers and dealers. To accomplish this, they must start by creating a ThingWorx model of their new tractors.

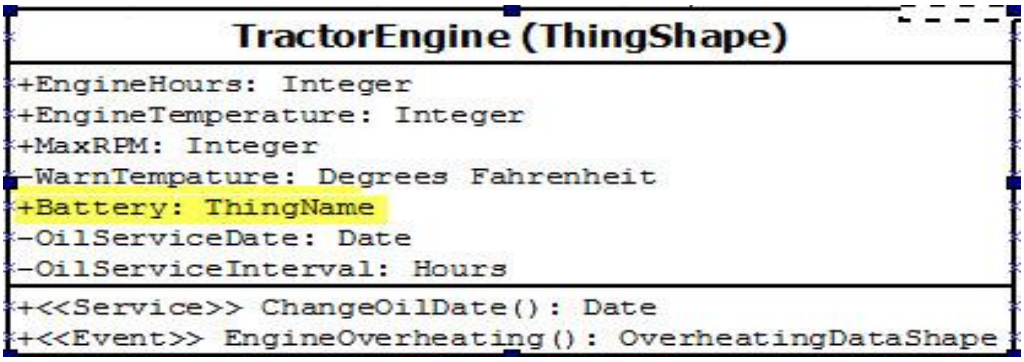
Using Things, Thing Shapes, and Thing Templates, they can construct a complex model. For example, Acme Tractors can have Serial Number SN100M123 (Thing) that is an Economy Series Tractor (Thing Template) and an MSeries Tractor (Thing Template) that has a Fuel Pump Assembly, Mower Deck, and Tractor Engine (Thing Shapes).



These entities have properties (for example, Tractor Engine has Engine Hours, Engine Temperature, and Marx), and their state changes may trigger events (for example, ChangeOilDate service and Engine Overheating alarm) that are handled by subscriptions.

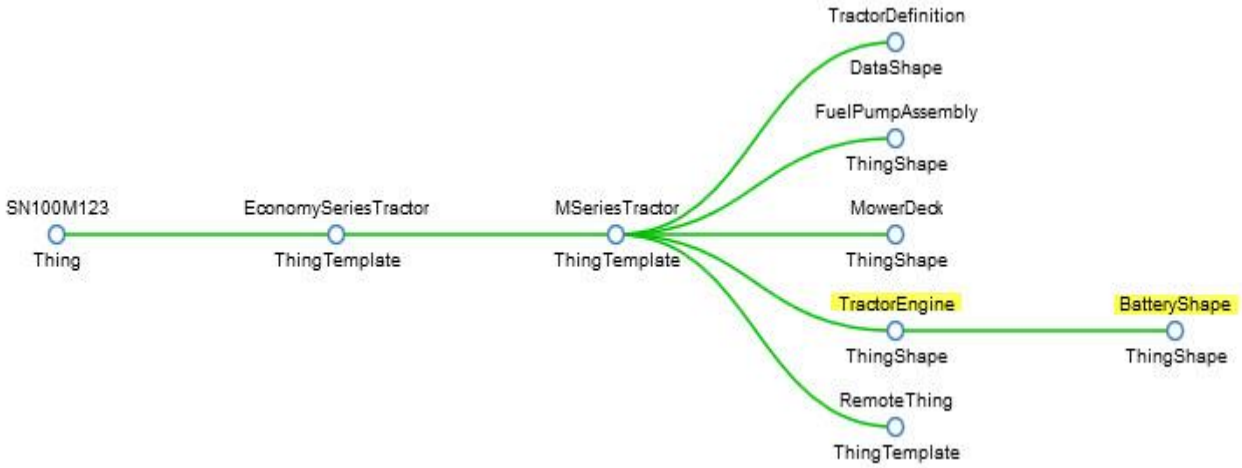


The model can become more intricate when subsystems, assemblies, and components have separate life cycles, need to be tracked and analysed independently, or have interchangeable options. For example, you can enhance the above model by adding a new property called Battery to the Tractor Engine. By defining the Battery property type as a Thing Name you can set the value of the property to a specific name of a Thing.



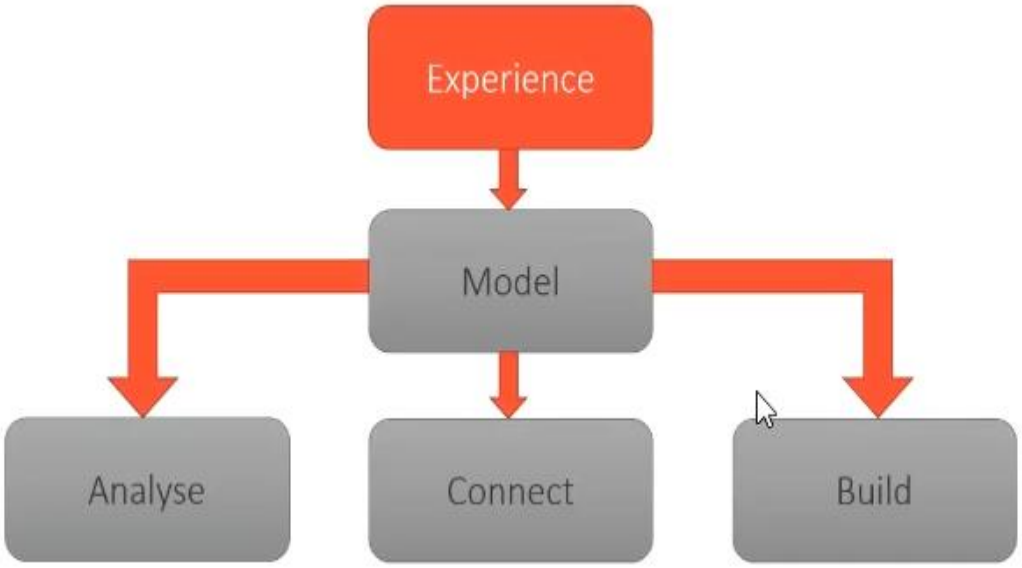
For example, you can add a Battery Shape Thing Shape, a Top Terminal Battery or SideTerminalBattery Thing Template, and finally the Things Size65TopTerminal or Size75TopTerminal. Depending on the battery installed for a given tractor engine, the Battery property value will be the name of appropriate battery Thing. Size75TopTerminal battery implements a TopTerminalBattery Thing Template which is composed of the Battery Shape Thing Shape.

Battery Shape becomes the relationship between your tractor and your battery.



Specific batteries can be tracked separately from the mower, so when batteries are changed, only their relationship with the engine needs to change.

MARS POWER SYSTEM PROJECT:



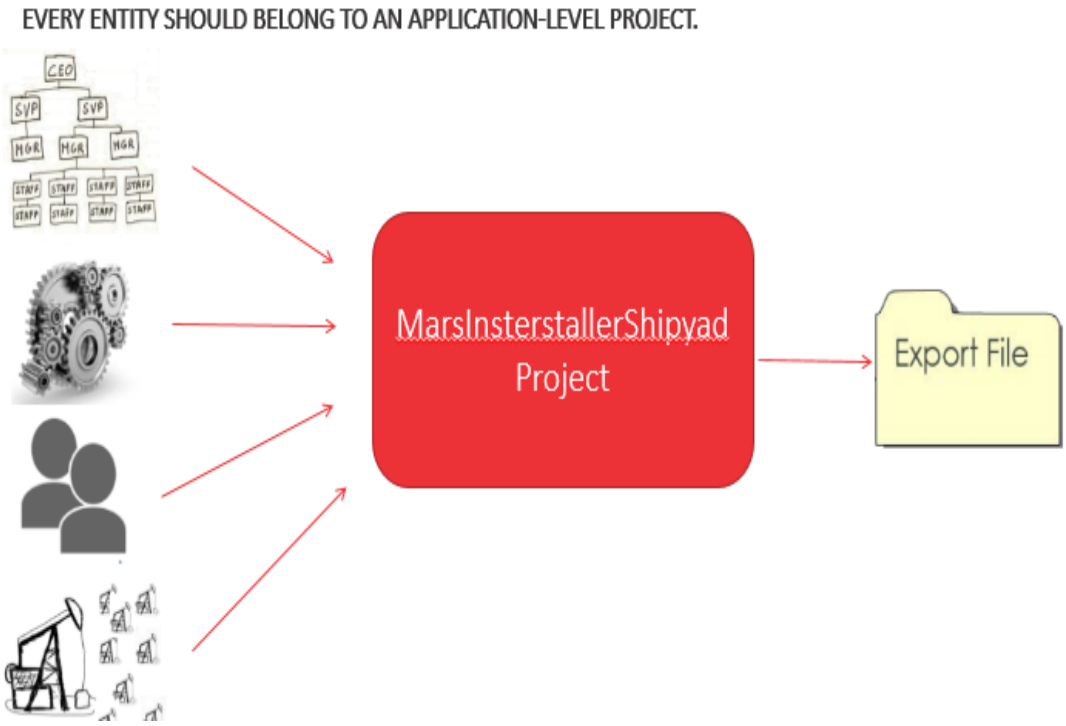
The groundwork of defining the scope of the application is done. There are three roles:

1. For Karen Lee, the Power administrator: – Monitors power assets. – Can conserve power by shutting down individual power assets. – Can put the system into a low-power emergency state to avoid brownout or blackout.

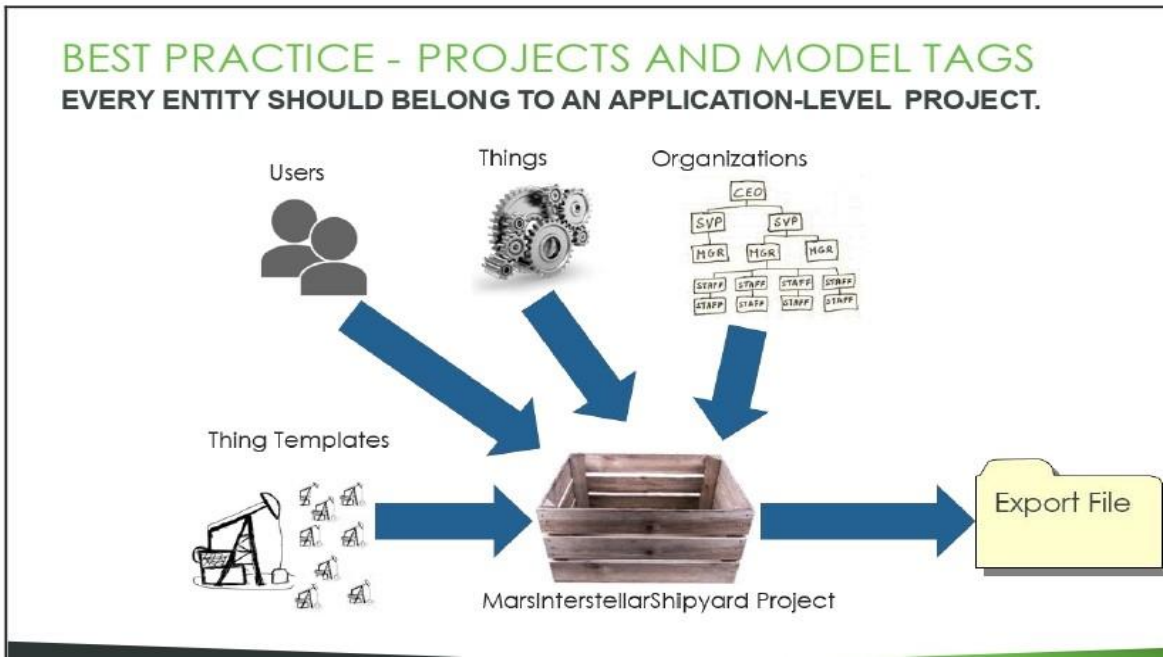
2. For Hector Conrad, Karen's boss: – Monitor high-level power usage as an aggregate – Monitor a calculated estimate of time before power runs out.

3. Automatically: – Detect when a blackout is immanent, and shut down power consumers to avoid it

Best Practice - Projects and Model Tags



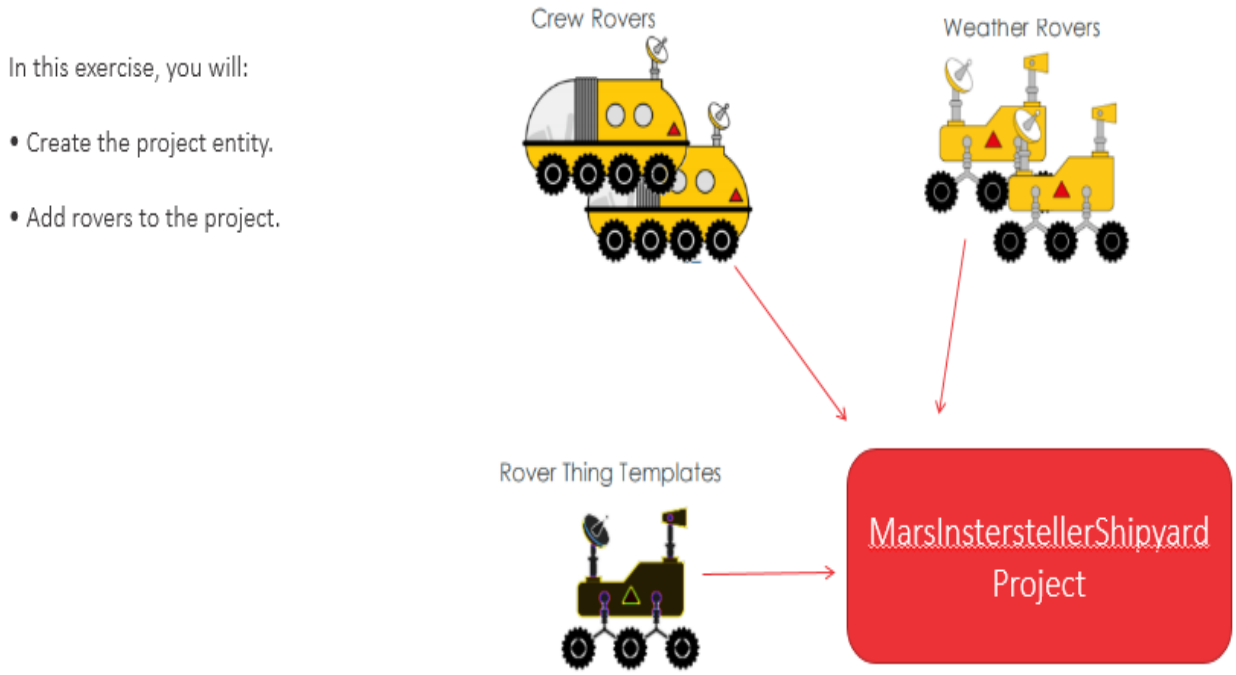
Best Practice - Projects and Model Tags



Remember when I told you we violated several best practices in the Fundamentals course? Well, one thing we didn't consider is application

portability. How do we move our application from a development environment to a production environment? Project entities help us do that. A project entity is an entity in ThingWorx used to collect up other entities. You can think of it as a box. In this course, we're creating the Mars Interstellar Shipyard project. Every entity we create will be part of that project all our thing templates, users, things, organizations, and so on. That way, when it's time to move our application to another system, we can export our project, and import it into another ThingWorx system. Model tags are an alternative to Projects, and are also used to organize entities. They are mostly used in older ThingWorx applications, since projects did not exist in earlier versions of ThingWorx.

Exercise 1: Create a ThingWorx Project Exercise



In this exercise, you will create Mars Interstellar Shipyard project, Which will store all the entities we create in this cour

Task1: Logon and validate, you can use ThingWorx environment

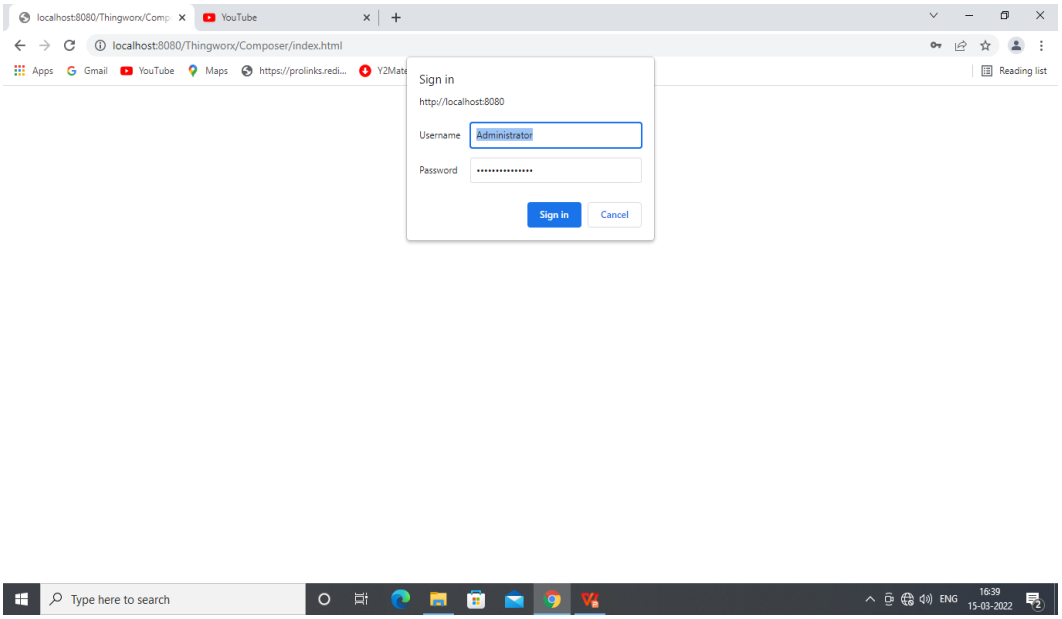
- Open Google chrome and navigate to the Server IP Address: port Number/ThingWorx/

Eg:192.168.1.29:8080/ThingWorx/

Enter username and password

User name: Administrator

Password: Thingworx@12345



Task2: - Create Mars Interstellar Shipyard project entity

1. Click +New at the top and select Project to create a New Project.
2. Type MarsInsterstallerShipyardProject in name field.
3. Click Save button.

Task3: - Set the project context

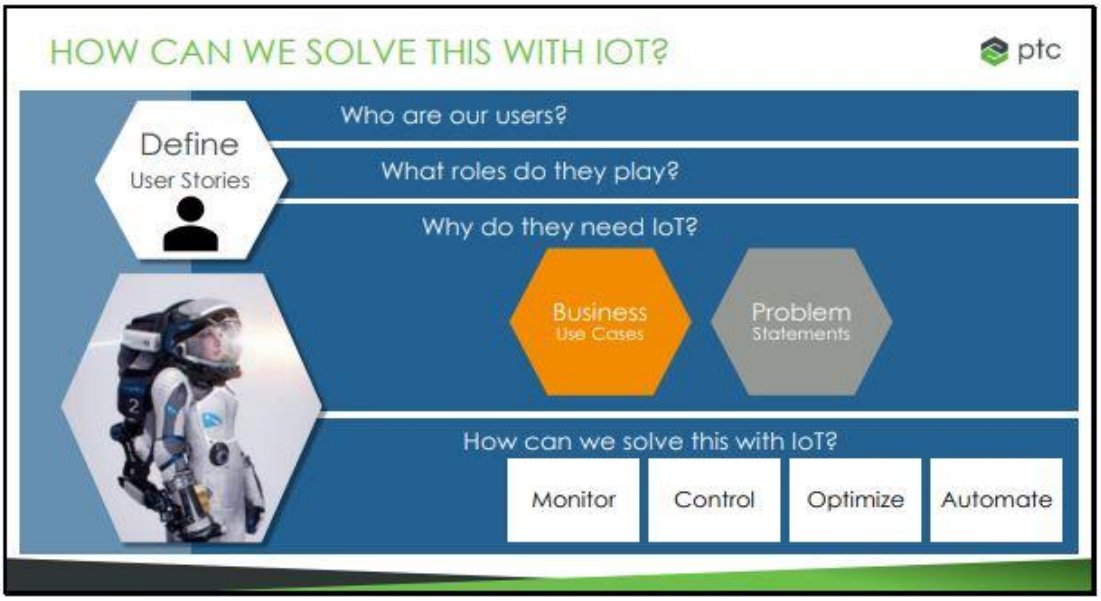
- In the set project Context field in upper left corner of Thing Worx Composer type/select MarsInsterstallerShipyardProject.

USERS AND ORGANIZATIONS

Users have modelled objects in ThingWorx. In order to model them in a way that they can be given the functionality they need, we need to understand those users. That is what this first phase is all about. Understanding our users, and modelling them in ThingWorx.

- Monitor power assets.
- Shut down power producers (control).
- Determine what power consumers are critical (optimize).
- Automatically shut down non-critical power consumers in an emergency (automate).

- Monitor the health of the power system as an aggregate.
- Warn Hector when the power system situation is a threat to production (optimize).

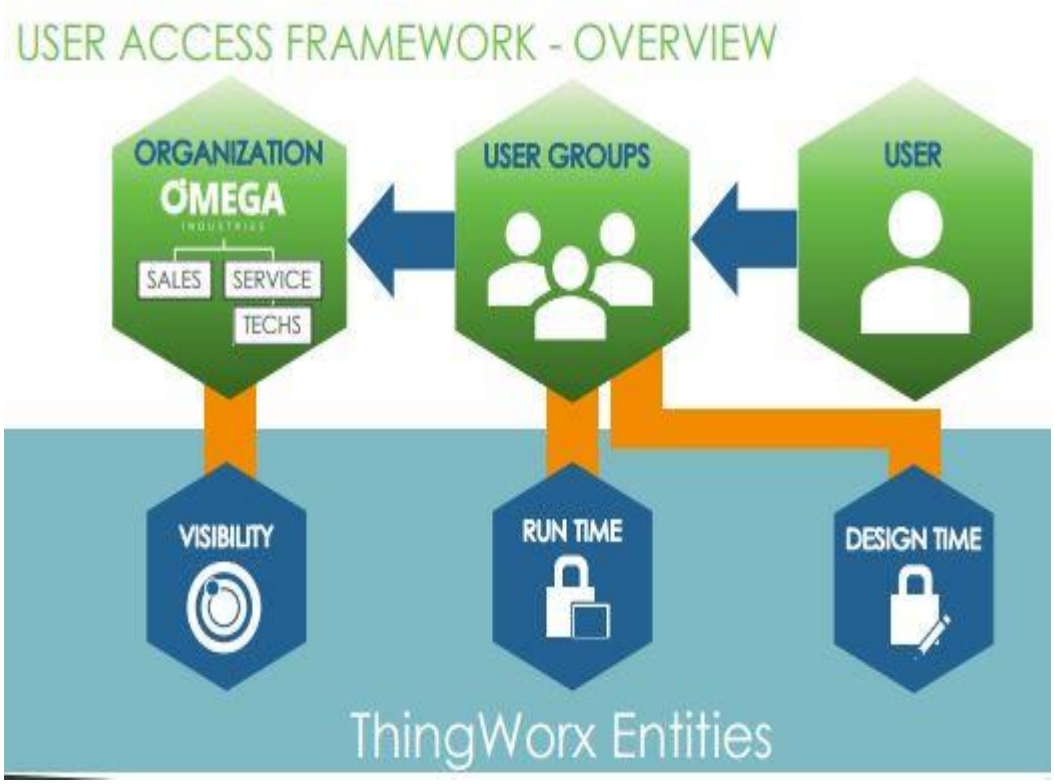


Once we know what our users need, we can apply the four capabilities of IoT – Monitor, Control, Optimize, and Automate. This tells us how we are going to fill those needs.

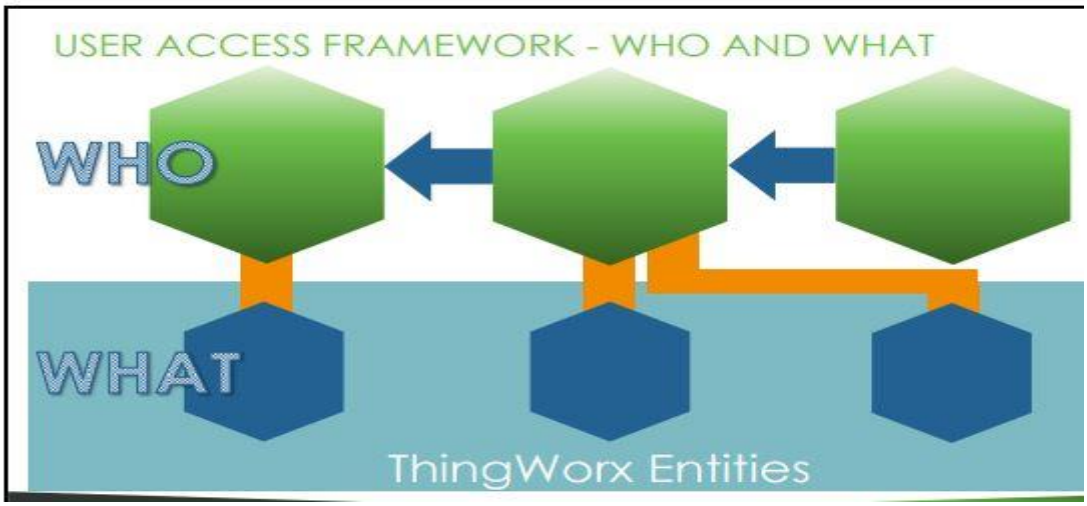
So, what can IoT do to help? Karen is obviously the biggest user of this system,

with the most needs. She needs to monitor and individually control every asset, determine what power consumers are critical, shut the non-critical ones down in an emergency, and determine when the power emergency is over.

Hector's needs for the power IoT system are more modest – he just needs rollup data to determine if power system problems are a threat to production. His dashboard may be more complex, including production, supplier, and logistics data, but that is beyond the scope of the application we are building in this course. We're just dealing with the power system
When we first consider users, we should also consider access control. This framework will guide us through understanding ThingWorx access control.



When we first consider users, we should also consider access control. This framework will guide
us through understanding ThingWorx access control. It's a best practice to deal with this up-front



Like everything in ThingWorx, access control is thing-centric, so when a user does anything, they are doing it to a specific ThingWorx entity – usually a Thing, but sometimes another entity such as a Thing Template or subsystem.

Our top-level "who" entity is the Organization, which generally includes every user that can access the system; let's say it's Omega Industries. Although it is possible for a single ThingWorx system to have multiple organizations, the typical configuration is to have only one.

Users have modeled objects in ThingWorx. In order to model them in a way that they can be given the functionality they need, we need to understand those users. That is what this first phase is all about. Understanding our users, and modeling them in ThingWorx.

Organizational Units

Each organization may have organizational units, sub-dividing the organization. It looks very much like an organization chart but doesn't need to be structured that way. It can be structured by department, facility, or any way the administrator desires. Organizational units inherit upwards. So, if someone is a member of the Techs unit, they are automatically a member of the Service unit, and of Omega Industries.

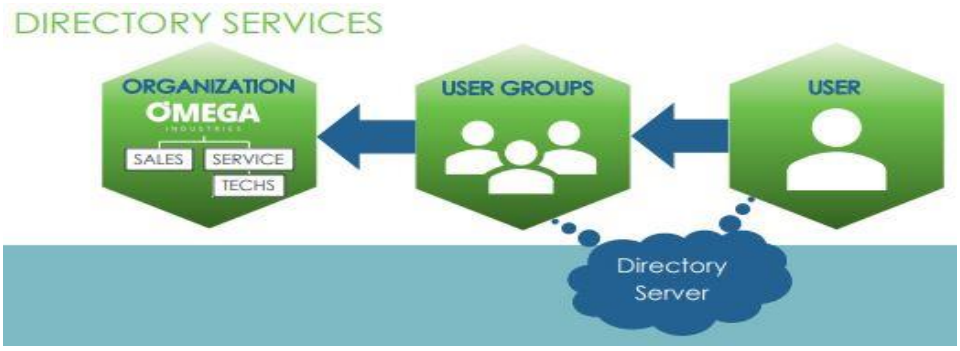
Users

A user is exactly what it sounds like. A person who can log in to the ThingWorx system, complete with a username and a password.

User Groups:

A group is a collection of users. They may be divided logically by department, but it is a best practice to divide them by role and the type of access they need. Also, user groups are assigned as members of organizational units. So, to recap, a user is an individual login.
A user group is a collection of users that have similar access control.
User groups are members of organizations and organizational units.

Directory Services:



Optionally, users and user groups may be connected to a directory server such as Microsoft Active Directory. If your system is configured this way, passwords will be managed in the Directory Server instead of ThingWorx, and ThingWorx users may be automatically created based on the users in the Directory Server. However, we won't be using a directory server in this course. We will keep all user and group information locally in ThingWorx.

Modeling | Module 2: Mars Power System Project

Exercise 1: Create a ThingWorx Project Instructor

In this exercise, you will create the Mars Interstellar Shipyard project, which will store all the entities we create in this course.

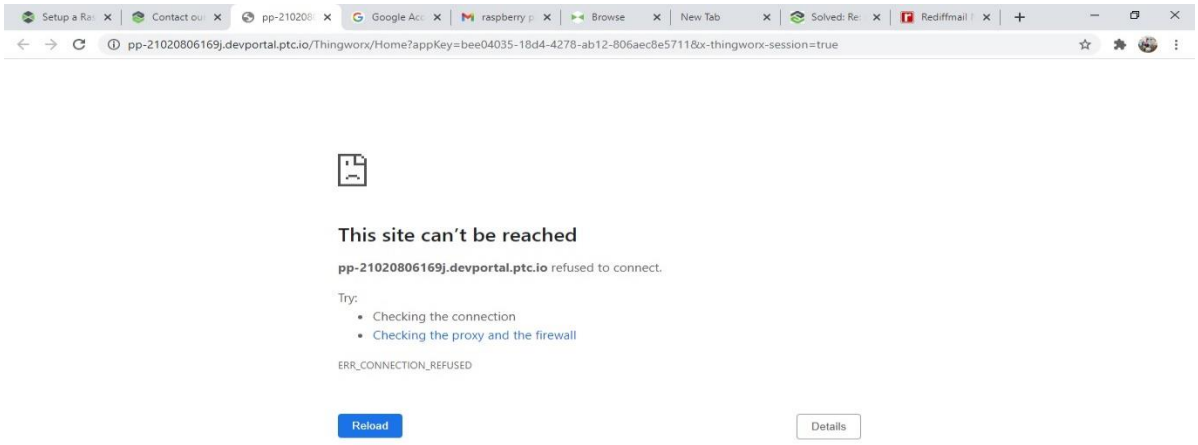
Objectives: After successfully completing this exercise, you will be able to:

- Create a ThingWorx project entity.
- Set the project context.

Task 1: Log on and validate you can use the ThingWorx environment

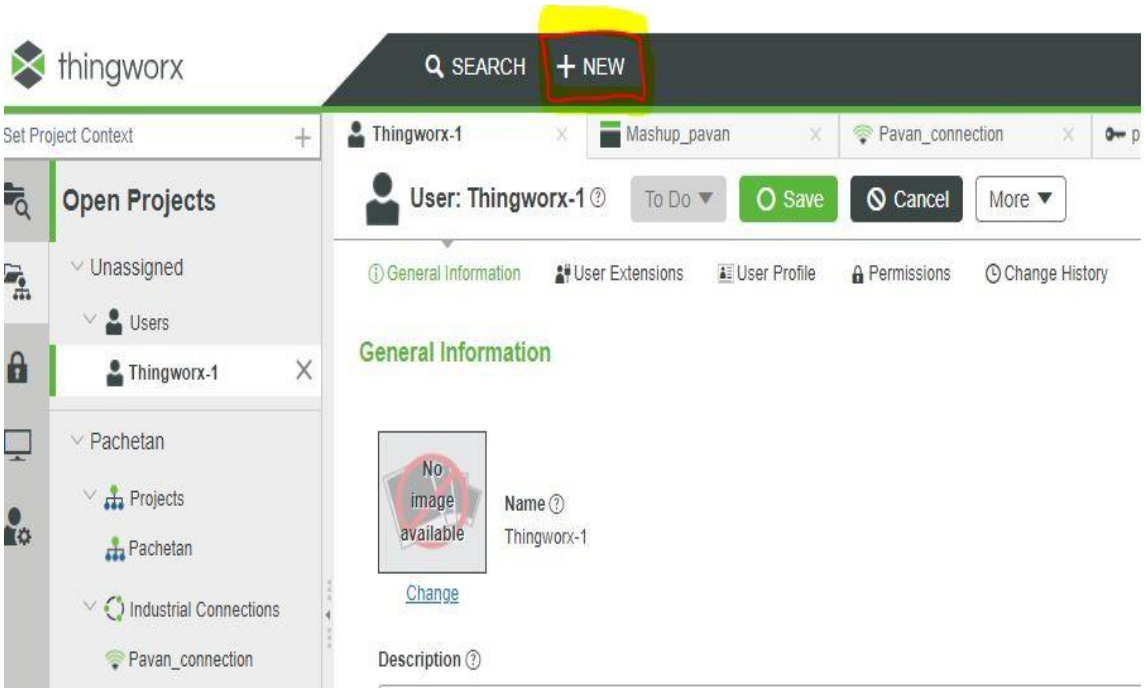
1. Double-click ThingWorx Composer on the VM's desktop or open Google Chrome and navigate to http:///Thingworx/Composer. The server name will be provided by the instructor. 2. Log on to ThingWorx Composer with the username Administrator and the password IoT foundations.

http://localhost:8080/Thingworx/Composer/index.html#/modeler/browse/User

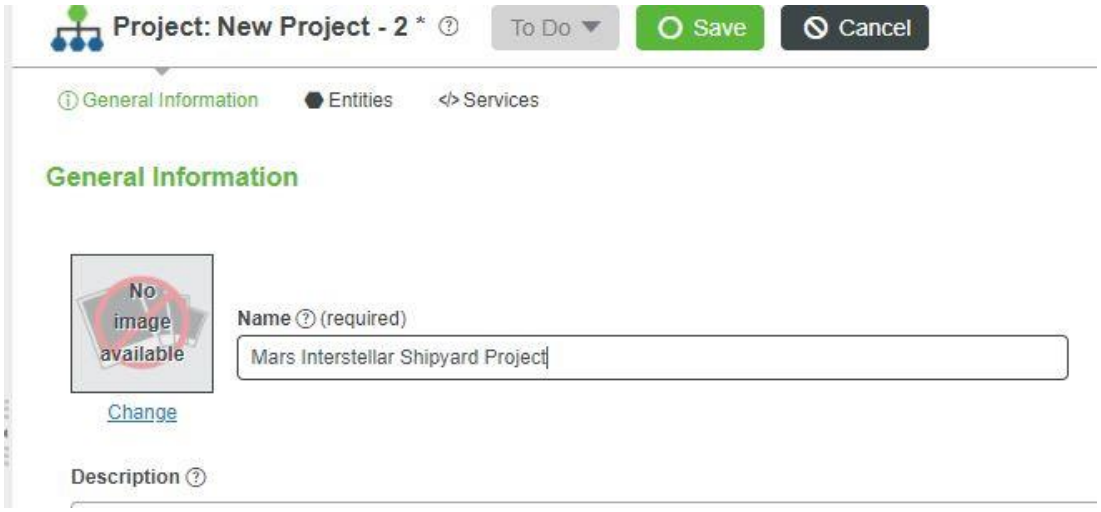


Task 2: Create the Mars Interstellar Shipyard project entity.

3. Click +NEW at the top and select Project to create a new project.



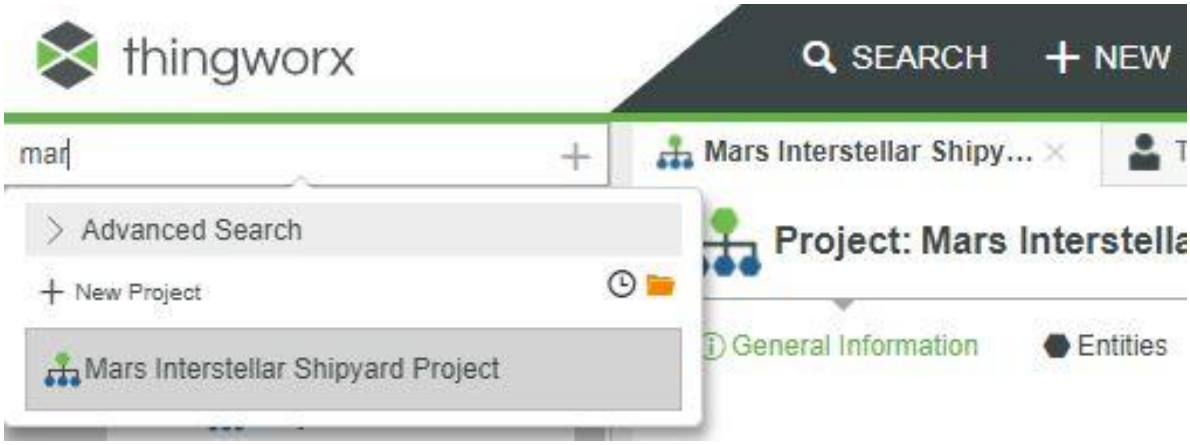
4. Type Mars Interstellar Shipyard Project in the Name field.



4. Click Save.

Task 3: Set the project context.

1. In the Set Project Context field in the upper-left corner of ThingWorx Composer, type/select Mars Interstellar Shipyard Project.



Note: Once the project has been set in the Set Project Context field, each of the new entities you create will automatically be assigned to this project context.

Modelling | Module 3: Users and Organizations Exercise

1: Create the Shipyard Organization

In this exercise, you will create the Mars Interstellar Shipyard organization, which will be used to control visibility to the shipyard entities.

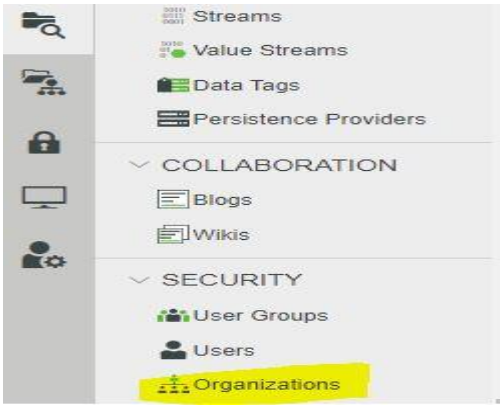
Objectives:

• Create a ThingWorx Organization.

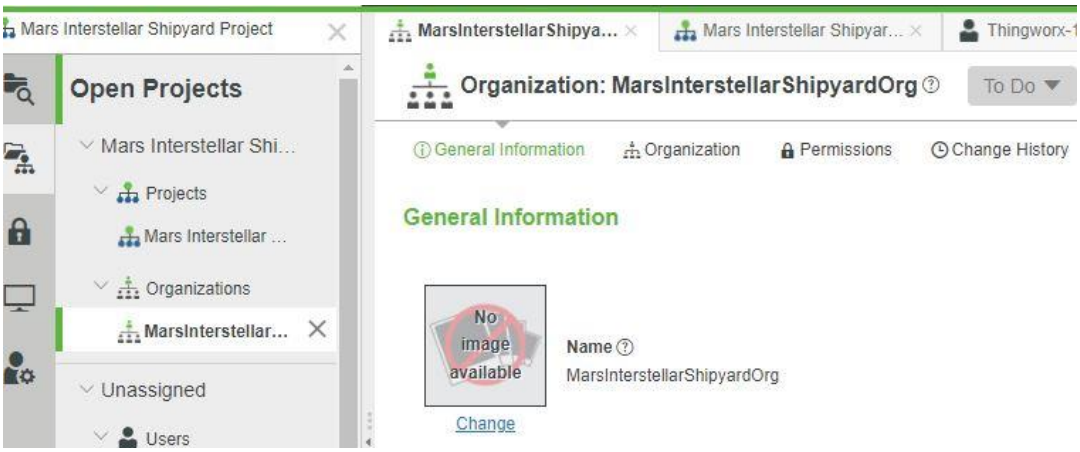
• Set visibility on a ThingWorx entity.

Task 1: Create the Mars Interstellar Shipyard Organization

1. Create a new organization entity.



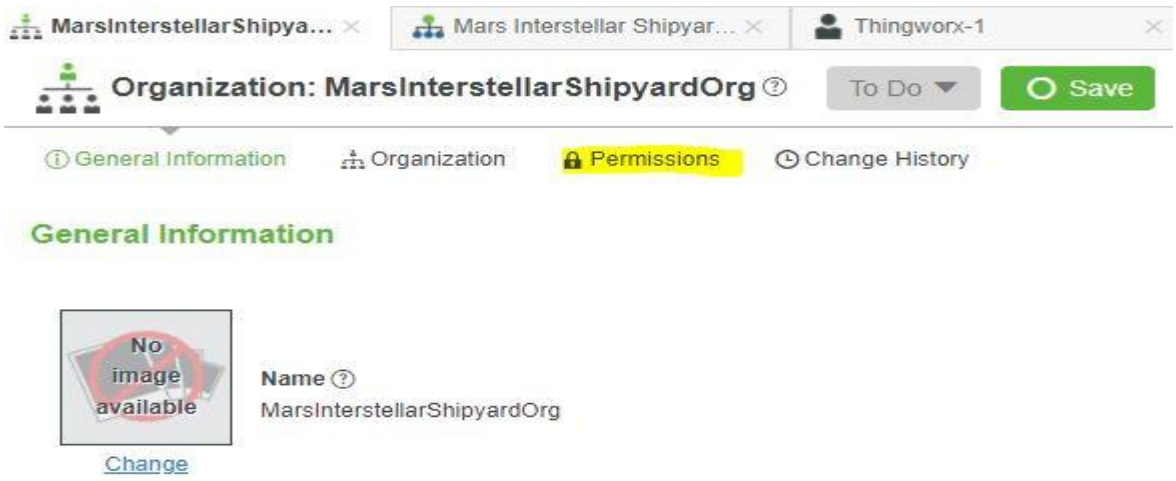
2. Type Mars Interstellar Shipyard Org in the Name field.



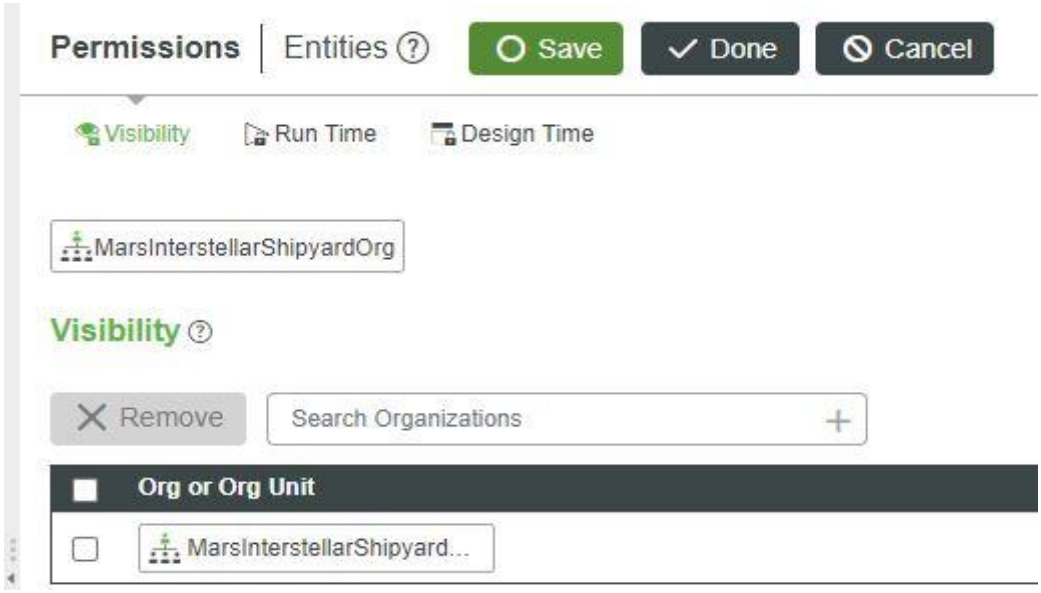
3. Click Save.

Task 2: Set visibility to the new entity

4. In the Entity Tabs section on the top, click Permissions.



5. In the Search Organizations field type/select Mars Interstellar Shipyard Org.



6. Click Save.

Exercise 2: Create Organizational Units

In this exercise, you will create two organizational units for the Mars Interstellar Shipyard organization. One unit for the Power administrators, and another for the Shipyard administrators.

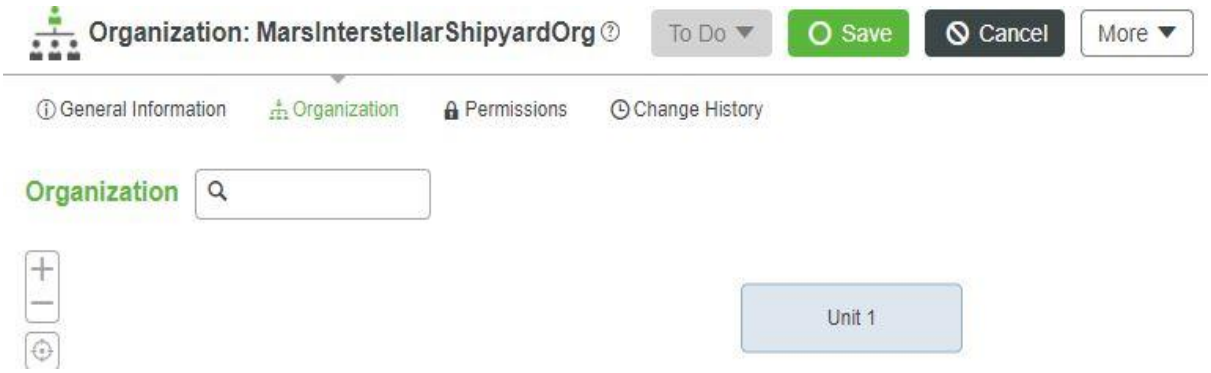
Objectives: After successfully completing this exercise, you will be able to:

• Create an organizational unit.

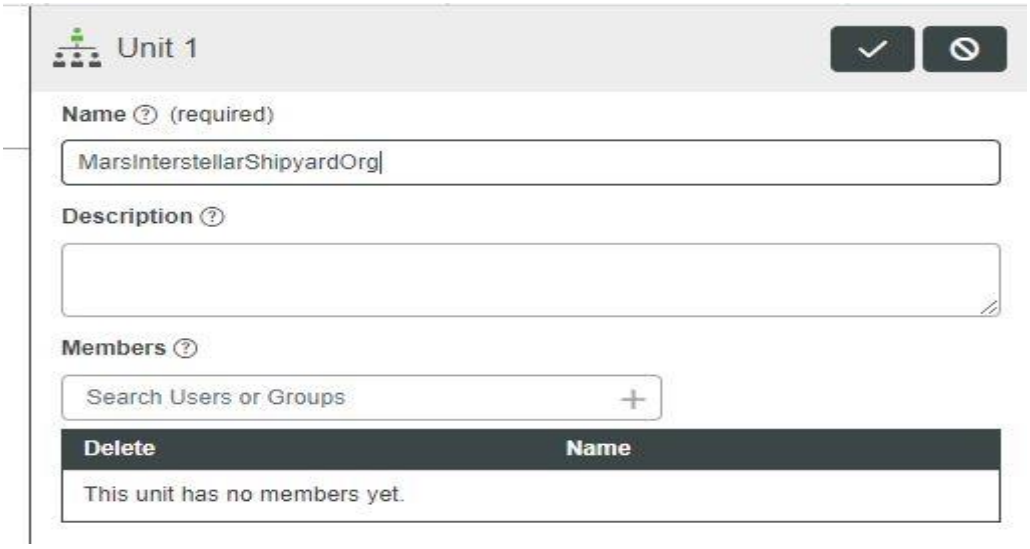
Task 1: Create the Power organizational unit

1. In the MarsInterstellarShipyardOrg entity, navigate to the Organization tab.

Instructor Note: This user interface was not covered in Fundamentals – go through organizations and organizational unit creation in a bit of detail.

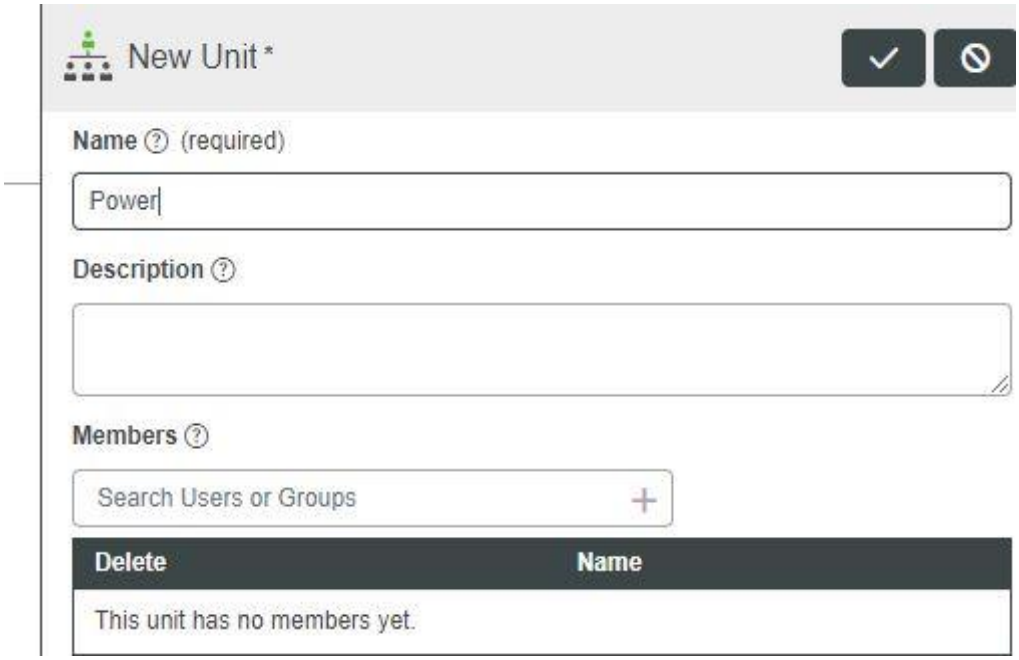


2. Click Unit 1.
3. Change the name to MarsInterstellarShipyardOrg.



Instructor Note: It is a best practice to name the top-level organizational unity the same name as the organization entity.

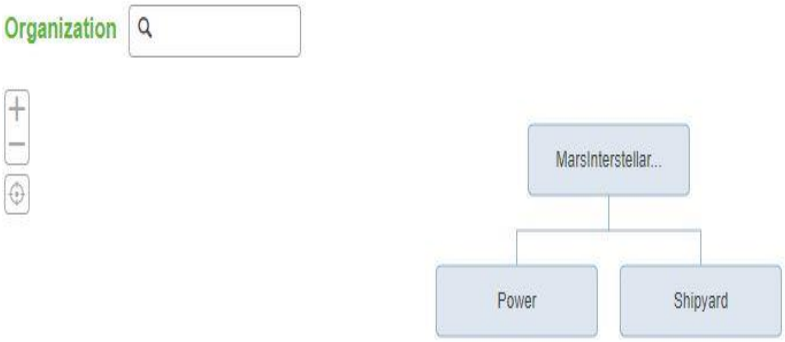
4. Click Done.
5. Hover over the MarsInterstellarShipyardOrg button and click the button underneath it.



6. Type Power in the Name field.
7. Click Done.

Task 2: Create the Shipyard organizational unit

8. Hover over the Mars Interstellar Shipyard Org button and click the button between Mars Interstellar Shipyard Org and Power.
9. Type **Shipyard** in the Name field.
10. Click Done.



11. Click Save.

Exercise 3: Creating Users

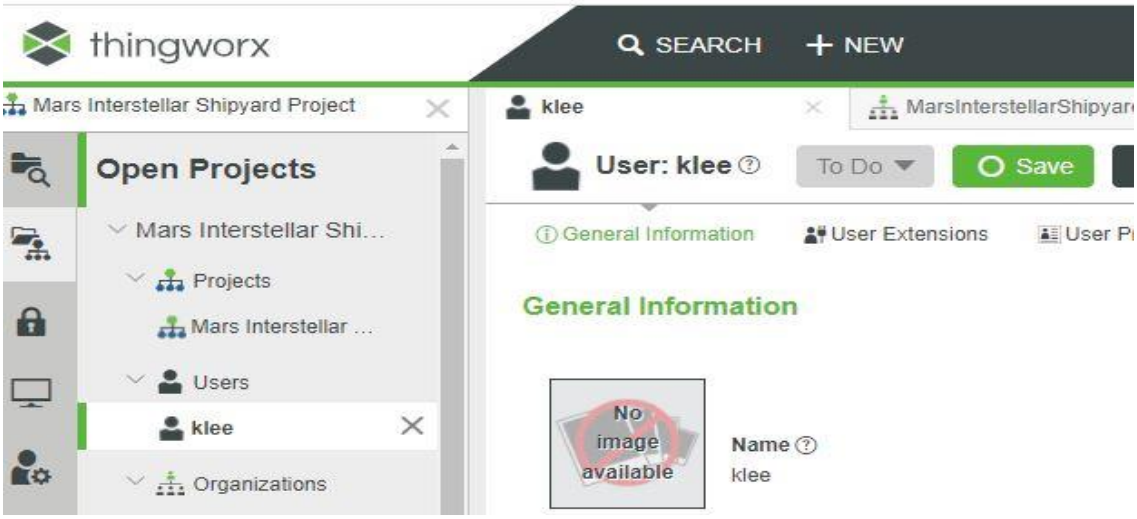
In this exercise, you will create the user entities for Karen the power administrator, and Hector the shipyard administrator.

Objectives: After successfully completing this exercise, you will be able to:

- Create a user entity.

Task 1: Create a user entity for Karen Lee, our power administrator

1. Create a new User entity.
2. In the Name field, type Klee.



3. In the New Password field, type Thingworx@12345.
4. In the Confirm Password field, type Thingworx@12345.
5. Click Save

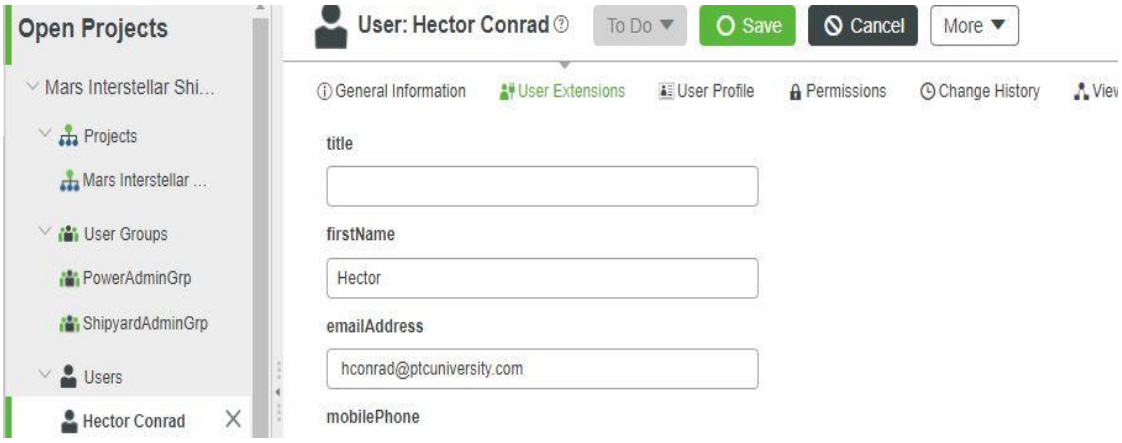
6. From the entities tabs, select User Extensions.

7. In the Last Name field, type Lee.
8. In the First Name field, type Karen.
9. In the E-mail Address field, type klee@ptcuniversity.com.
10. Click Save.

Task 2: Create a user entity for Hector Conrad, our shipyard administrator

11. Create a user entity with the following details:

User Name	Project	Password	First Name	Last Name	E-mail Address
hconrad	MarsInterstellarShipyardProject	ptcuniversity	Hector	Conrad	hconrad@ptcuniversity.com



Exercise 4: Create User Groups

In this exercise, you will create logical groups for the shipyard power administrators.

Objectives: After successfully completing this exercise, you will be able to:

- **Create a user group.**

Task 1: Create user groups

1. Create new user group entities with the following details:

Name: Project: Power Admin Grp Mars Interstellar Shipyard

: Shipyard AdminGrp MarsInsterstallerShipyardProject

Exercise 5: Add Users to User Groups

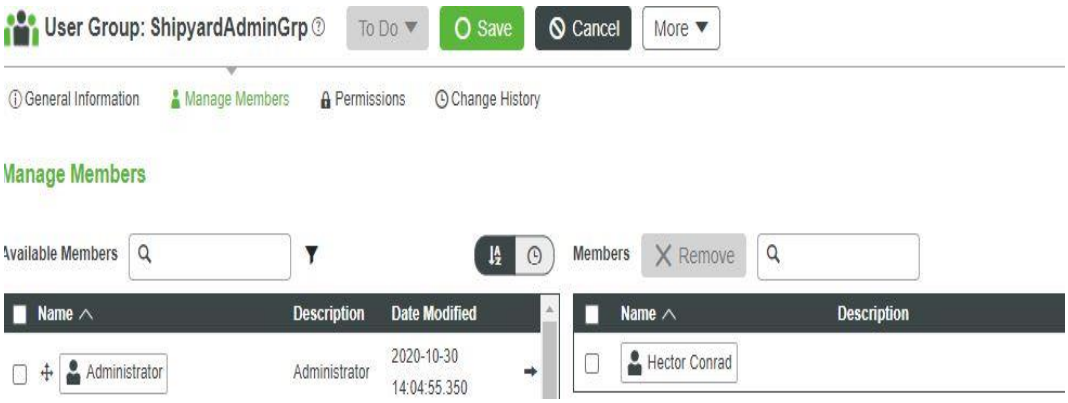
In this exercise, you will set membership for your logical groups, making Karen a member of the power group and Hector a member of the Shipyard group.

Objectives: After successfully completing this exercise, you will be able to:

- Add users to groups.

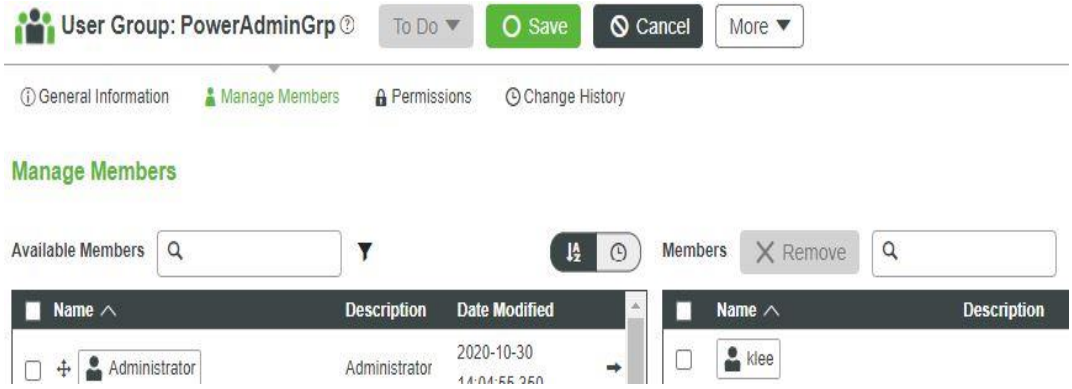
Task 1: Add Hector to the Shipyard group

1. Click Manage Members for the Shipyard AdminGrp user group.
2. Type h in the Type to filter list.... field in the left panel.
3. Drag h Conrad from the left panel to the right panel.
4. Click Save.



Task 2: Add Karen to the Power group

5. Open PowerAdminGrp from the Recent tab.
6. Click Manage Members for the PowerAdminGrp user group.
7. Drag Klee from the left panel to the right panel.
8. Click Save.



Exercise 6: Create Role Based Groups

In this exercise, you will create three role-based groups: One for power administrators, one for users with read-only access to power assets, and one for users with write access. As a best practice, logical groups are members of role-based groups. You will make the power administrators logical group a member of the Power. Admin role-based group, which indirectly makes Karen a member.

Objectives:

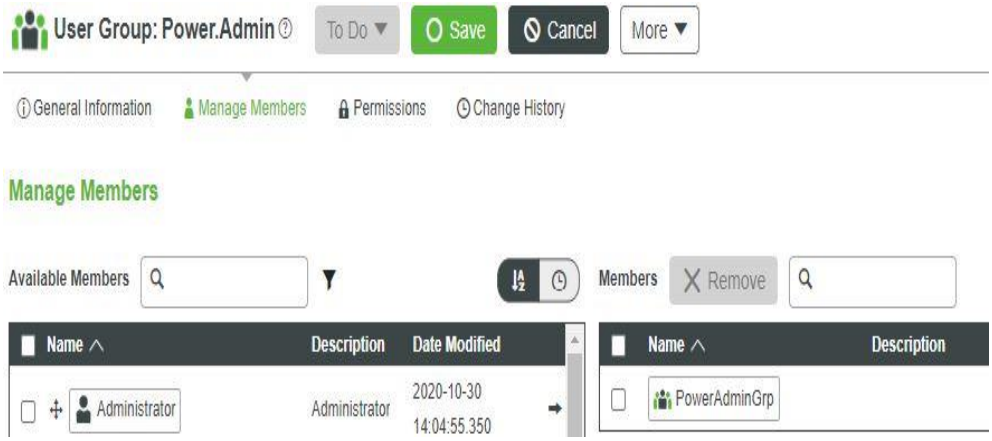
After successfully completing this exercise, you will be able to:

- Create role-based groups.

Task 1: Create role-based groups

1. Create user groups with the following details:

Name	Project	Members
Power.Admin	MarsInterstellarShipyardProject	PowerAdminGrp
Power.Write	MarsInterstellarShipyardProject	Power.Admin
Power.General	MarsInterstellarShipyardProject	Power.Write



Exercise 7: Add Role-Based Groups to Organizations

In this exercise, you will set membership for the Mars Interstellar Shipyard organization you created earlier.

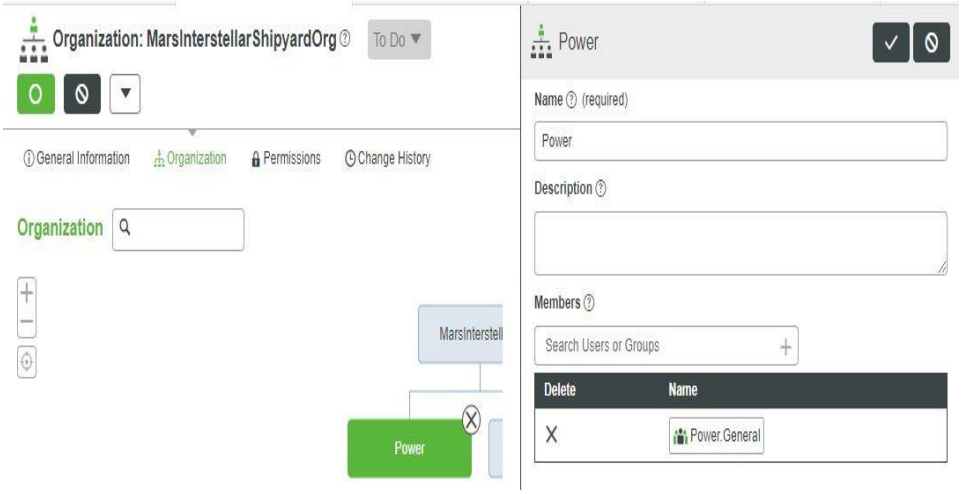
Objectives: After successfully completing this exercise, you will be able to:

- Add members to organizations and organizational units.

Task 1: Make the Power. General role-based group a member of the Power organizational unit

1. Navigate to the Organization page of the MarsInterstellarShipyardOrg from the Recent tab.

2. Click the Power box in the Organization chart.



3. In the Members field, type/select Power General.

4. Click Done.

What is Data Source?

A data source is the location where data that is being used originates from.

A data source may be the initial location where data is born or where physical information is first digitized, however even the most refined data may serve as a source, as long as another process accesses and utilizes it. Concretely, a data source may be a database, a flat file, live measurements from physical devices, scraped web data, or any of the myriad static and streaming data services which abound across the internet.

Types of Industrial IOT data sources

1.Industrial Control Systems

2.Bussiness Applications

3.Wearables

4.Sensors and Devices

5.Open and Web data

6.Media

7.Location

1. Industrial Control Systems

IoT makes it possible to leverage the data you already have in your SCADA system or historian.

A lot of companies we talk to have been gathering data in these systems for almost 30 years. But this data could only be used retrospectively until now.

By applying a machine learning algorithm to your SCADA data, you can predict a pump failure. Or determine the remaining useful life of a turbine engine.

If you already have a large volume of machine log data, machine learning will help you put that data to good use.

But knowing about an imminent failure isn't enough. That's why our IoT Application Suite has a strong focus on driving real-time actions

2. Business Applications

Data silos are still very common in industrial organizations. And this leads to missed opportunities because the data is already there. The right people just don't have access to it when they need it.

Data from applications like your CRM, ERP or EAM can provide context that goes beyond what's wrong with a machine.

IoT-enabled field service can dramatically improve customer experience. Giving technicians access to CRM data from their tablet shows them a detailed customer history. And they won't have to call the office to answer the customer's questions.

You can also build upon predictive maintenance with business data. When the machine learning algorithm predicts an asset failure you connect to your EAM system and check the warranty.

If the EAM data shows that the asset is still under warranty, you don't send a maintenance crew. Instead, you can have it kick off a task for someone to call out the manufacturer to fix the problem.

By automatically checking the warranty, you can prevent compromising warranties and reduce maintenance costs.

3. Wearables

Data from smart watches and fitness trackers aren't as useful as machine data for IIoT. But there is a new breed of industrial wearables making a name for itself.

These new wearables promise to make difficult and often dangerous jobs safer and easier.

Data from wearable gas detection sensors can track employee exposure levels. That data can then be displayed alongside their work schedule. This helps dispatchers adjust the schedule based on the worker's exposure. And ultimately it leads to fewer health issues.

Another wearable that's gaining popularity with large mines and constructions companies is the Smart Cap. The Smart Cap was created to prevent accidents. It measures truck driver fatigue levels by monitoring their brain activity.

When it picks up driver fatigue, an alarm will trigger to stop the driver and also let their manager know of the event. The possibilities to use this data go even further than just sounding alarms.

4. Sensors & Devices

Advances in sensor technology have made streaming real-time data easier than ever. Temperature, flow, pressure and humidity sensors have become big sources of industrial IoT data.

Sensors like this one from Labellum simplify remote water quality monitoring. Process industries produce waste water that could contaminate drinking water if procedures aren't followed. Contamination does damage to more than the environment. It often results in a PR disaster for the company responsible.

By monitoring water quality, you can respond to contamination faster than ever before.

5. Open & Web Data

What's the most common example of using open and web data? It's usually how to improve customer service by using social media posts. You employ a sentiment analysis algorithm and respond to negative posts quicker.

But in this post, we're going to cover an industrial story that builds on the water contamination example.

Here's how you can use web data to prevent waste water in effluent dams from overflowing and killing cows on the farm next door.

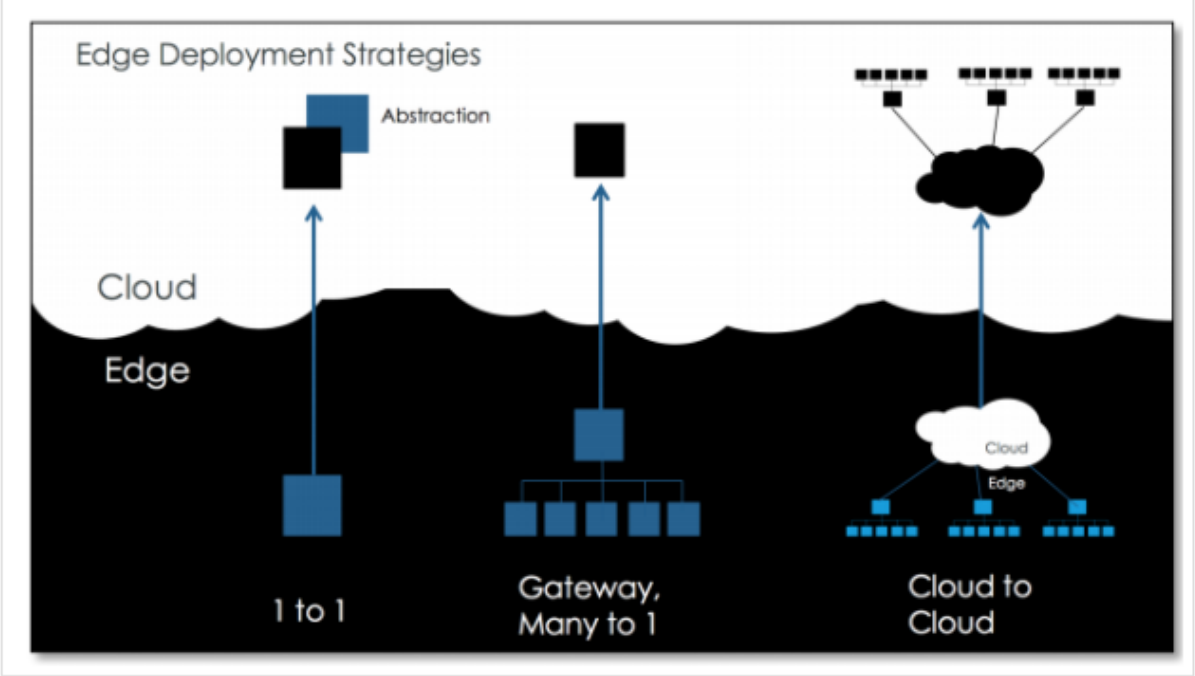
Using online weather services, you can predict when effluent dams are likely to overflow. Which means they are likely to contaminate water in the surrounding area.

Combine that with map data and you can also predict which specific reservoirs are in danger. This means you can take pre-emptive action and prevent the contamination from happening.

Open data sources aren't limited to weather, traffic and maps. You can also use open data from places like the NYC Open Data project. To see this in action check out our NYC Vermination cartoon.

Edge Deployment Strategy – Where is the Data?

The goal in this stage is to lay out all of the sources of data on the edge that feed into the model. While this is mostly a brainstorming and planning process, the insights and answers we arrive at will tee us up nicely for the next stage of the modelling process.



Edge Deployment Strategies

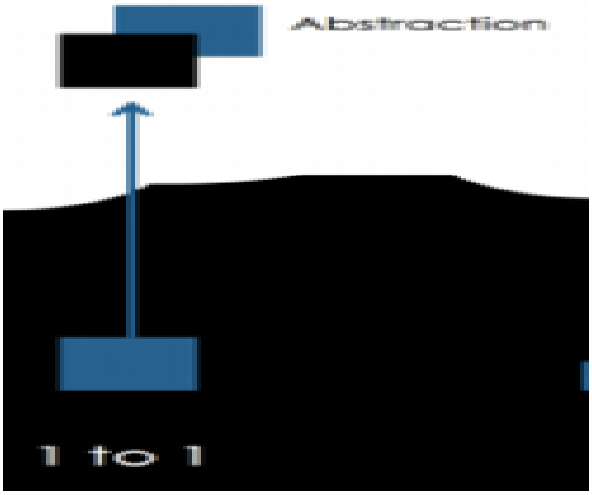
1. One to One Deployment Strategy
2. Gateway many to one deployment strategy
3. Cloud to Cloud Deployment Strategy
4. Gateway many to Many Deployment Strategy

Edge Computing:

Edge computing is a networking philosophy focused on bringing computing as close to the source of data as possible in order to reduce latency and bandwidth

use. In simpler terms, edge computing means running fewer processes in the cloud and moving those processes to local places, such as on a user's computer, an IoT device, or an edge server. Bringing computation to the network's edge minimizes the amount of long-distance communication that has to happen between a client and server.

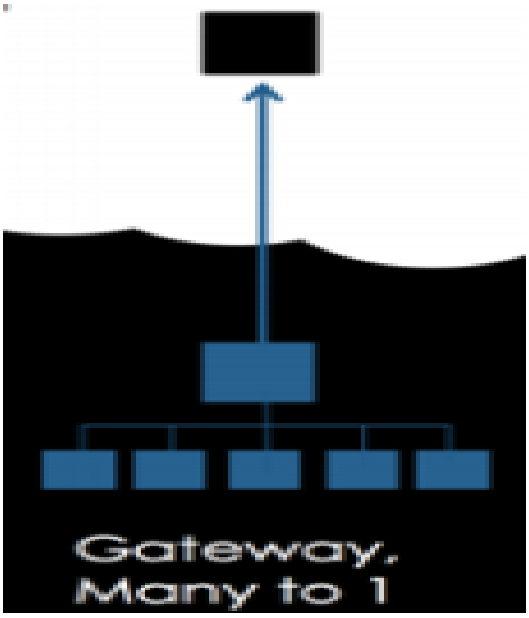
One to One Deployment Strategy



1 to 1: This deployment strategy is one of the simplest strategies. It is one edge device/thing talking to one digital representation in the cloud. This is also known as a digital twin.

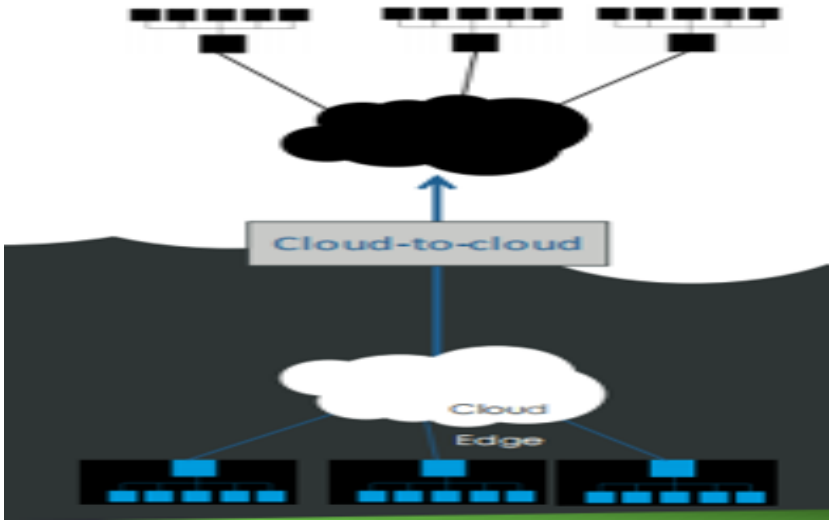
Abstraction: An abstraction edge deployment strategy is where we have *a* virtual thing that only exists in the cloud. This is a common strategy for rolling up data from various things into one virtual thing. Not every "thing" that exists in Thing Worx must be connected directly to something in the physical world, and that is why it is important to call out an abstraction deployment strategy.

Gateway many to one deployment strategy:



Gateway, many to 1: A gateway many to one deployment strategy is where we have some edge device communicating locally to things connected to it, and we have an **agent** on the gateway communicating on behalf of all of those locally connected things to the cloud.

Cloud to Cloud Deployment Strategy:



CLOUD:

Cloud storage is a model of computer data storage in which the digital data is stored in logical pools, said to be on "**the cloud**". The physical storage spans multiple servers (sometimes in multiple locations), and the physical environment is typically owned and managed by a hosting company. These cloud storage providers are responsible for keeping the data available and accessible, and the physical environment secured, protected, and running. People and organizations buy or lease storage capacity from the providers to store user, organization, or application data.

Cloud storage services may be accessed through a collocated cloud computing service, a web service application programming interface (API) or by applications that use the API, such as cloud desktop storage, a cloud storage gateway or Web-based content management systems.

Types of Cloud Platforms:

There are several types of cloud platforms. Not a single one works for everyone. There are several models, types, and services available to help meet the varying needs of users. They include:

- **Public Cloud**: Public cloud platforms are third-party providers that deliver computing resources over the Internet. Examples include Amazon Web Services (AWS), Google Cloud Platform, Alibaba, Microsoft Azure, and IBM Blue mix.

- **Private Cloud**: A private cloud platform is exclusive to a single organization. It's usually in an on-site data centre or hosted by a third-party service provider.

- **Hybrid Cloud**: This is a combination of public and private cloud platforms. Data and applications move seamlessly between the two. This gives the organization greater flexibility and helps optimize infrastructure, security, and compliance.

A cloud platform allows organizations to create cloud-native applications, test and build applications, and store, back up, and recover data. It also allows organizations to analyse data. Organizations can also stream video and audio, embed intelligence into their operations, and deliver software on-demand on a global scale.

Cloud to Cloud: This is a common edge deployment strategy where the edge in this case is another cloud. Examples of other clouds could be a **device cloud**, where we have some **other software/hardware vendor's cloud** receiving data and forwarding the data to Thing Worx. Another common example is a CRM system, where we need to collect service/repair case data and send it to Thing Worx. We can model this in Thing Worx, a single Thing representing the entire cloud, or individual Things, representing the individual devices connected to the other cloud.

Cloud providers:

Here is a list of cloud service providers:

1. Amazon Web Services (AWS)
2. Microsoft Azure
3. Google Cloud
4. Alibaba Cloud
5. IBM Cloud
6. Oracle
7. Salesforce
8. SAP
9. Rack space Cloud
10. VMWare

We will discuss about some of the cloud services

1. Amazon Web Services (AWS)

Amazon Web Services (AWS) is an Amazon company that was launched in the year 2002. AWS is the most popular cloud service provider in the world.

Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud platform, offering over 165 fully-featured services from data centres globally. This service is used by millions of customers.

AWS's revenue in the year 2018 was $25.6 billion with a profit of $7.2 billion. The revenue is expected to grow to $33 billion in 2019.

AWS Services

AWS offers hundreds of services. Some of these include Virtual Private Cloud, EC2, AWS Data Transfer, Simple Storage Service, Dynamo DB, Elastic Compute Cloud, AWS Key Management Service, Amazon CloudWatch, Simple Notification Service, Relational Database Service, Route 53, Simple Queue Service, Cloud Trail, and Simple Email Service

AWS Security

Cloud security is the highest priority for AWS. As a customer, you will benefit from a data centre and network architecture built to meet the requirements of the most security-sensitive organizations.

2. Microsoft Azure

Microsoft Azure is one of the fastest-growing clouds among them all. Azure was launched years after the release of AWS and Google Cloud but is still knocking on the door to become the top cloud services provider.

While Microsoft Azure revenue is difficult to predict, Microsoft broke down its revenue of the last quarter into three categories, Productivity and Business Processes, Intelligent Cloud, and Personal Computing. The respective revenue was $11.0 billion, $11.4 billion, and $11.3 billion.

Microsoft's Azure revenue is expected to grow between $33 billion to $35 billion. This makes Azure one of the most profitable cloud services in the world.

Azure Services

Azure offers hundreds of services within various categories including AI + Machine Learning, Analytics, Block chain, Compute, Containers, Databases, Developer Tools, DevOps, Identity, Integration, Internet of Things, Management, Media, Microsoft Azure Stack, Migration, Mixed Reality, Mobile, Networking, Security, Storage, Web, and Windows Virtual Desktop.

1.IBM Cloud

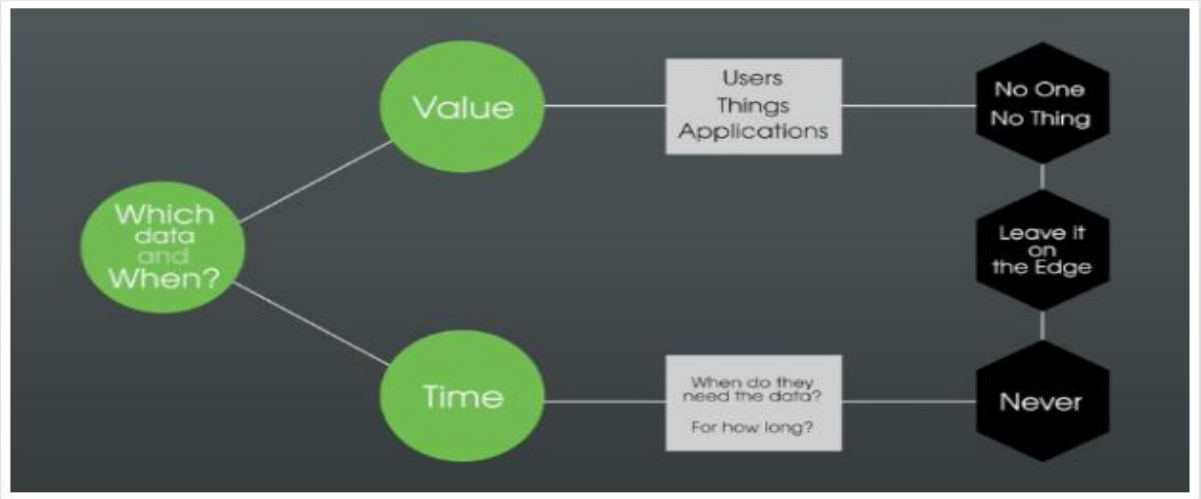
IBM Cloud developed by IBM is a set of cloud computing services for businesses. Similar to other cloud service providers, the IBM cloud includes IaaS, SaaS, and PaaS services via public, private, and hybrid cloud models.

Compute, Network, Storage, Cloud Packs, Management, Security, Database, Analytics, AI, IoT, Mobile, Dev Tools, Block chain, Integration, Migration, Private Cloud, and VMware.

2.Google Cloud

Google cloud platform is Google's cloud. Similar to AWS and Azure, Google Cloud also offers similar services in various categories, including compute, storage, identity, security, database, AI and machine learning, virtualization, DevOps and more.

3. **Data Strategy:**



Data in IOT grows exponentially, it is here we have to apply NO strategy. If No one or Nothing ever needs data, then its best we leave that data on Edge. Unneeded data can get out of control.

Data Structures – Primitive (Name, Base Type, **Value**, **Time**, Quality) and Advanced (Entities- Type of Objects-specialized to contain structured data). Data shape defines schema of data structures and contains name and base type. Output of services is result variable. If there are multiple data points in output, then we need to use info table with data shape. Persistent properties are stored in TW DB, non-persistent are stored in Memory.

Non persistent properties are more resourceful and efficient as last changes values are not stored in database. This is particularly useful when we are interested in reacting to data points. Data will be lost on resetting a Thing.

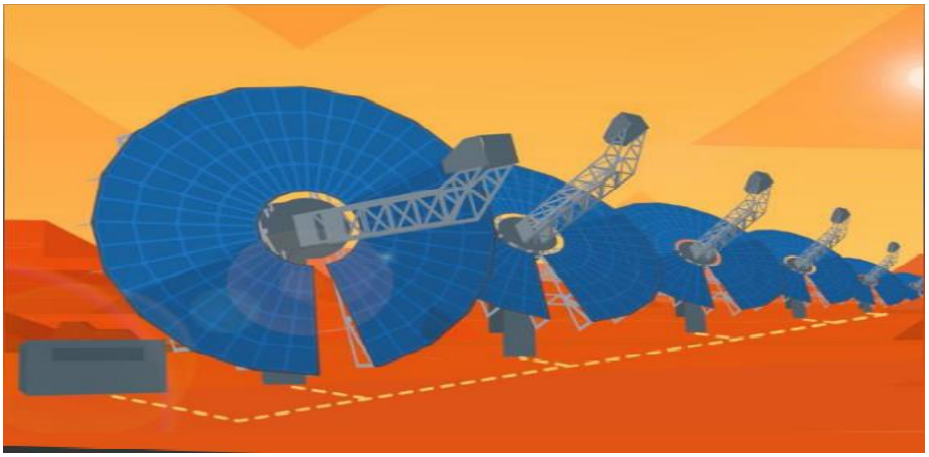
Thing will get reset when Thing template or Data-shape on which thing is dependent is changed or TW Platform is restarted. Everything will be set to default value.

in short if property is read from device on demand, then there is no need of persistence. However, if property is generated on Thing Worx server then it should be persistent so it is not lost on thing reset.

value streams – for logging properties. Properties can be logged by checking Logged option but that thing should have value stream attached to it to store those logs. Every value stream is associated with Persistent provider entity, which specifies which database stores steam. user changing property must have visibility to persistent provider or an error will occur when setting the property. Events/Alerts can be generated from properties of custom JavaScript services.

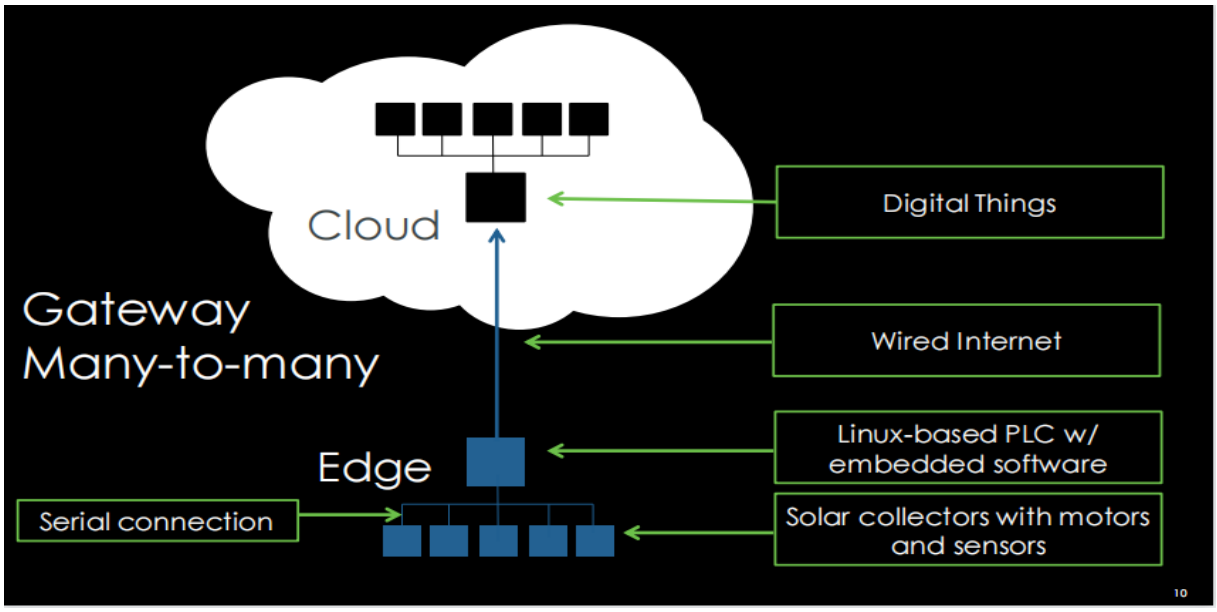
Gateway many to Many Deployment Strategy:

SHIPYARD POWER SYSTEM – POWER GENERATORS



Solar Collectors:

- Generate all shipyard power.
- Power diversification planned for the future.
- Many solar collectors connected to Linux based programmable logic controller (PLC).
- PLC is connected to the Internet.

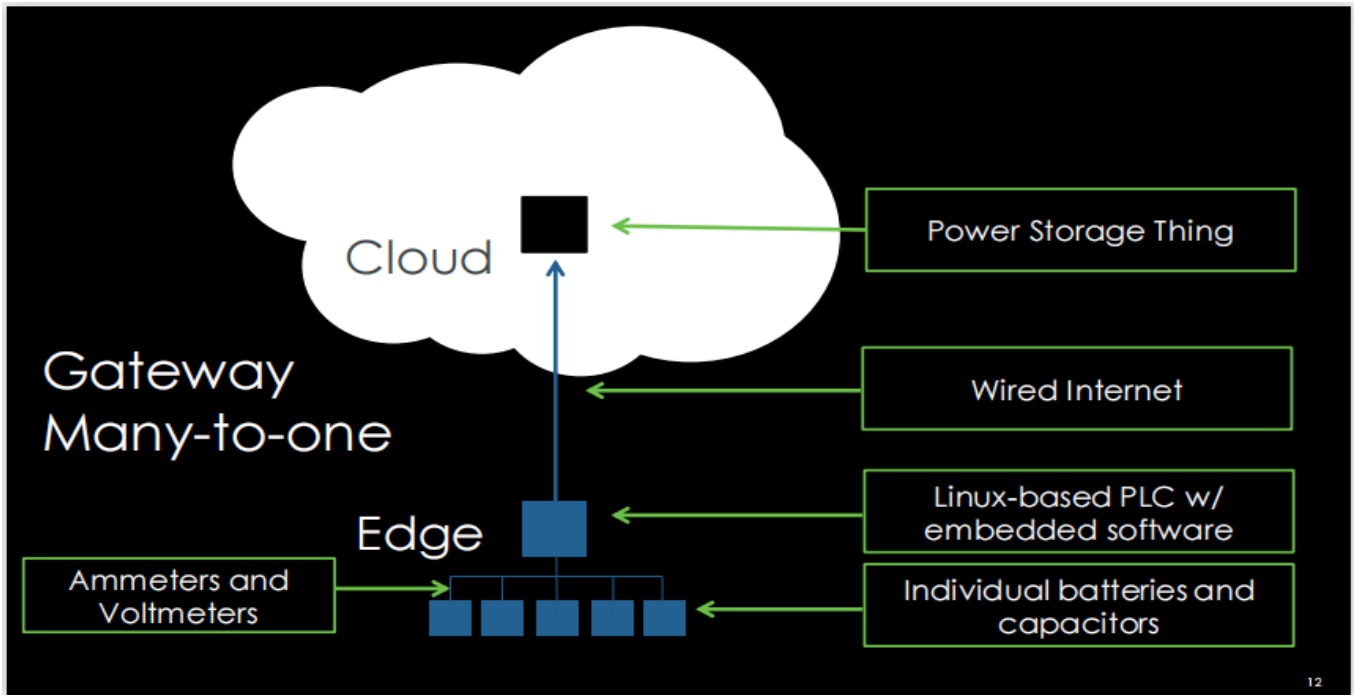


SHIPYARD POWER SYSTEM – POWER STORAGE

Power Storage:

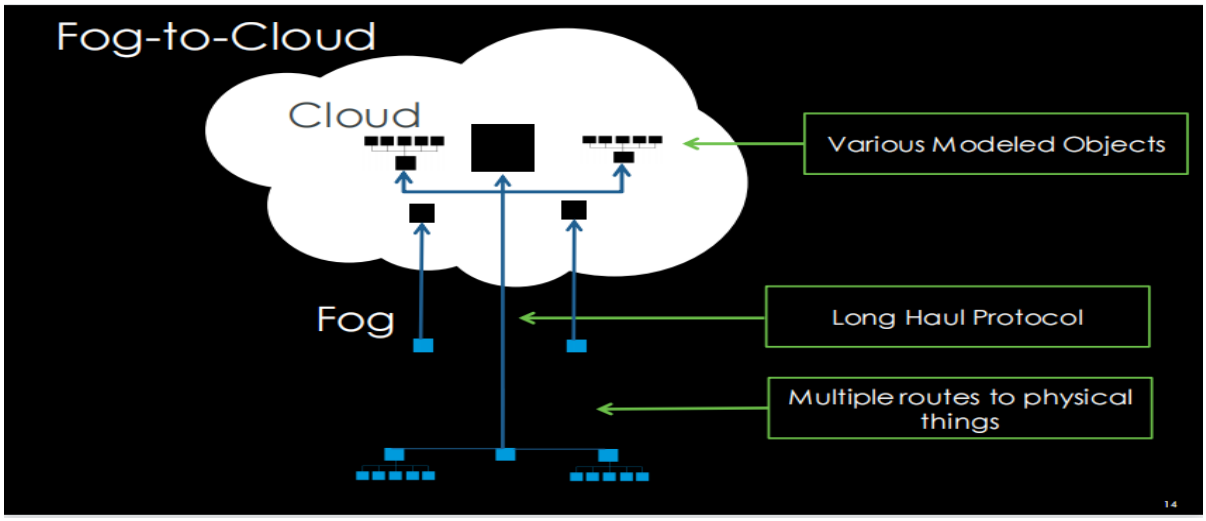
- Under Ground Bunkers with dumb Batteries.
- Power Lines Connecting Batteries.
- Ammeters/Voltmeters.
- PLC Watching Power Sensors.

Gateway Many to One:



In above fig. Describes about Gateway Many to one strategy, it explains about shipyard power storage System. The power is stored in individual batteries and capacitors. which are inter connected. In that Ammeters and Voltmeters, it shows the values The entire system is connected with Linux based PLC/embedded software. And PLC Controller is connected to internet and data is passing to cloud, it having power storage thing.

Fog to Cloud:



Fog computing is a decentralized computing infrastructure in which data, compute, storage and applications are located somewhere between the data source and the cloud. Like edge computing, fog computing brings the advantages and power of the cloud closer to where data is created and acted upon. Many people use the terms *fog computing* and *edge computing* interchangeably because both involve bringing intelligence and processing closer to where the data is created. This is often done to improve efficiency, though it might also be done for security and compliance reasons.

The fog metaphor comes from the meteorological term for a cloud close to the ground, just as fog concentrates on the edge of the network. The term is often associated with Cisco; the company's product line manager, Ginny Nichols, is believed to have coined the term. Cisco Fog Computing is a registered name; fog computing is open to the community at large.
How does fog computing work?

Fog networking complements -- doesn't replace -- cloud computing; fogging enables short-term analytics at the edge, while the cloud performs resource-intensive, longer-term analytics. Although edge devices and sensors are where data is generated and collected, they sometimes don't have the compute and storage resources to perform advanced analytics and machine

learning tasks. Though cloud servers have the power to do this, they are often too far away to process the data and respond in a timely manner.

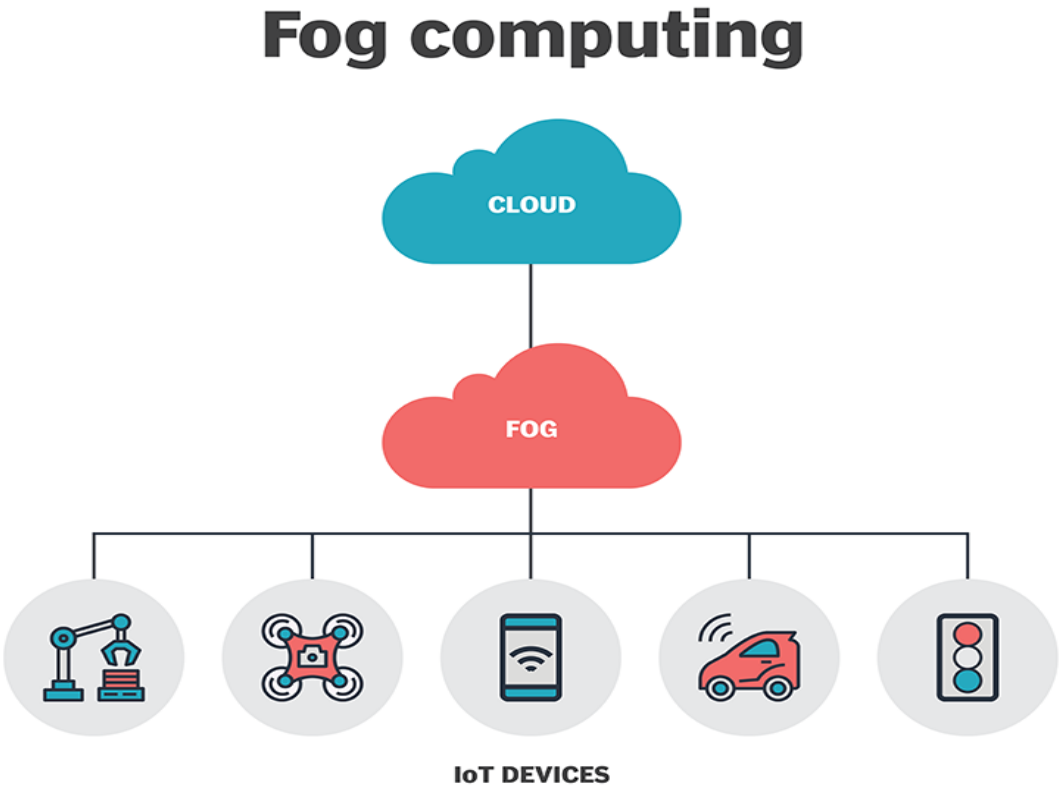
In addition, having all endpoints connecting to and sending raw data to the cloud over the internet can have privacy, security and legal implications, especially when dealing with sensitive data subject to regulations in different countries. Popular fog computing applications include smart grids, smart cities, smart buildings, vehicle networks and software-defined networks.

How and why is fog computing used?

There are any number of potential use cases for fog computing. One increasingly common use case for fog computing is traffic control. Because sensors -- such as those used to detect traffic -- are often connected to cellular networks, cities sometimes deploy computing resources near the cell tower. These computing capabilities enable real-time analytics of traffic data, thereby enabling traffic signals to respond in real time to changing conditions.

This basic concept is also being extended to autonomous vehicles. Autonomous vehicles essentially function as edge devices because of their vast onboard computing power. These vehicles must be able to ingest data from a huge number of sensors, perform real-time data analytics and then respond accordingly.

Because an autonomous vehicle is designed to function without the need for cloud connectivity, it's tempting to think of autonomous vehicles as not being connected devices. Even though an autonomous vehicle must be able to drive safely in the total absence of cloud connectivity, it's still possible to use connectivity when available. Some cities are considering how an autonomous vehicle might operate with the same computing resources used to control traffic lights. Such a vehicle might, for example, function as an edge device and use its own computing capabilities to relay real-time data to the system that ingests traffic data from other sources. The underlying computing platform can then use this data to operate traffic signals more effectively.



Fog computing has emerged as a promising technology that can bring the cloud applications closer to the physical IoT devices at the network edge. While it is widely known what cloud computing is, and how data centres can build the cloud infrastructure and how applications can make use of this infrastructure, there is no common picture on what fog computing and a fog node, as its main building block, really is. One of the first attempts to define a fog node was made by Cisco, qualifying a fog computing system as a “mini-cloud,” located at the edge of the network and implemented through a variety of edge devices, interconnected by a variety, mostly wireless, communication technologies. Thus, a fog node would be the infrastructure implementing the said mini-cloud. Other proposals have their own definition of what a fog node is, usually in relation to a specific edge device, a specific use case or an application.

MODEL COMPOSITION

What is the Model Composition?

This is where we plan out in detail the various things we are going to model, and how to structure them. In many ways, Thing Worx Model objects can be compared to some basic object-oriented principles. The terms "extend" and "implement" can be used to describe the model.

Shape's properties and services when you implement it in Thing Worx. The composition can be described as a "has a" characteristic. For example, an Acme Tractor "has a" mower deck and "has a" motor.

You can think of a Thing Template as a class. In object-oriented design, a class can extend a base class. It is also referred to as inheritance, which means you can redefine or extend the established behaviour of a base class. When you extend a Thing Template in ThingWorx, you inherit its properties and services. Inheritance can be described as an "is a" characteristic.

For example, Economy Series Model 10 tractor:

•"is an" Economy Series tractor.

•"is a" Home and Garden tractor.

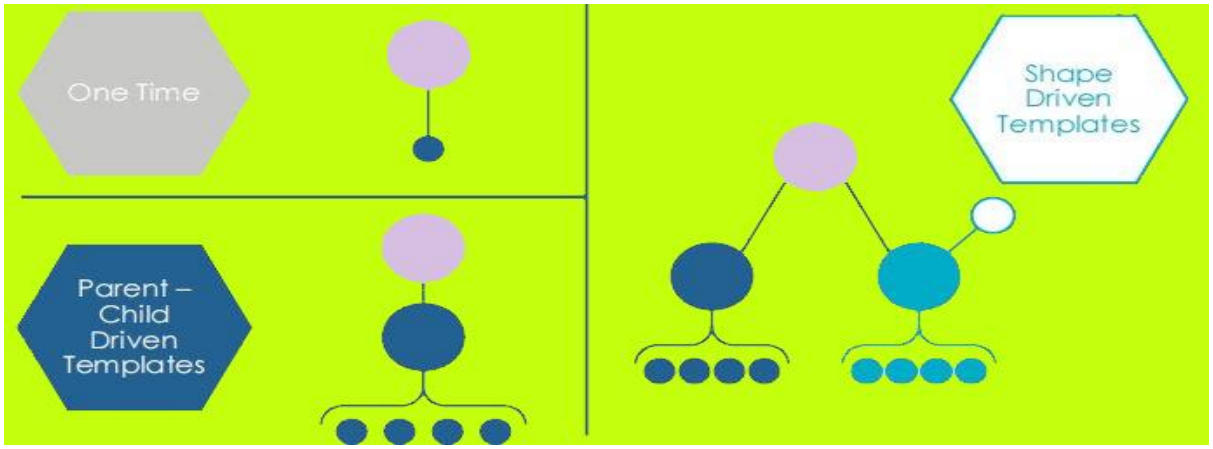
•"is an" Acme tractor.

Three Types of Modelling Patterns

The modelling pattern is a way of structuring things, shapes, and templates in your model.

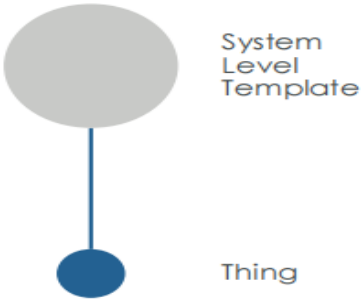
There is three basic modelling pattern:

- One Time
- Parent-Child Driven Templates
- Shape Driven Templates



We'll go over each of these, starting from the simplest to the most complex.

One Time Modelling Patterns



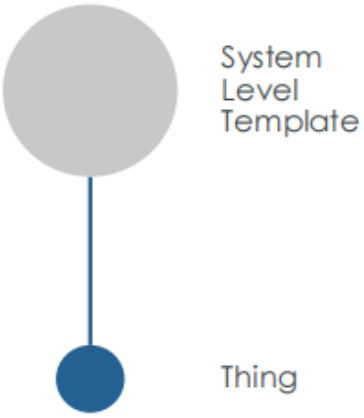
One Time **Modelling Patterns** Simplest modelling pattern because it's:

- One Time **Modelling Patterns** Only appropriate for a unique thing.
- System-level templates are derived from things.
- It is possible to add functionality directly to a thing in a One-time pattern.

These Patterns Not appropriate if:

- There is more than one of these things.
- There will ever be more than one of these things.
- This pattern has poor reusability

SYSTEM-LEVEL TEMPLATES



There is a system-level template that comes with ThingWorx or an imported ThingWorx extension.

The files are read-only and cannot be edited

You must select the right base template when you first create an entity since it cannot be changed once it has been created. If you decide a different base template is needed, you will have to delete the entity and re-implement

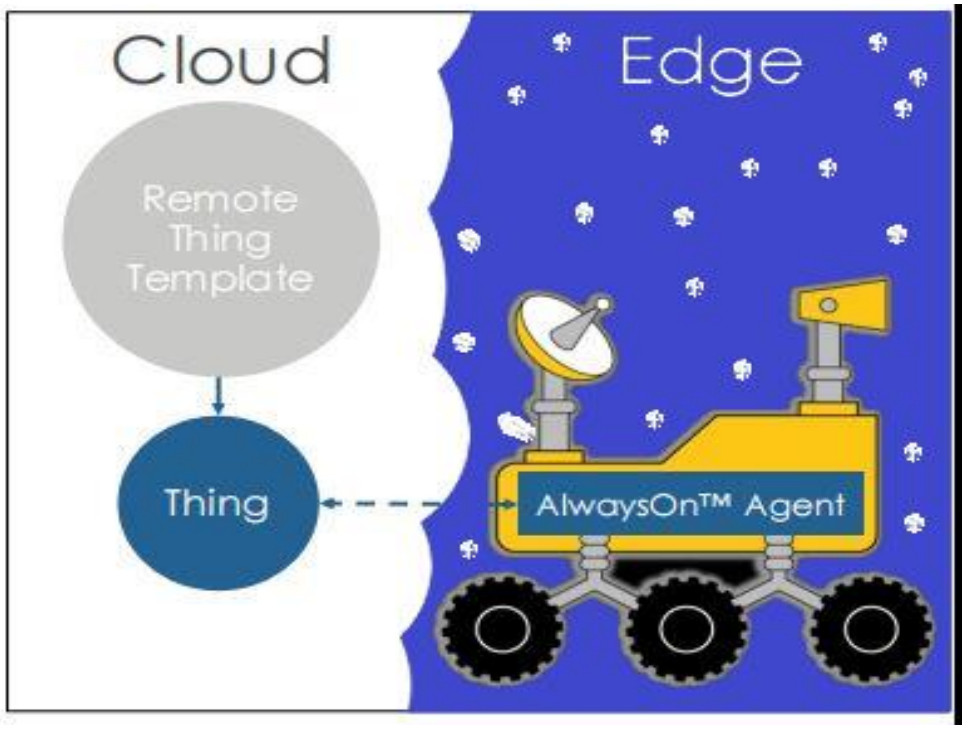
Major system-level templates are:

- Generic Thing
- Remote Thing
- File Repository

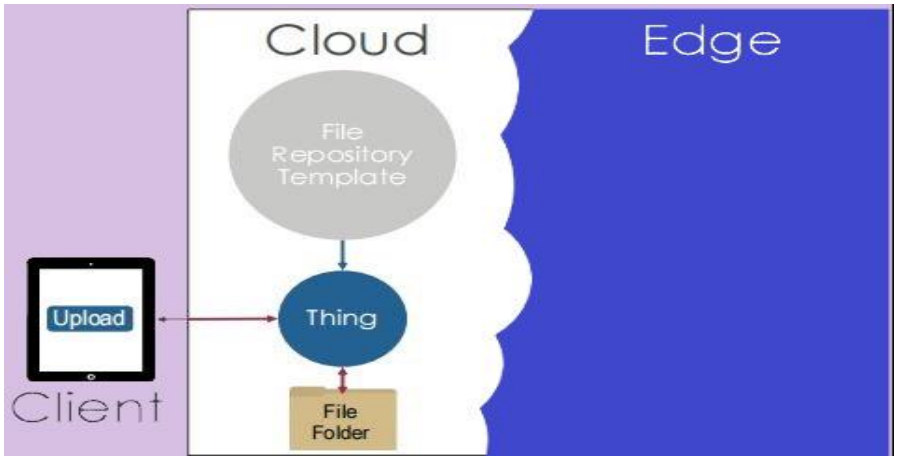
a. *Generic Thing*: You used the generic thing template for the lesson on fundamentals of IoT with ThingWorx. There are many thing templates available, but this is the most basic one. It provides minimal functionality, such as the ability to get things properties or disable things.

b. Remote Thing: A Remote Thing is an edge device or data source whose location is at a distance from the location of the Thing Worx Platform. A

Remote Thing accesses the platform, and can itself be accessed, over the network. For example, if you want to connect to an Always On™ agent on the edge, you need a Remote Thing template.

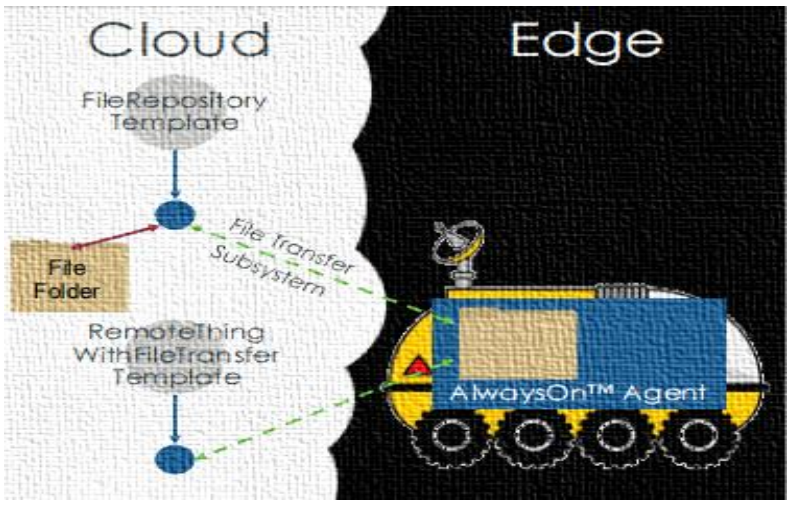


a. *File Repository*: It allows a user client to upload, download, and browse files on the ThingWorx server when it is derived from the File Repository thing template.

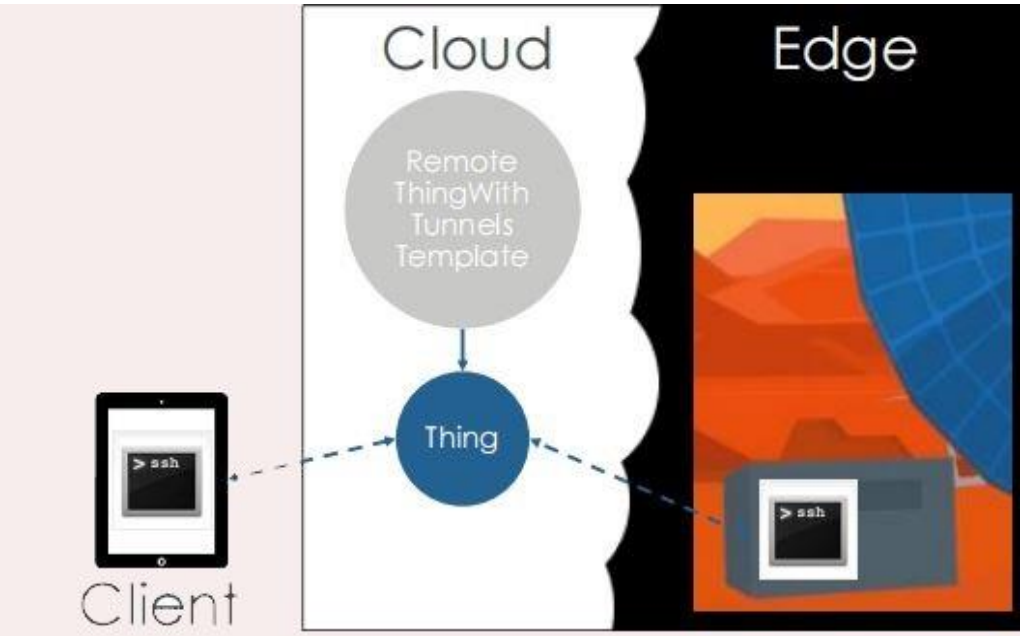


b. RemoteThingWithFileTransfe: A thing derived from the RemoteThingWithFileTransfe thing template creates a file repository on the edge instead of the ThingWorx server. This enables you to transfer files between the edge device and ThingWorx.

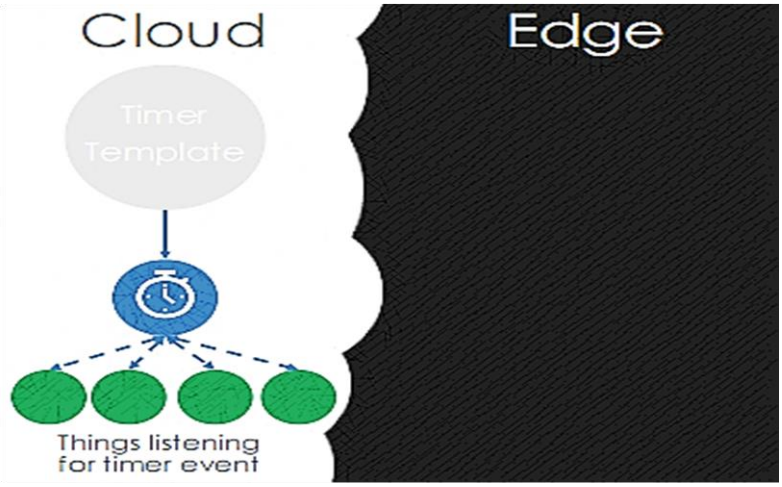
However, the client cannot transfer files with the edge directly using Always On™. The client has no direct connection to the edge device. Instead, the client transfers files to a ThingWorx file repository, then services in the file transfer subsystem transfer files between ThingWorx and the folder on the edge.



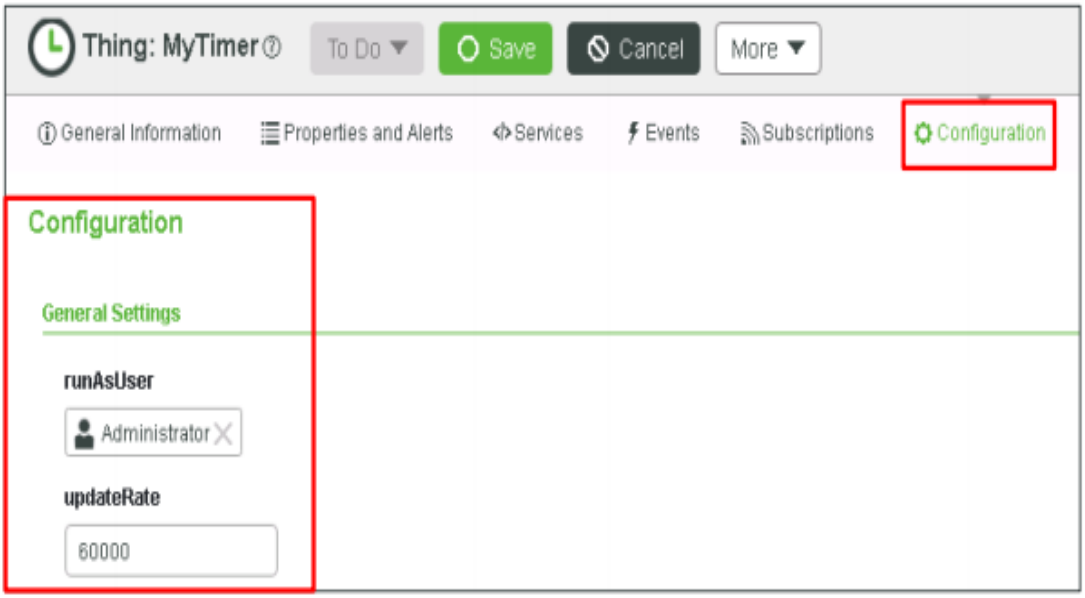
a. **RemoteThingWithTunnel**: A RemoteThing plus tunnelling enablement. A RemoteThingWithTunnel enables us to use ThingWorx to communicate with the edge device like we were on its la local network. In our example, the Linux-based gateway to our solar collectors is accessible via SSH, just like most Linux systems. However, the client can't reach the gateway directly because it's behind a NAT-based firewall.



b. *Timer*: A simple timer that fires an event on a defined interval. A timer thing emits a timer event periodically and is useful for tasks that need to run on a regular schedule.



Predefined Thing Templates/Configuration Tables



Things derived from advanced thing templates have configuration tables.

– Constants are required for advanced functionality.

ThingWorx Extensions

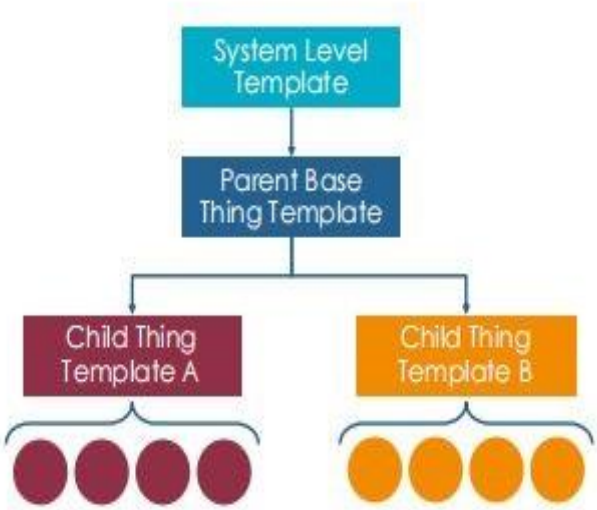
Extensions enable you to quickly and easily **add new functionality** to an IoT solution. Extensions can be service (function/method) libraries, connector templates, functional widgets, and more.

1. Thing Template and Shape Extensions:
   - Provide functionality beyond Always On™.
   - Examples: Database, Notification, Twilit.
2. Resource Extensions:
   - Provide platform services not associated with a thing.
3. Widget Extensions:
   - Add widgets to Mashup Builder.
   - May communicate with external systems.
   - Example: Google Maps.

PTC Marketplace – Pre-Built Extensions.

PTC Marketplace is a digital space where partners and customers can learn about exciting Industrial IoT apps and market-ready solutions, and promote innovative technology. PTC's Marketplace makes it easy for solution builders to find market-ready solutions and customized accelerators while providing a space for PTC partners to promote and showcase their innovative IoT and AR technologies, solutions, services, and industry expertise to customers and prospects looking to leverage them. The website where you can find the extensions.

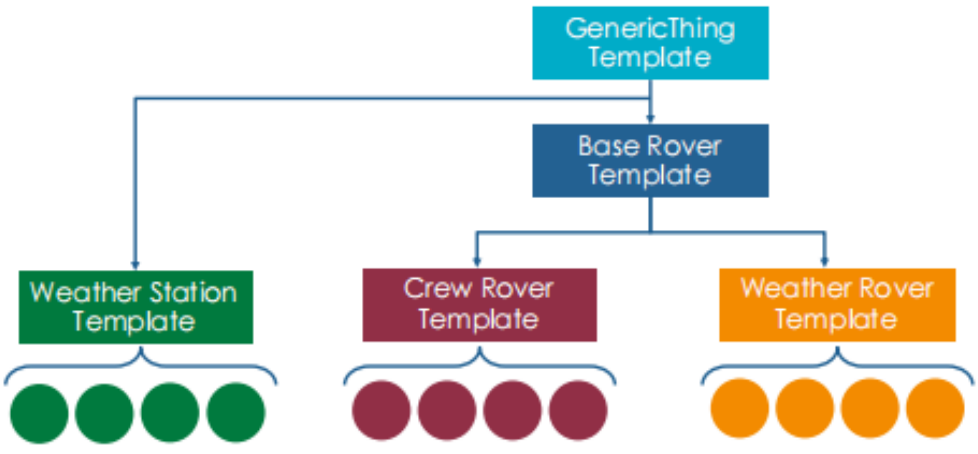
Parent-Child Modelling Pattern



Functionality in Thing Templates of **Parent-Child Modelling Pattern**:

- Functionality not placed directly in things.
- Enhances reusability
- Easy to create new things.

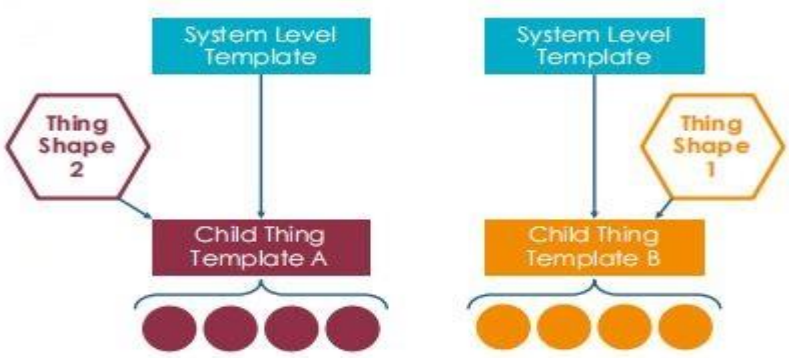
If you took the Fundamentals of IoT with ThingWorx, the parent/child modelling pattern should look very familiar.



Modular Shape-Driven Modeling Pattern

Thing Shapes are functionality modules.

- Contains:
  - Properties
  - Services
  - Events
  - Subscriptions
- Applied to Templates or Things.



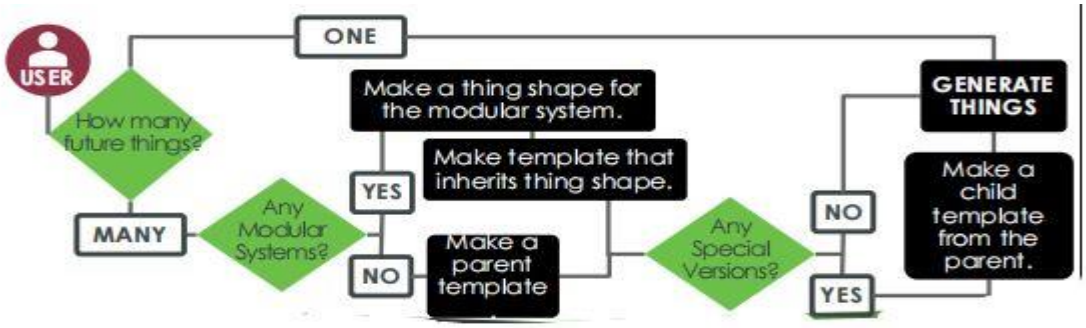
Consider again the project we did with ThingWorx as part of the basics of IoT. Can there be any modular capabilities that we can abstract out of many different things and use them in many different places?

Well, both our crew rovers and weather stations can have people in them, and thus require basic life support and climate control. That could be a thing shape that we apply to weather stations and crew rovers. Weather rovers are unmanned, so they would not need the life support functionality, and would not need the thing shape.

Similarly, both our weather station and weather rovers have sensors for detecting wind speed and other weather conditions. That functionality could be

abstracted out into a shape and applied to the Weather Rovers and Weather Stations, but the crew rovers don't have this feature.

Model Composition Decision Tree



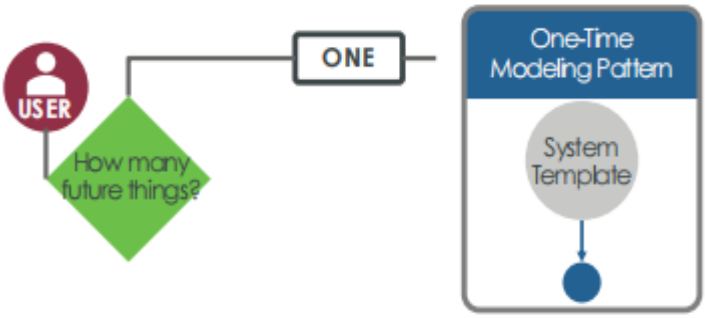
A decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on the most significant splitter/differentiator in input variables.

In the version composition selection tree you can find out what things, factor shapes, and aspect templates you need to construct your model.

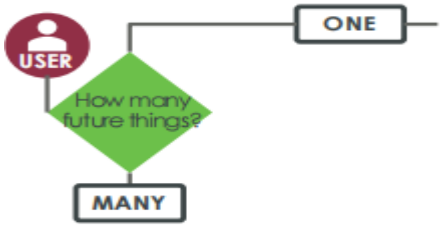
This decision tree helps you decide how to model your systems and things, which is a very critical stage in IoT application development.

- Our first question is **how many future things**?

- If there are only one of these things, and there will never be another, we should just create the thing and use the one-time modeling pattern

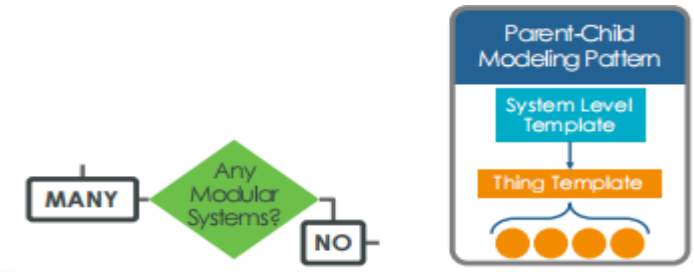


- If there will be more than one of these, we need to ask more questions.

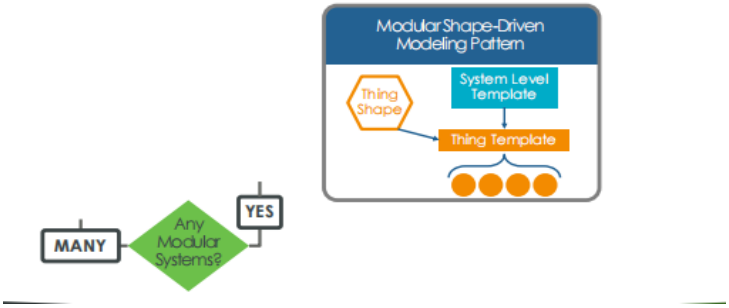


- Our next question is if there is any functionality in our thing that would be useful to other things?

- If not, we should use the parent-child modeling pattern, putting all our functionality in the thing template.

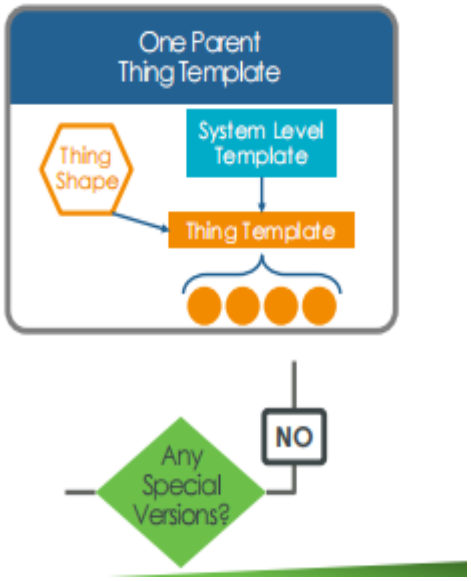


- If so, we should use the shape-driven modeling pattern, taking the modular functionality and putting it in a thing shape, and putting the non-modular functionality in a thing template.

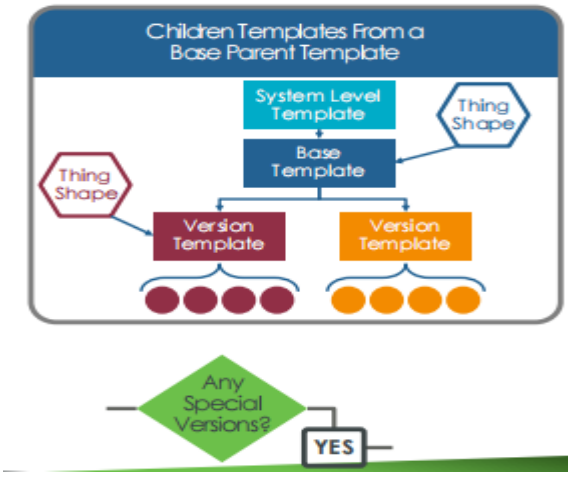


- Next, are there any special versions of this thing?

- If all the things are pretty similar, you have one parent thing template encapsulating all the functionality



- If there are different versions with different functionality, like our rovers, we need a base thing template with the functionality that is common to all versions, and version thing templates with functionality unique to that version.
- We can use thing shapes to either the base template or version templates.
- For example, a battery shape would apply to the base rover thing template, while a weather sensor shape would apply to the weather rover version only



Quiz - Power Storage Model Composition

How many future things?

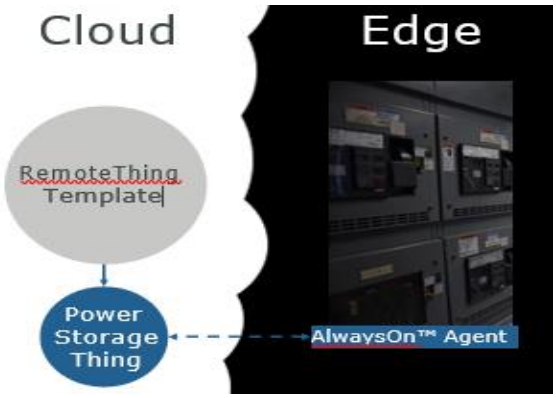
– One

What modeling pattern will we use?

– One-time modeling pattern

What system thing template will we use?

– Remote Thing



(Hint: The power storage facility will have an Always On™ agent.)

Quiz – Shipyard Model Composition

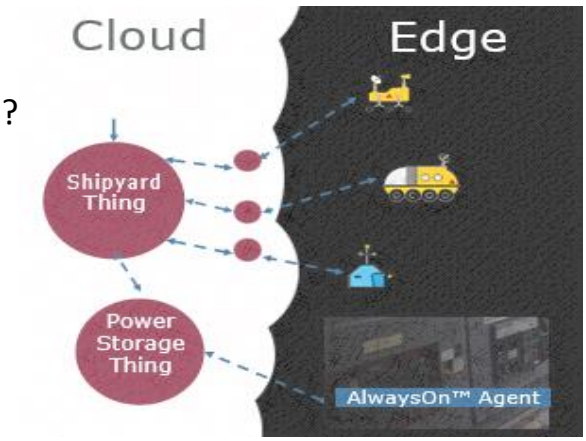
How many future things?

– One What modelling pattern will we use?

– One-time modelling pattern

What system thing template will we use?

- Timer Simple



(Hint: The power storage facility will have an Always On™ agent.)

Quiz – Solar Collector Model Composition

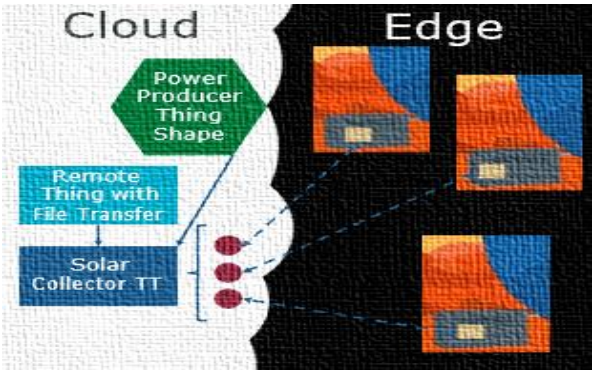
There are many solar collectors and they are all very similar, but there will be other types of power producers, such as wind and nuclear.

What modelling pattern will we use?

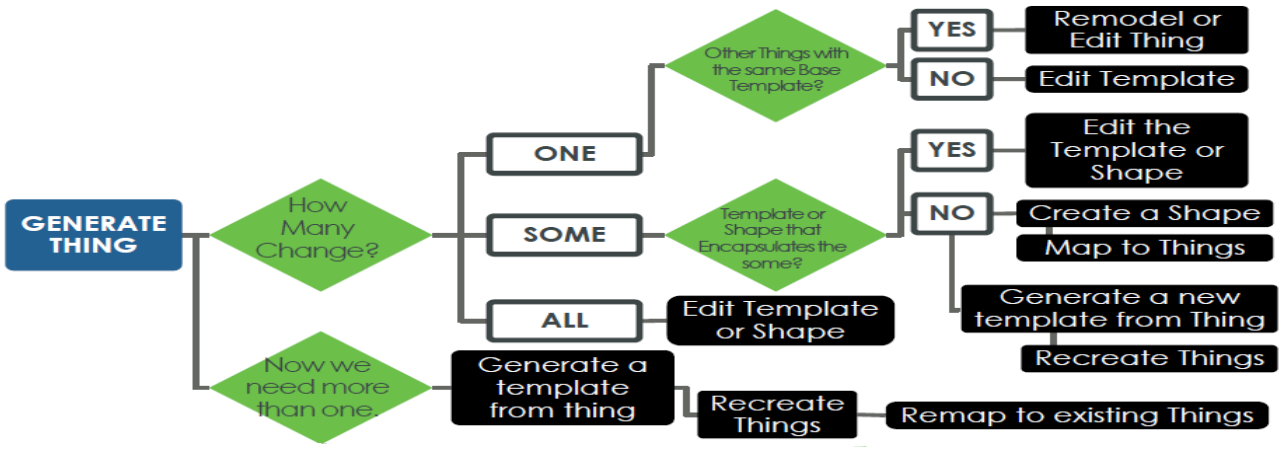
– Shape-driven

What system thing template will we use?

- Remote Thing with File Transfer



(Hint: The solar collector has an Always On™ agent and has a log file) Model Iteration Decision Tree

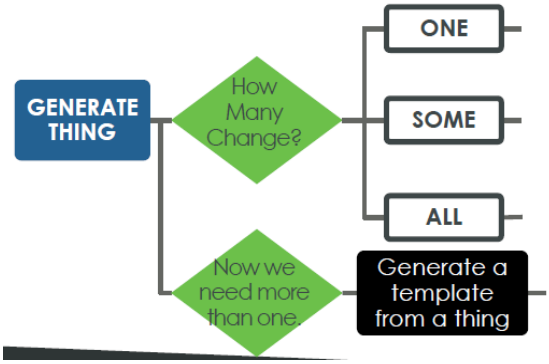


The compositional decision tree of the model is excellent when you build your model. But what if you already have a model and need to make modifications due to new versions or features?

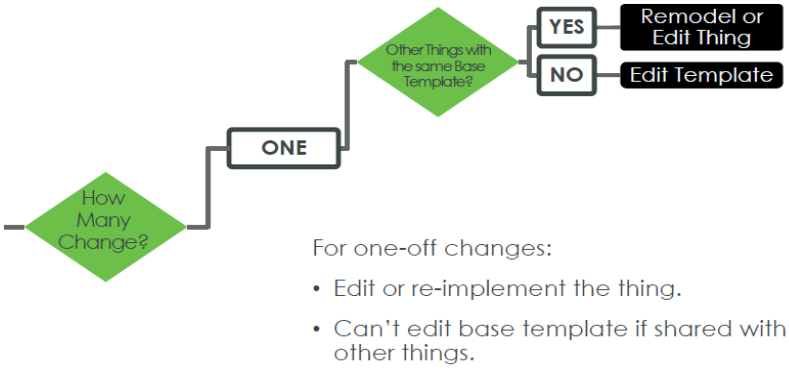
That's what the Model Iteration Decision Tree is for.

- So, our model iteration decision tree starts with things already generated.

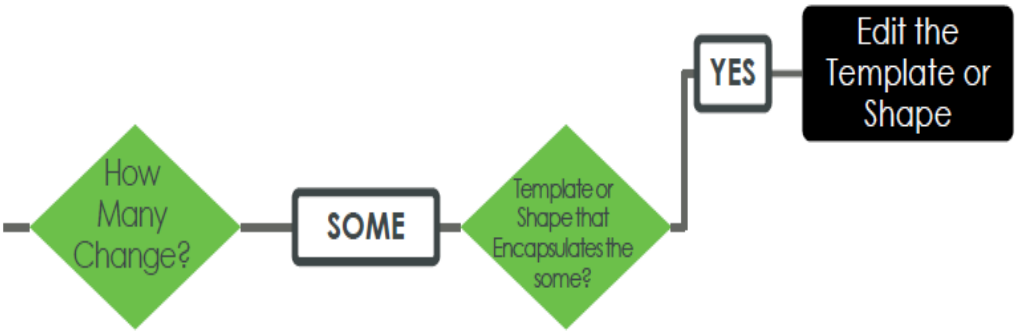
The first question is how many of those things change?



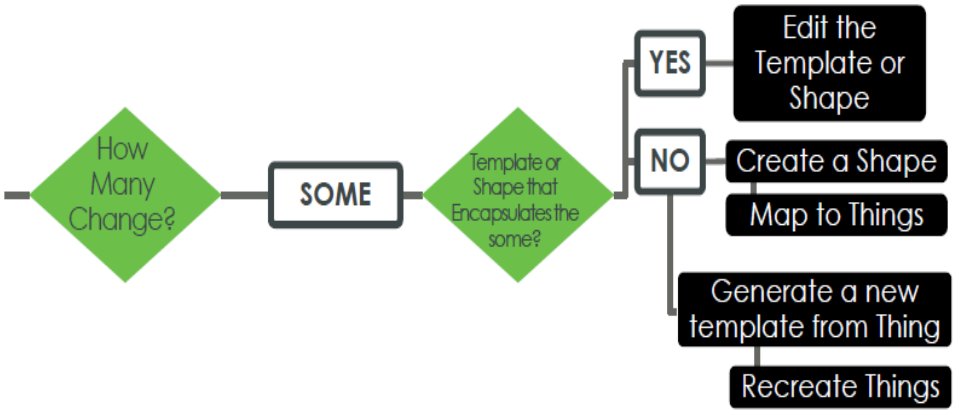
- Let's suppose you only need to change one thing and don't need to make others like it.
- If other things are using the same base thing template, you can't edit the thing template. Doing so would alter all of the other things derived from it.
- So, you will usually either edit the thing directly, or you might delete the thing and re-create it using the one-off modeling pattern.



- If some of the things are changing and there is already a thing shape or thing template that is being used by all the affected things, your job is easy, just edit the thing shape or thing template to add the functionality



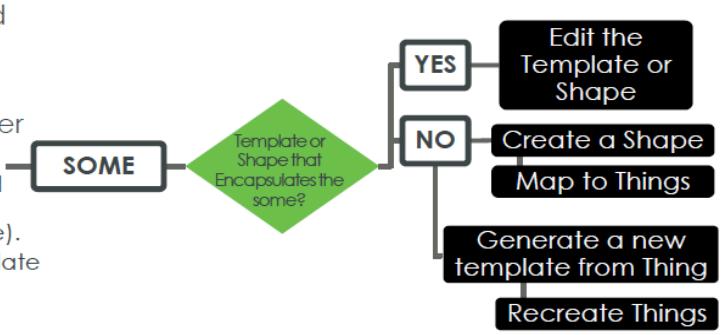
- But, if that isn't the case, the issue gets more complex.
- You could create a thing shape and apply it to all the affected things. However, that means you are applying the thing shape to things one at a time, which isn't a good practice. It is better to apply thing shapes to thing templates.
- Or, you could create a new thing template. But, you can't change the base thing template after creation, so you would have to delete all the affected things and re-create them using the new thing template.



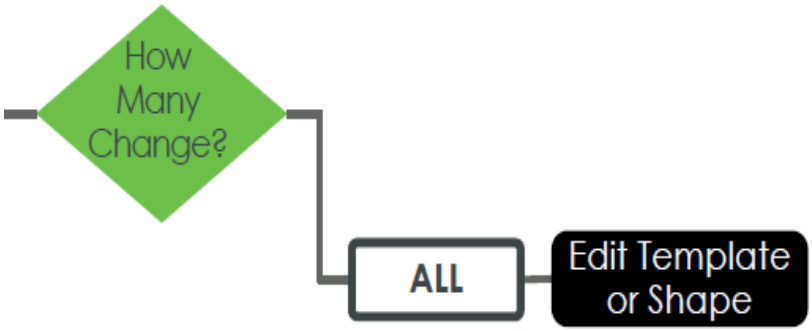
- But, if that isn't the case, the issue gets more complex.
- You could create a thing shape and apply it to all the affected things. However, that means you are applying the thing shape to things one at a time, which isn't a good practice. It is better to apply thing shapes to thing templates.
- Or, you could create a new thing template. But, you can't change the base thing template after creation, so you would have to delete all the affected things and re-create them using the new thing template.

If some things derived from the template/shape change:

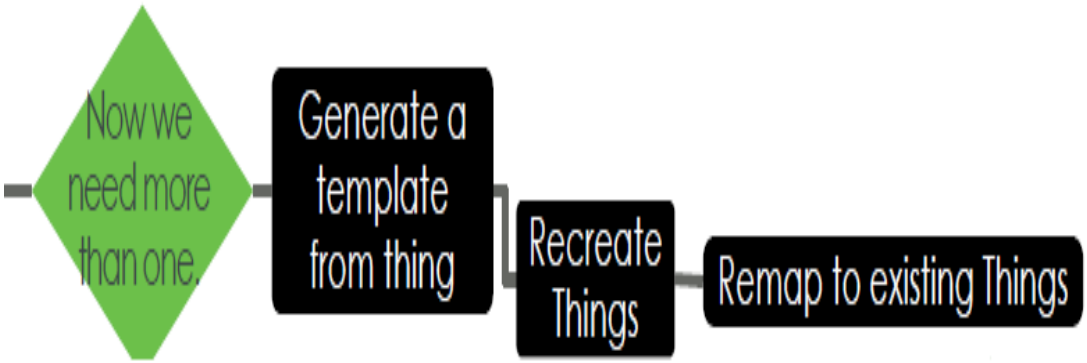
- If template/shape covers all the changing things and nothing else, edit template/shape.
- If not, two options, neither good:
  - Create a thing shape and apply to changing things individually (poor practice).
  - Create a new thing template and re-implement all the things that use it (a lot of work).



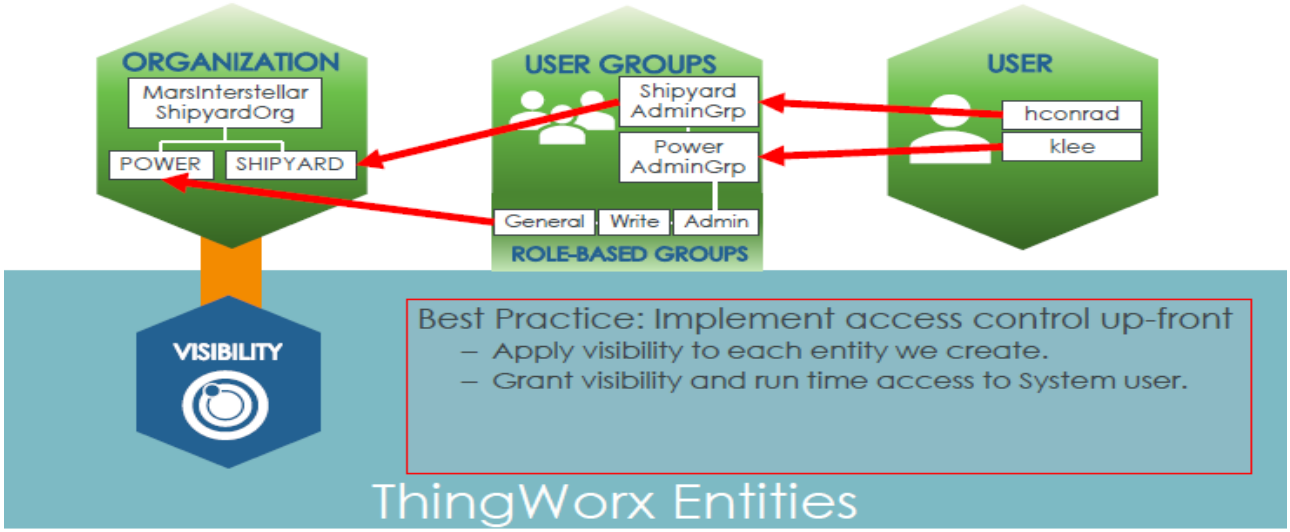
If all of the things of a given type change, the answer is simple – change the Thing Template or a Thing Shape that is used by all the things.



- What if you created a thing using the one-off modeling pattern. You never expected to have another thing like it, and now you do?
- You should create a thing template to encapsulate the functionality, then re-implement the thing.



Access Control Considerations



As we build our model, we need to apply visibility to each entity we create.

We don't have to apply run time access now, since our model only has a structure. We haven't given it any properties or services that would need runtime access control. We'll do that in the next module.

However, we do need to talk about the System user.

EXERCISE SESSION

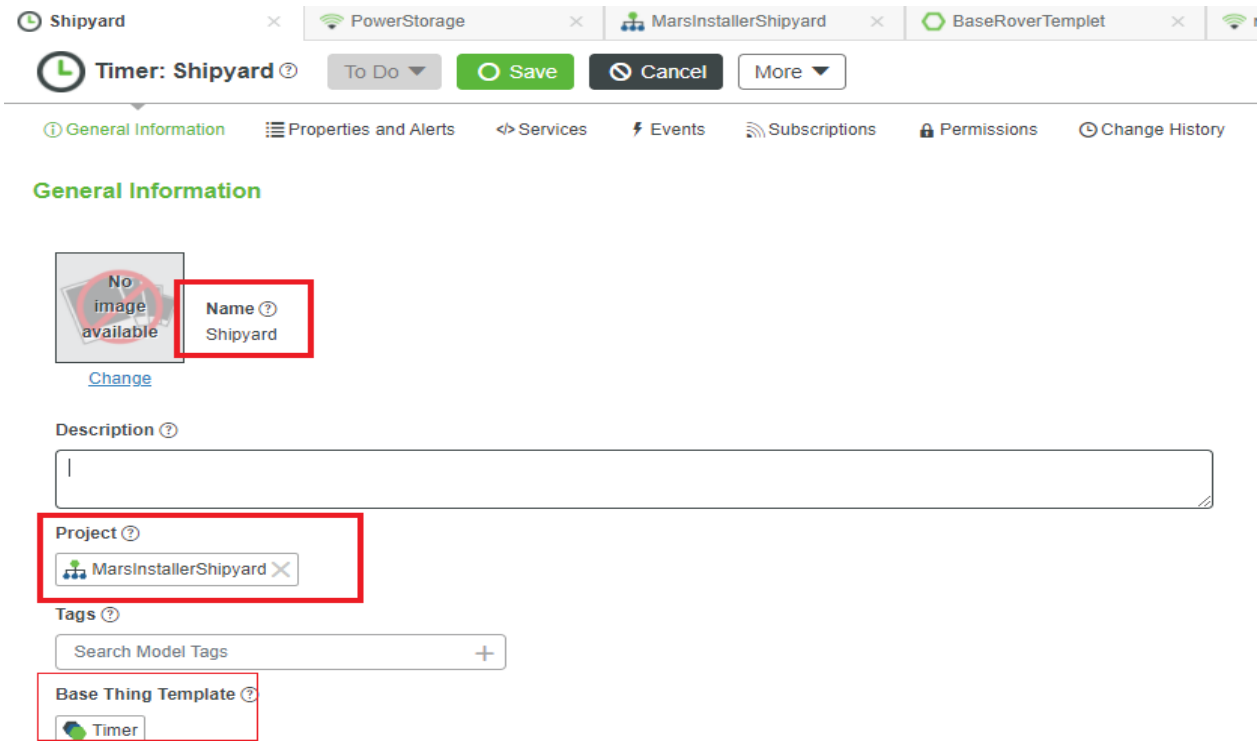
Exercise 1: Create Thing: Shipyard

In this exercise, you will create the Shipyard thing, set its visibility, and examine its timer configuration.

1. Create a thing entity with the following details:

Name	Project	Base Thing Template
Shipyard	MarsInsterstallerShipyardProject	Timer Simple

2. Click **Save**.

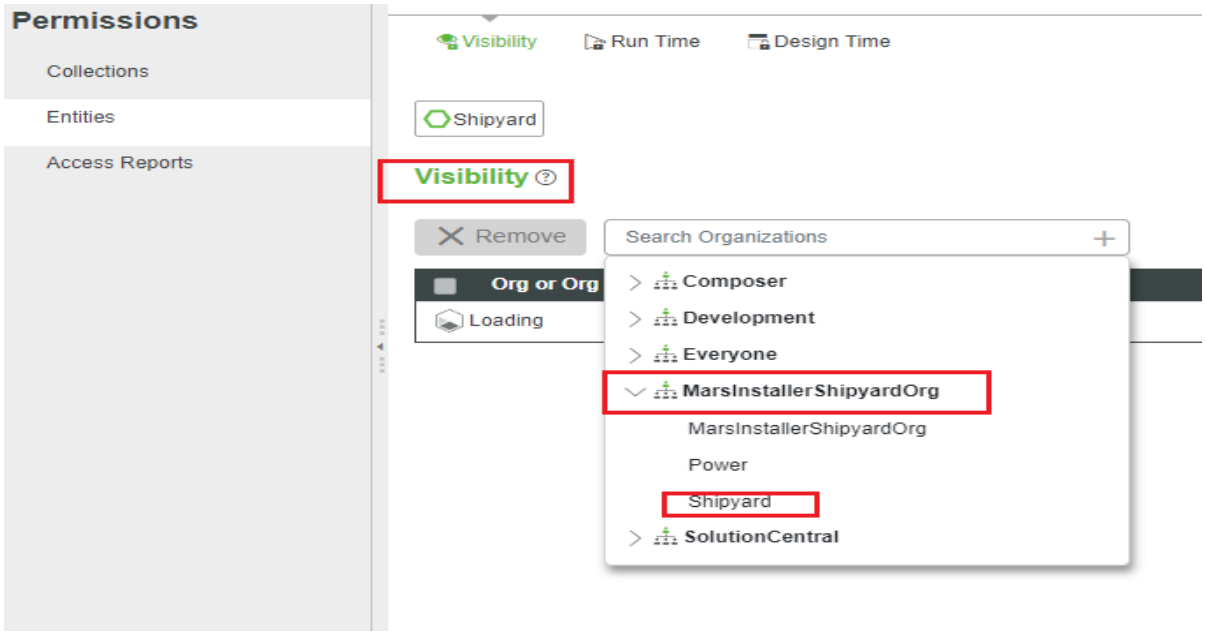


3. Navigate to the Permissions tab.

4. Verify the Visibility tab is selected.

5. Type **M** in the Search Organizations field and click to expand, but do not select MarsInterstellarShipyardOrg.

6. Click to expand MarsInterstellarShipyardOrg.



7. Select **Shipyard**.

8.Verifytheorganizationinthevisibilitytableis MarsInterstellarShipyardOrg: Shipyard.

9. Click **Save**.

10. Navigate to the Configuration page of the Shipyard thing.

Exercise 2: Create Thing Shape: Power Producer

In this exercise, you will create the Power Producer thing shape.

1. Create a Thing Shape entity with the following details:

Name	**Project**
PowerProducerTS	MarsInsterstallerShipyardProject

2. Navigate to the Permissions tab of the PowerProducerTS thing shape.

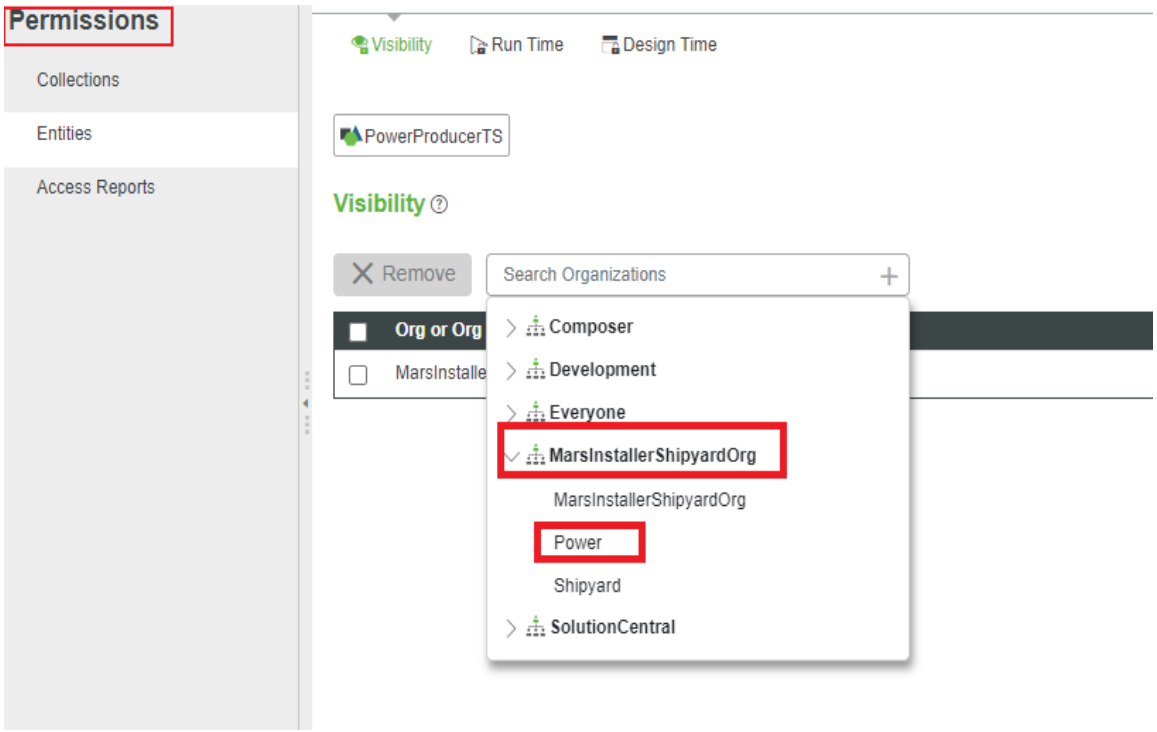
3. Verify the Visibility tab is selected.

4. Type **M** in the Search Organizations field and click to expand, but do not select MarsInterstellarShipyardOrg.

5. Click to expand MarsInterstellarShipyardOrg.

6. Select **Power**.

7. Likewise, add visibility to the MarsInterstellarShipyardOrg: Shipyard sub organization.



8. Verify the organizations listed in the visibility table are MarsInterstellarShipyardOrg: Power and MarsInterstellarShipyardOrg: Shipyard.

9. Click **Save**.

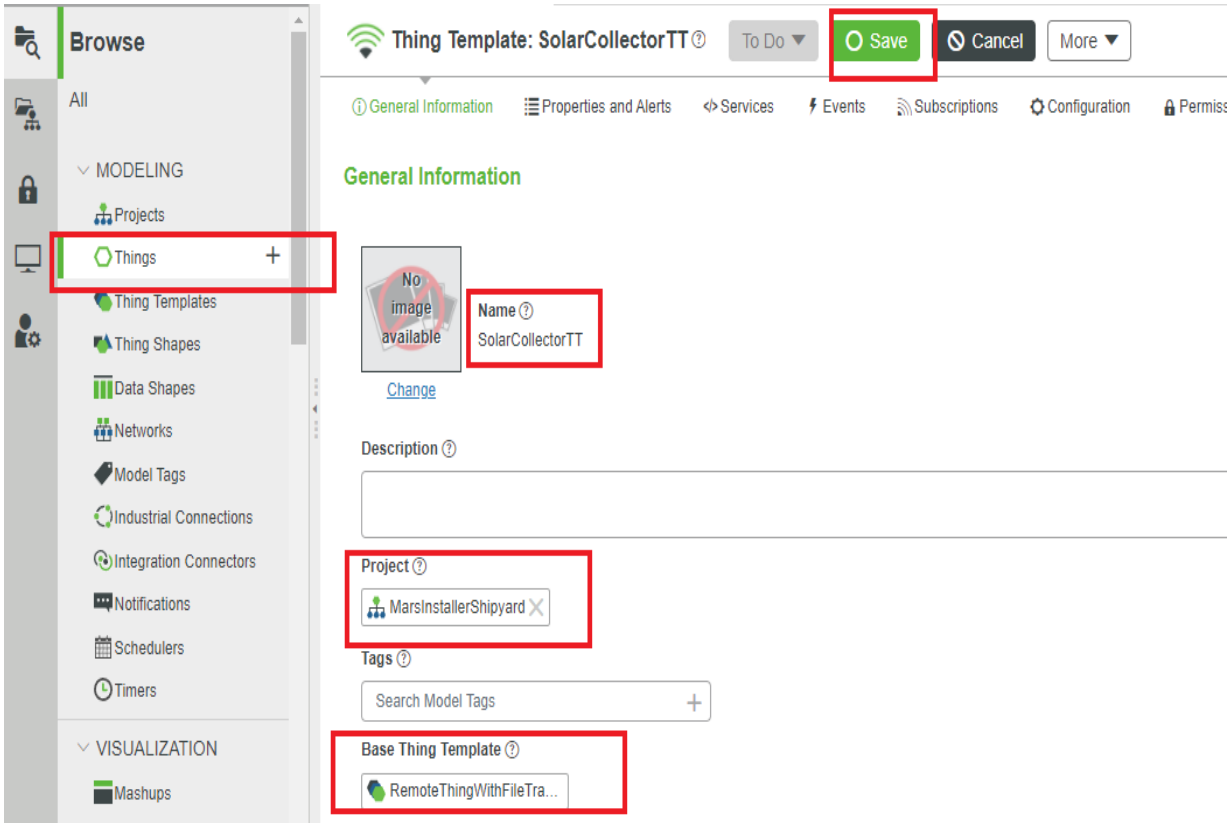
Exercise 3: Create Template: Solar Collector

In this exercise, you will create the Solar Collector thing template. Then, you will set visibility to the thing template and to things derived from the template.

1. Create a Thing Template entity with the following details:

Name	Project	Base Thing Template	Implemented Shape
SolarCollectorTT	MarsInterstellar Shipyard Project	RemoteThing WithFileTransfer	PowerProducerTS

Navigate to the Permissions tab of the SolarCollectorTT thing template.

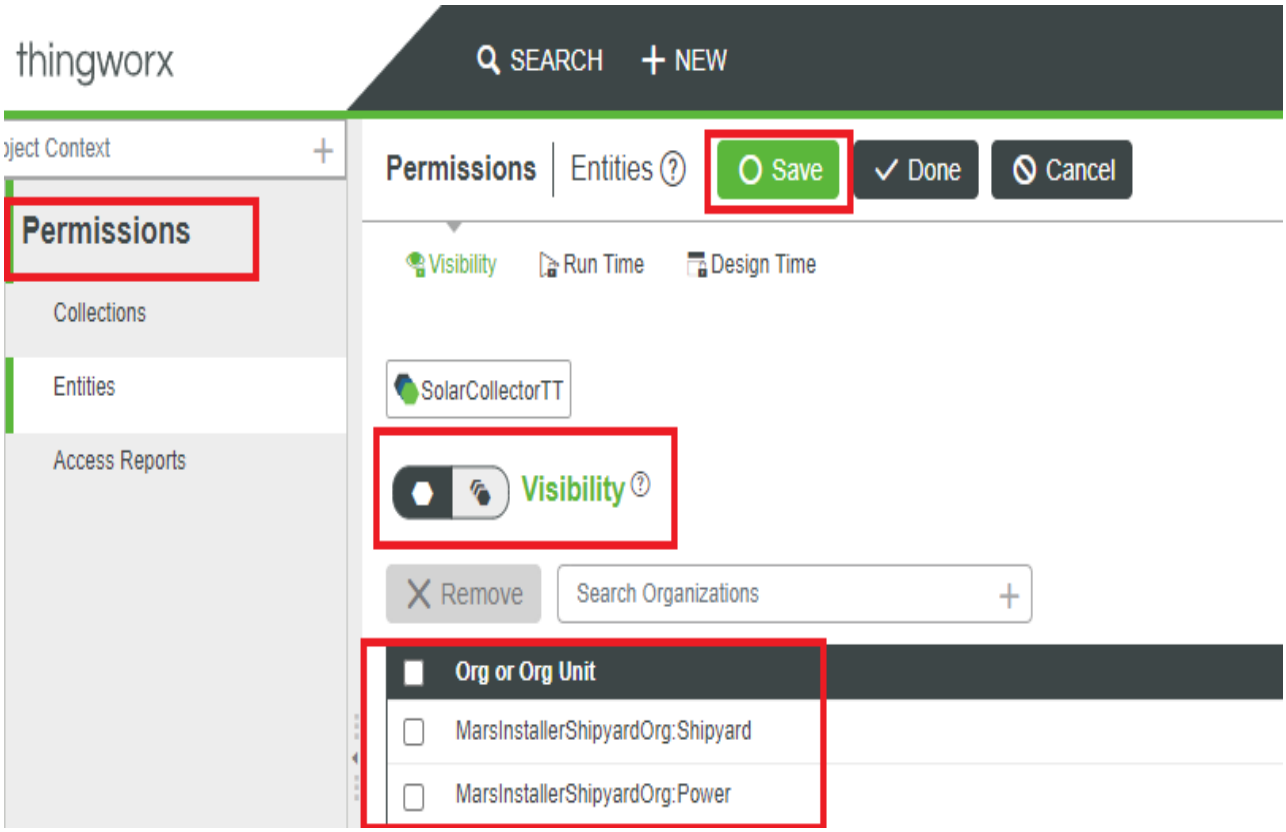


3. Verify the Visibility tab is selected.

4. Click the **Instance of Entity** button to set visibility instance permissions.

5. Grant instance visibility permissions to:

Org or Org Unit
MarsInterstellarShipyardOrg: Power
MarsInterstellarShipyardOrg: Shipyard



Exercise 4: Create Thing: Solar Collector

In this exercise, you will create a solar collector. Then, you will use access control reports to validate visibility to your entities.

1. Create a thing entity with the following details:

Name	**Project**	**BaseThing Template**
SolarCollector-1	MarsInsterstallerShipyardProject	SolarCollectorTT

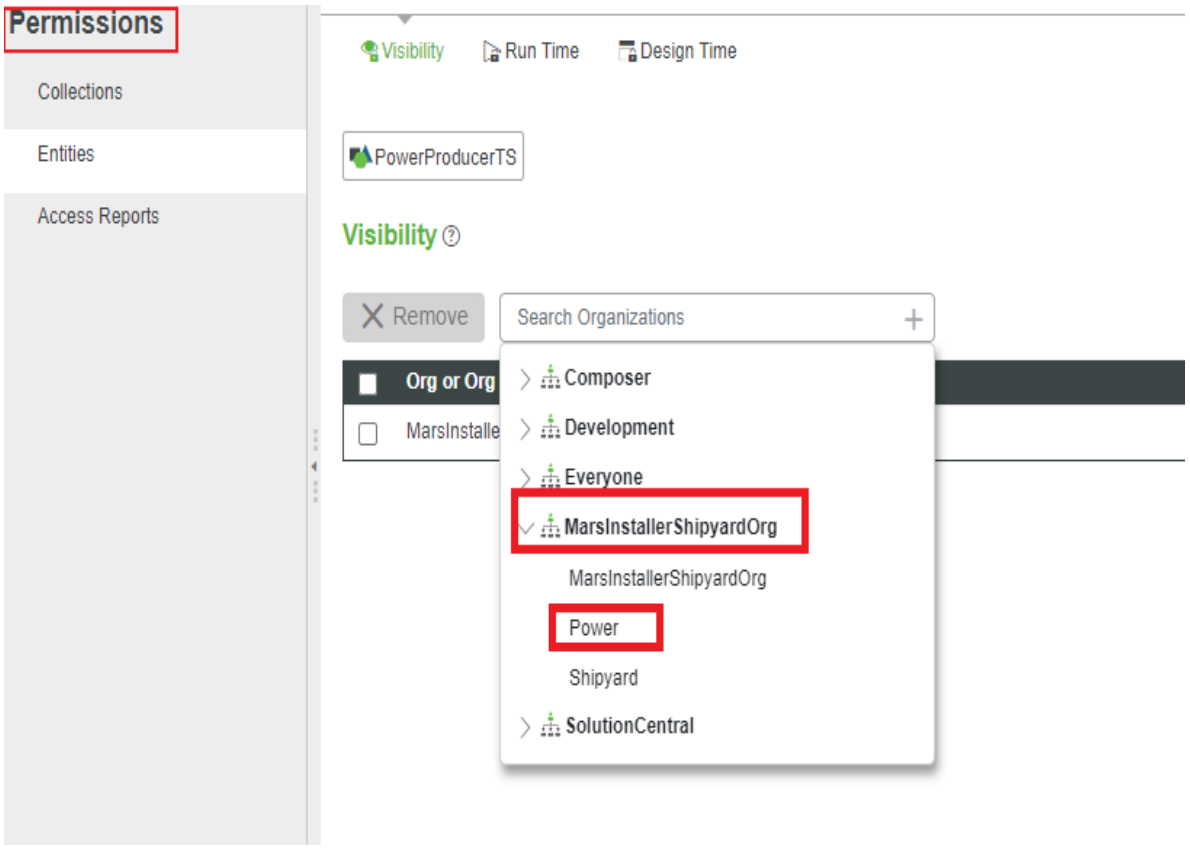
2. Click the **Permissions** tab.

3. Click the **Access Reports** link.

4. In the Search Users or Groups field, type/select **Klee**.

5. In the Search Entities field, click +.

6. Type/select **Shipyard**.



7. Click **Apply**.

8. Remove Klee from the User or Group field.

9. In the Search Users or Groups field, type/select **hconrad**.

10. Click **Apply**.

11. Click the symbol to view the visibility details report for Hector.

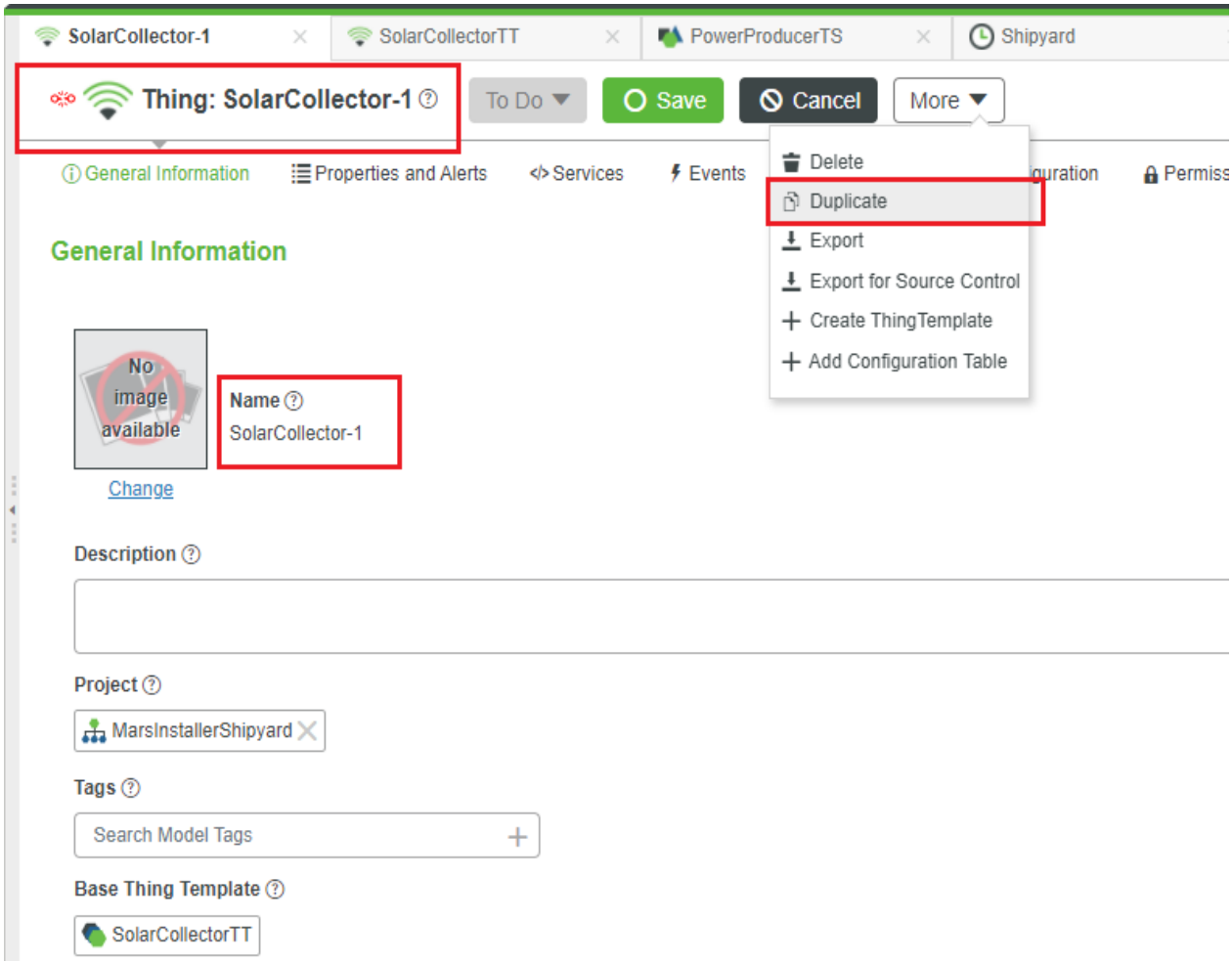
12. Hover over the MarsInterstellarShipyardOrg in the visibility report to view the full text.

Exercise 5: Duplicate Thing: Solar Collector

In this exercise, you will create a second solar collector by duplicating the one you created in the previous

1. Select the **Browse** tab.

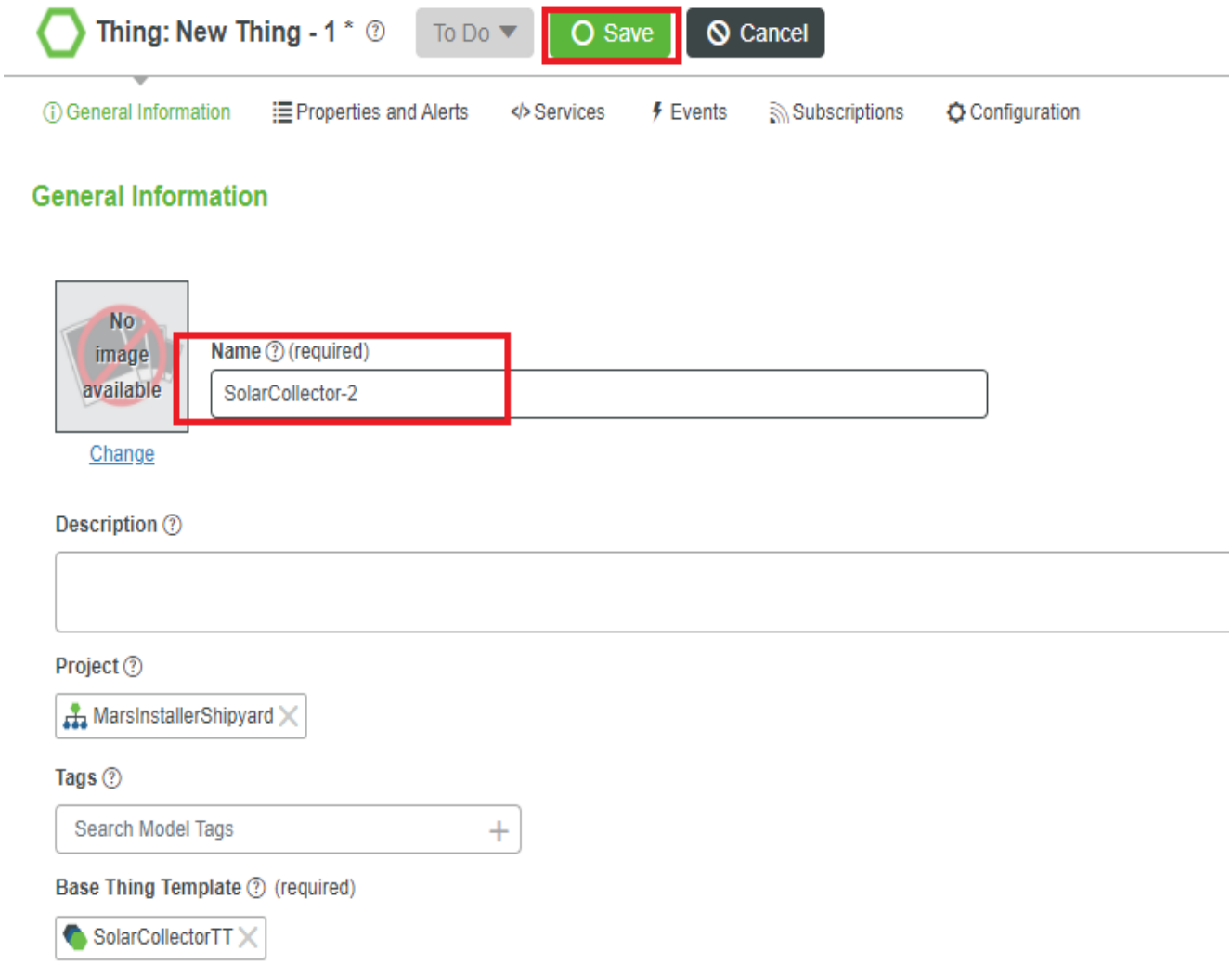
2. Click **Things** under the MODELING section.



3. Select the check box in the **SolarCollector-1** row from the entities list area.

4. Click **Duplicate**.

5. Type **SolarCollector-2** in the Name field.


Thing: New Thing - 1 *
To Do
Save
Cancel
General Information
Properties and Alerts
Services
Events
Subscriptions
Configuration
General Information
No image available
Change
Name (required)
SolarCollector-2
Description
Project
MarsInstallerShipyard
Tags
Search Model Tags
Base Thing Template (required)
SolarCollectorTT

6. Click **Save**

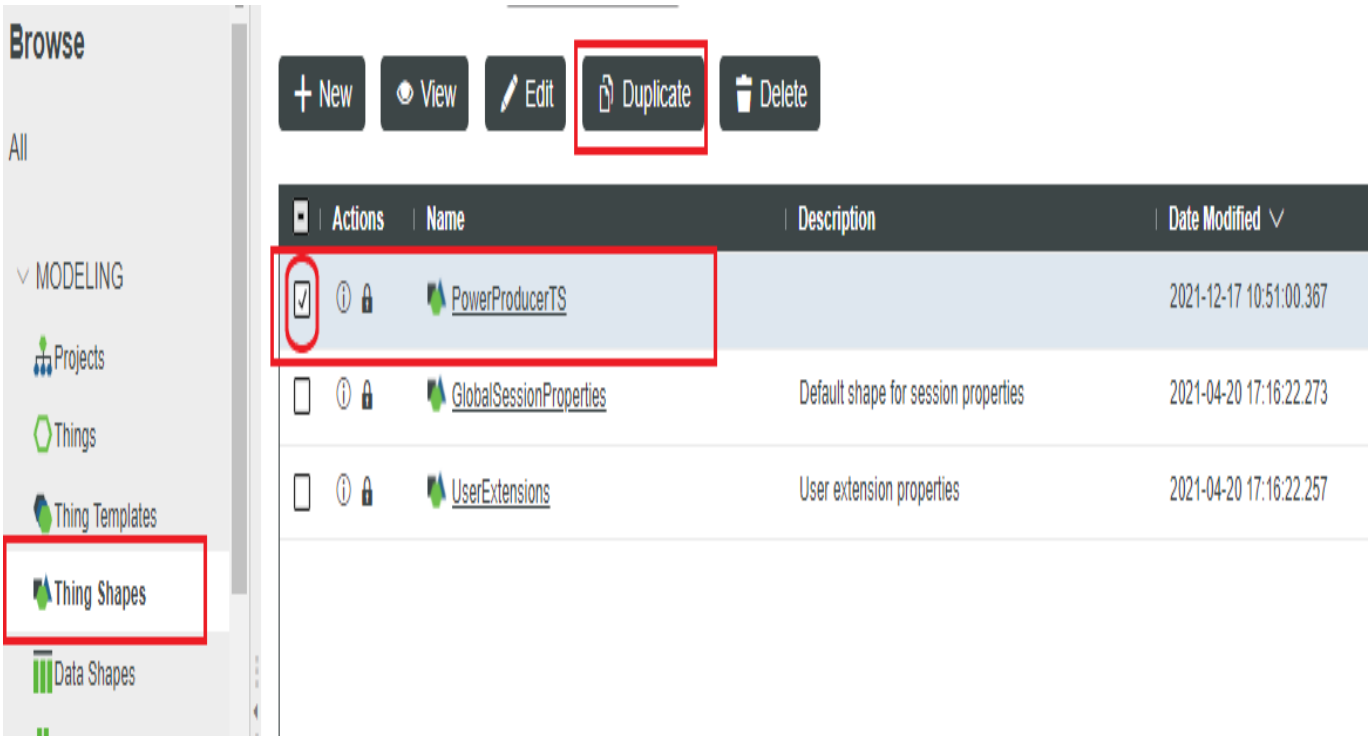
Exercise 6: Create Thing Shape: Power Consumer

In this exercise, you will create the power consumer thing shape by duplicating the power consumer thing shape.

1. Select the **Browse** tab.

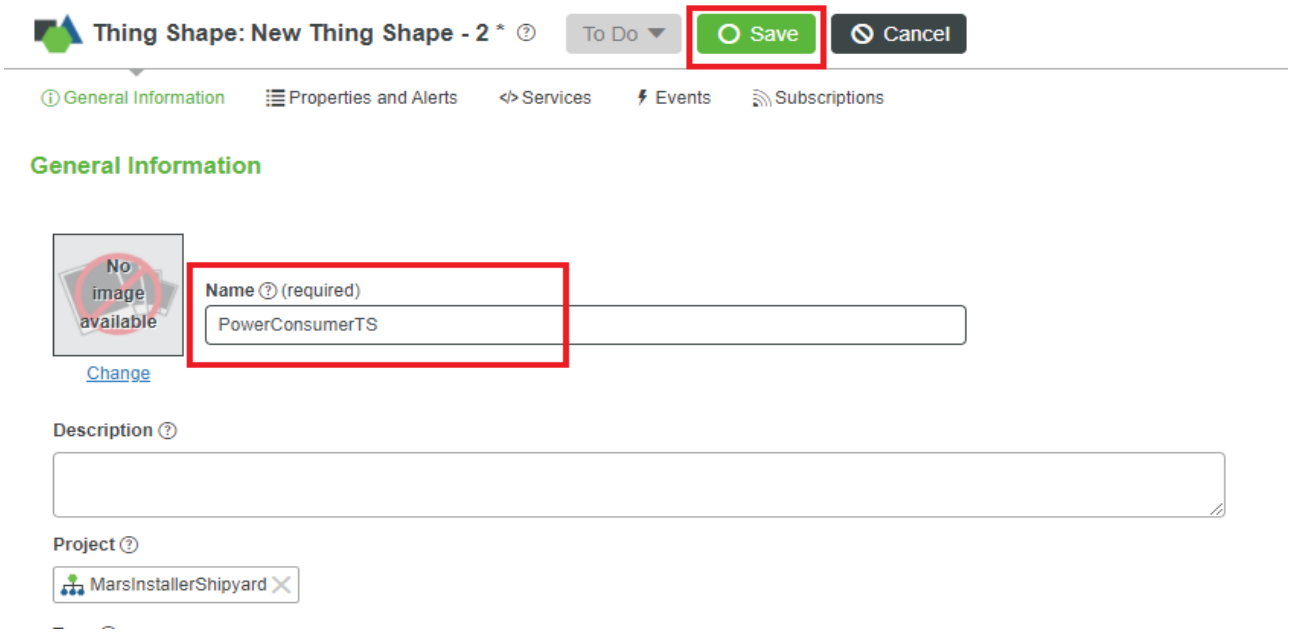
2. Click **Thing Shapes** under the MODELING section.

3. Select the check box in the **PowerProducerTS** row.



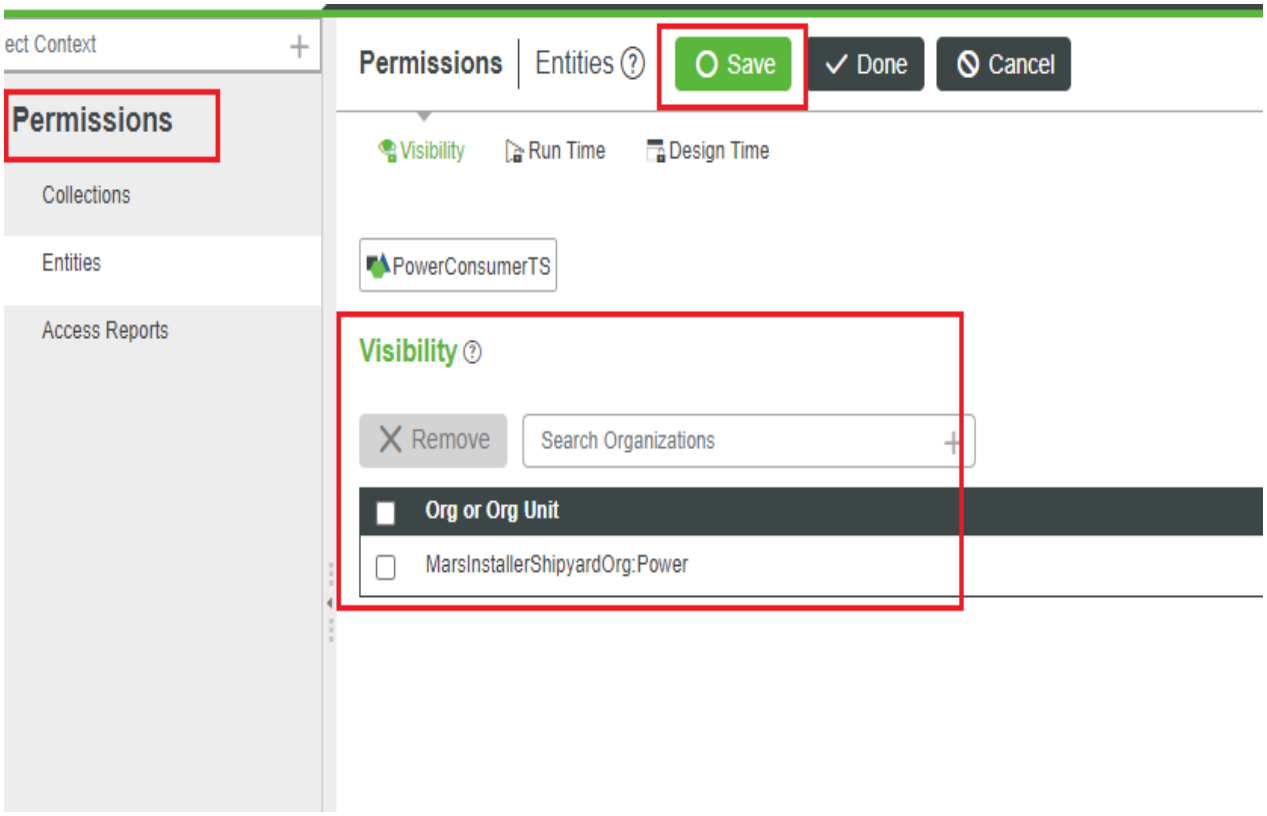
4. Click **Duplicate**.

5. Type **Power Consumer TS** in the Name field.



5. Click **Save**.

6. Click **Permissions** to verify the power and shipyard units have visibility.



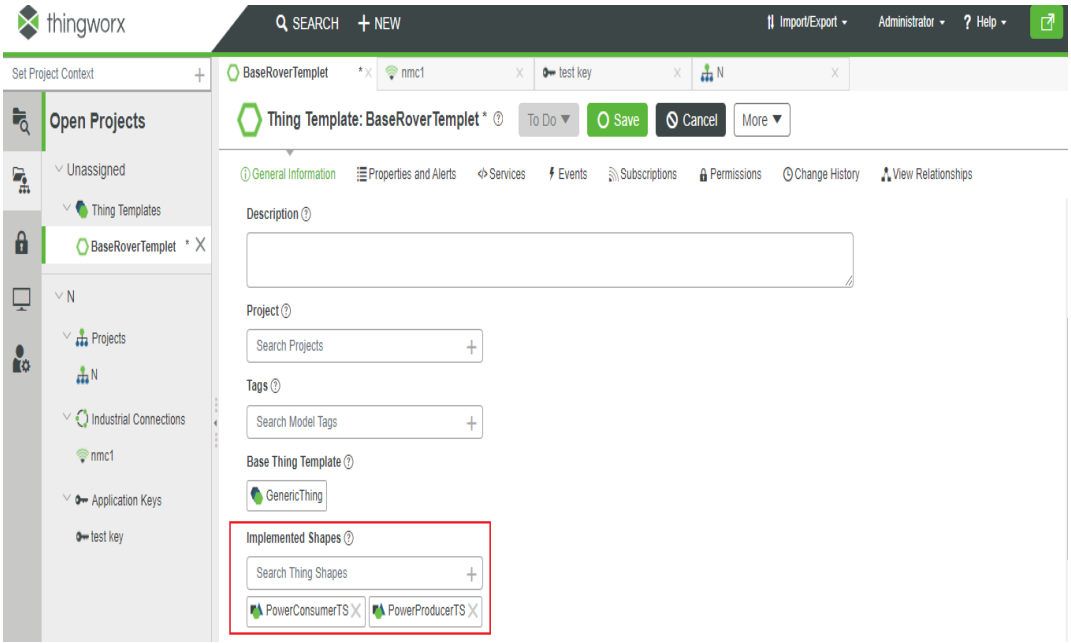
Exercise 7: Iterate Things: Mars Rover

In this exercise, you will edit the mars rovers which you created in the Fundamentals of IoT with ThingWorx course. These entities aren't part of your project, aren't implemented as power producers or power consumers, and don't have visibility configured. You're going to fix that.

1. Click **SEARCH** on the header of the New Composer. Type/select **BaseRoverTemplate** to open the BaseRoverTemplate thing template.

2. In the Implemented Shapes field, type/select **PowerConsumerTS**.

3. In the Implemented Shapes field, type/select **PowerProducerTS**.

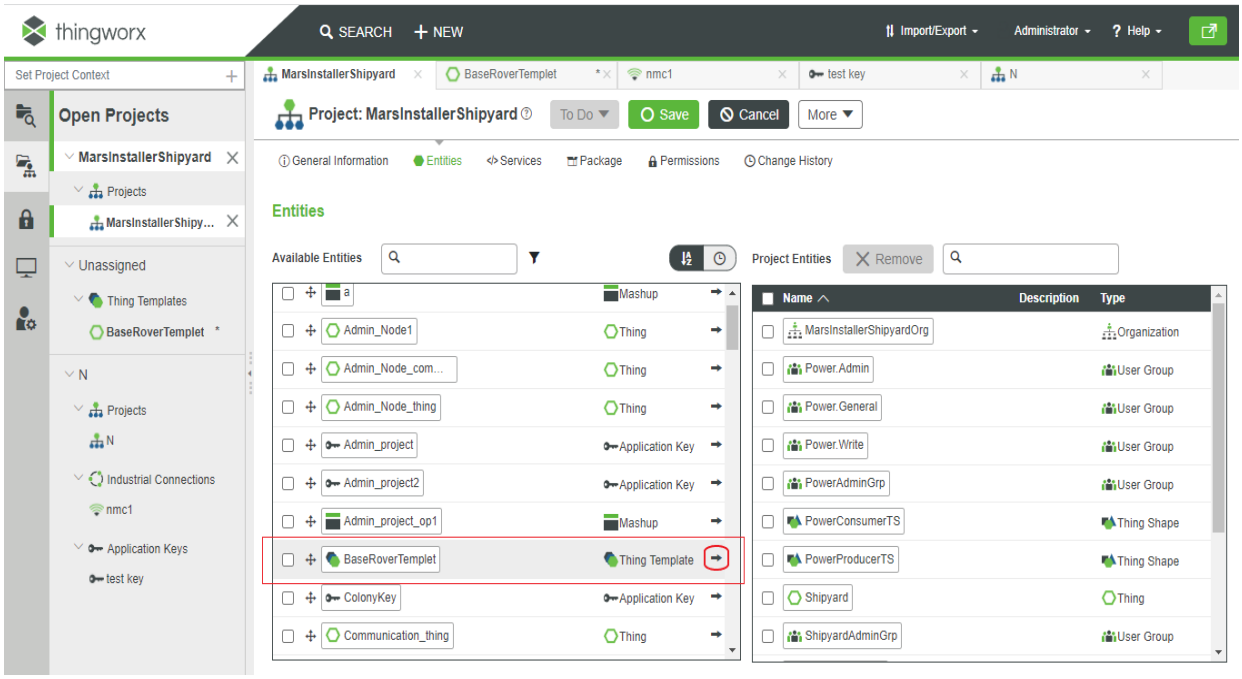


4. Click **Save**.

5. Grant visibility permissions for the BaseRoverTemplate to:

Visibility Permissions	Org or Org Unit
Entity	MarsInterstellarShipyardOrg
Instance	MarsInterstellarShipyardOrg

6. Click **Save**.



7. Open the MarsInsterstallerShipyardProject.

8. Select the **Entities** tab.

9. From the Available Entities table, for the following seven entities, click the **black arrow** on the right column to bring them into the Project Entities table:

`

- BaseRoverTemplate
- CrewRover1
- CrewRover2
- CrewRover3
- CrewRoverTemplate

- WeatherRover1
- WeatherRoverTemplate.

10. Click **Save**.

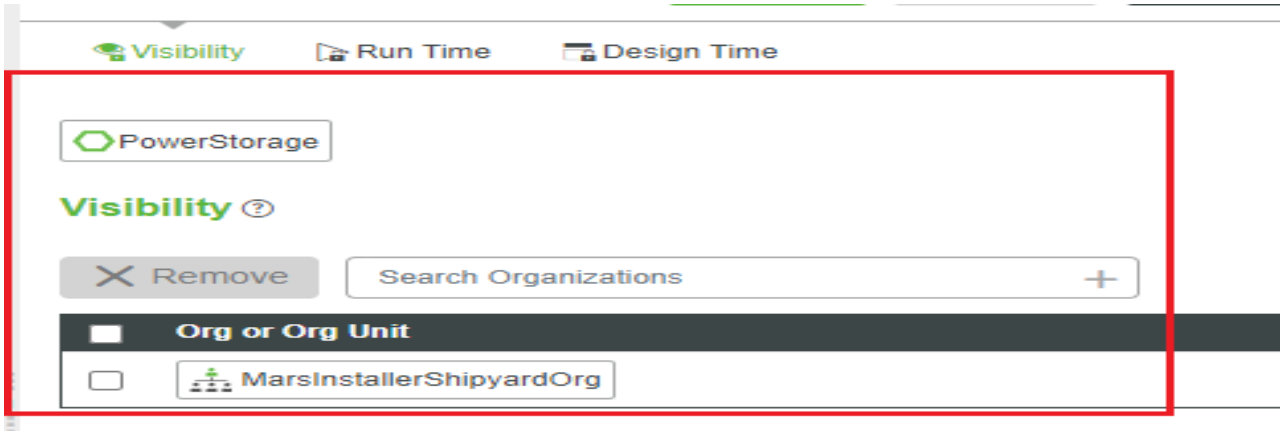
Exercise 8: Create Thing: Power Storage

In this exercise, you will create the power storage thing.

1. Create a Thing with the following details:

Name	Project	Base Thing Template
Power Storage	MarsInsterstallerShipyardProject	RemoteThing

2. Grant visibility to the entire MarsInterstellarShipyardOrg organization.



Data Strategy

A data strategy is a highly dynamic process employed to support the acquisition, organization, analysis, and delivery of data in support of business objectives

Data Strategy is a strategic plan to manage data as a corporate asset. It ensures a sustainable competitive advantage in the future and with a positive rate of return on investment. As with any strategic asset, its absence jeopardises the future wellbeing of the organization

What happens without a strategic view of data?

Your current competitors will be optimizing and reducing costs and targeting your customers with more relevant communications. Your vendors adding data-rich services that connect them directly to your customers. When data becomes an afterthought, your agility is reduced by delays launching new insights, features, and products. Most significantly an eye out for startups in your industry that have worked out how to disintermediate your business

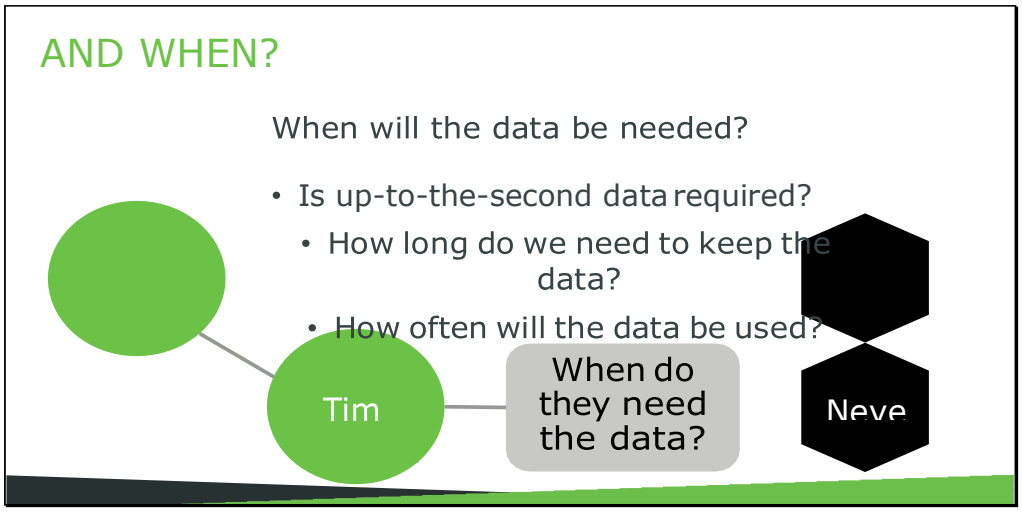
Which Data and When?

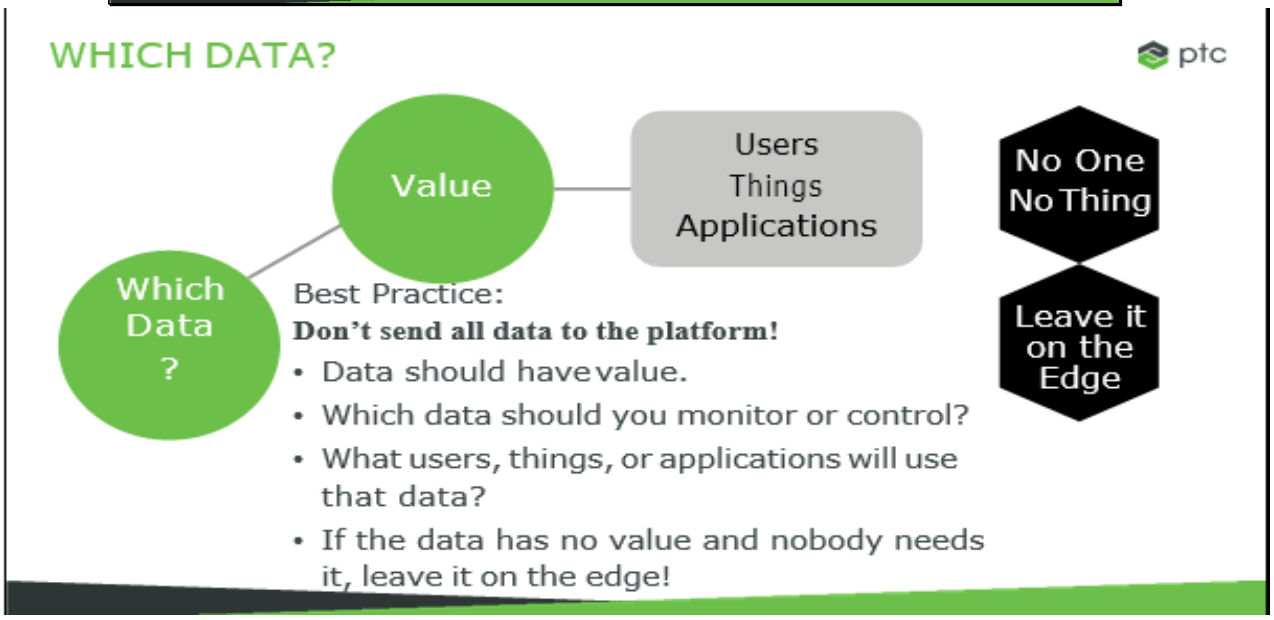
In this phase, we look at the IoT data and how you interact with it. This is where we f lush out the functionality of the system – the properties, services, events, and subscriptions.

No, No, Never! Data Strategy Framework

A data strategy is needed to determine what data we need and a plan on how to handle that data. How up-to-date does the data need to be? Do we get the data when we need it, or is the data pushed to the platform?

Which Data?





A new IOT developer will frequently send all the data they can to the platform, just because they can; that's a bad practice. You should only send data that will be used and has value.

What users will use that data? Will it help fulfil any of our use cases, by monitoring, controlling, optimizing, or automating a device?

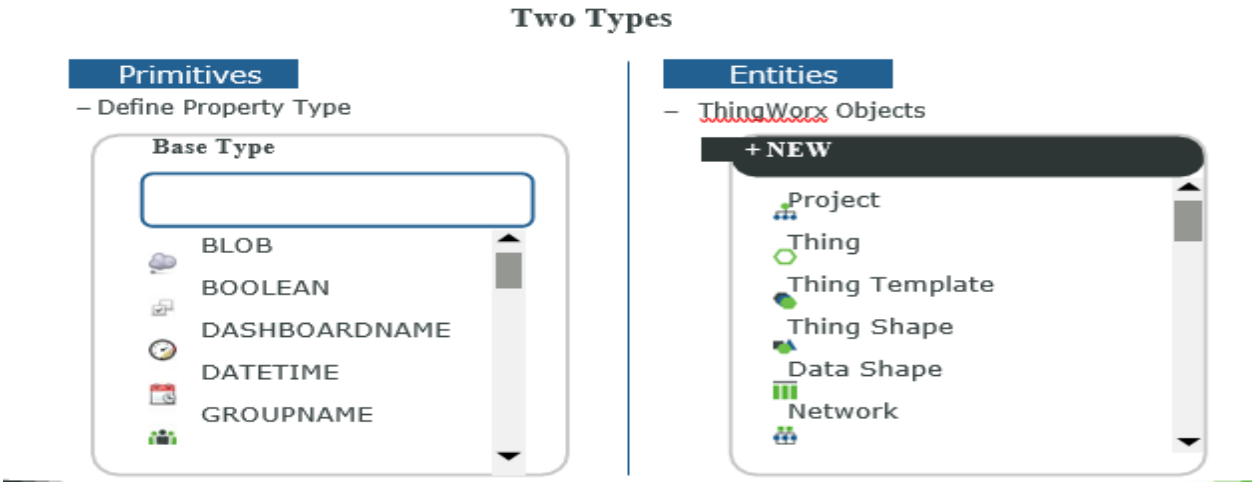
And When?

The next consideration: When is the data needed? Up-to-the-second data may not be required. For example, our weather stations may measure the Martian temperature, which doesn't change rapidly. Sending a measurement every second would be pointless. If we send a measurement every five minutes, we would still always have a very accurate temperature.

The next question is how long the data will be relevant for. If we need to keep the data forever but it's unlikely to be read, such as audit records, we may need some kind of cold storage or data backup facility.

Lastly, the data may have value, but how frequently is it really used? If the data isn't used very often, maybe we should leave it on the edge and have the platform request it when it's needed, instead of the edge proactively sending values that are unlikely to be used

DATA STRUCTURES



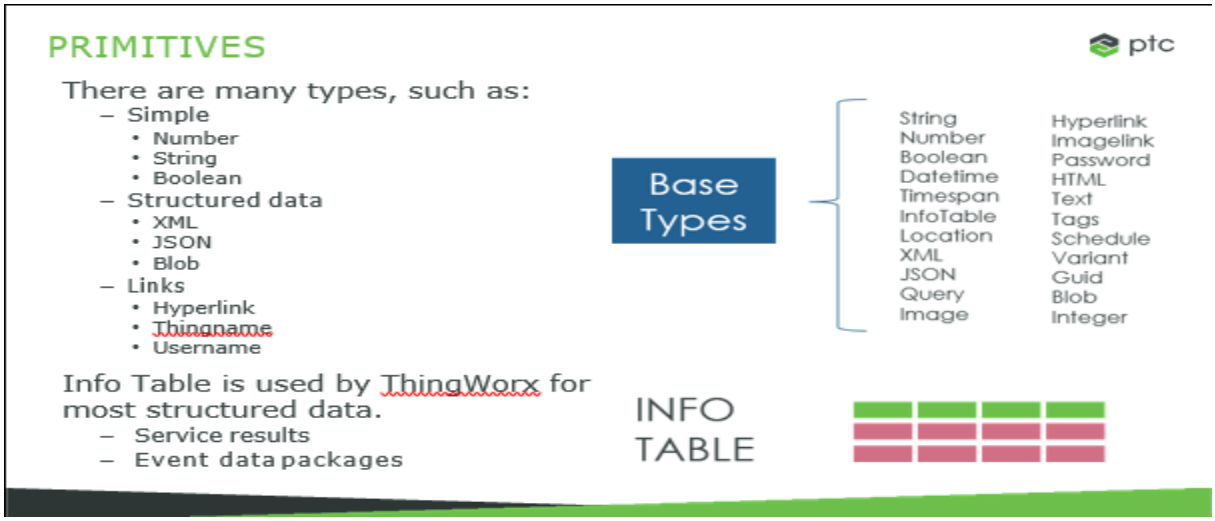
There are two types of data structures in ThingWorx.

Primitives are used for properties- defining the type of data that can go in them. For example, a temperature property uses the number primitive, enabling it to store numeric values.

Entities are ThingWorx objects in their own right. For example, you created the MarsInsterstallerShipyardProject, which is an entity, but it is also a data structure, containing links to all the entities in the project.

The easiest way to tell the difference between the two is that primitives are listed as the Base Type of a property. Entities are listed in the Create a New Entity drop-down list.

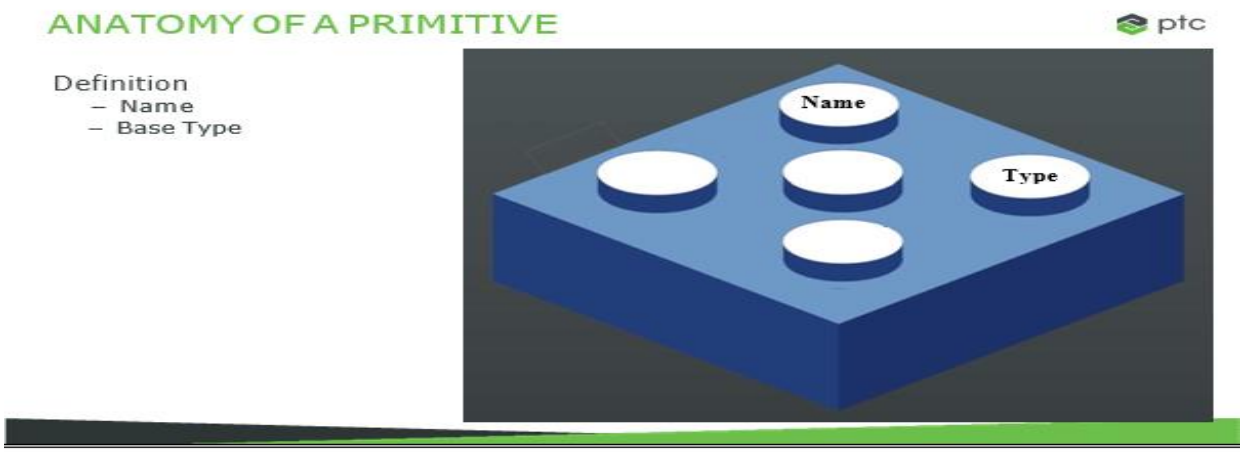
Primitives



There are lots of different primitive types. We won't go into all of them in detail. Some are simple, such as numbers and strings. Others have structure data, such as XML or JSON. Still others contain links to other places or objects, such as hyperlinks or thing names.

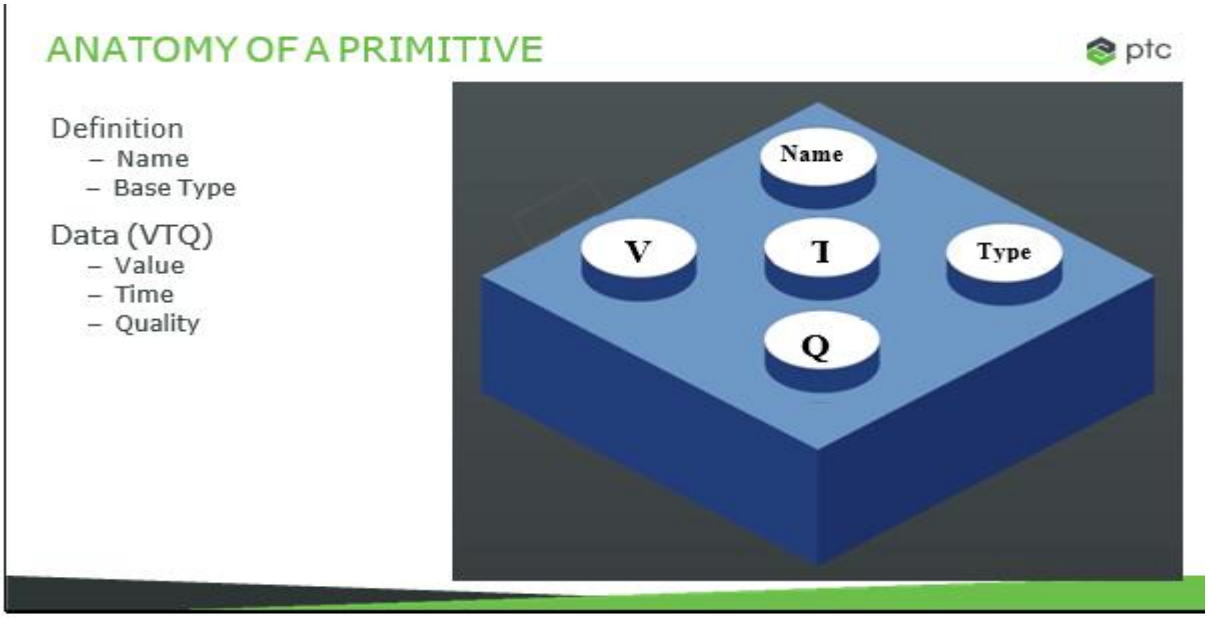
From a ThingWorx standpoint, the Info Table is extremely important. It is used internally by ThingWorx to pass structured data, such as the results of a service containing more than one piece of data.

A primitive has five components.



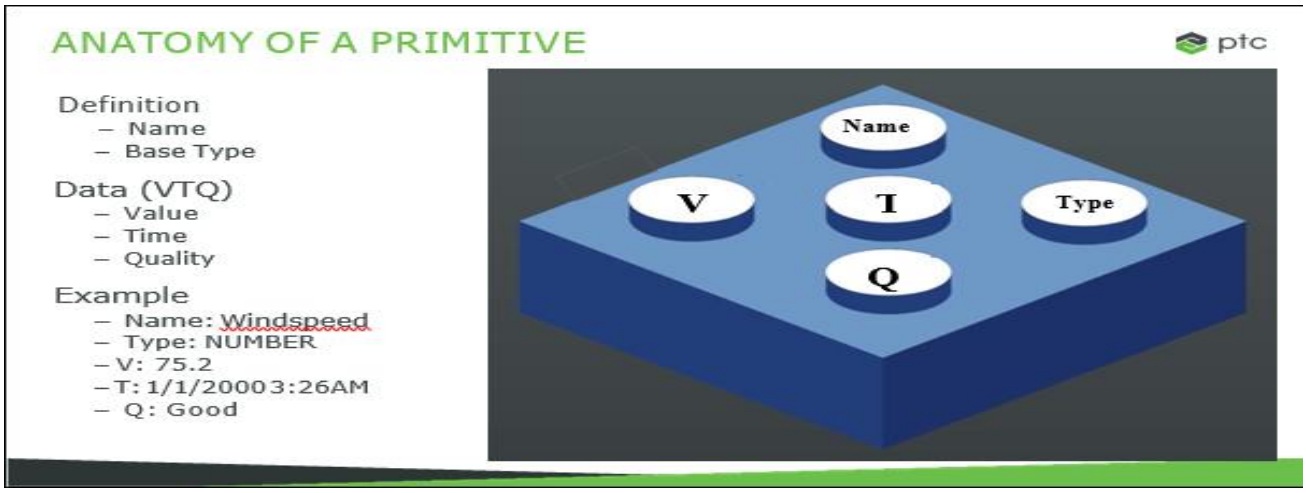
The first two defines the primitive, its name and its base type. For example, the property named wind speed uses the base type number.

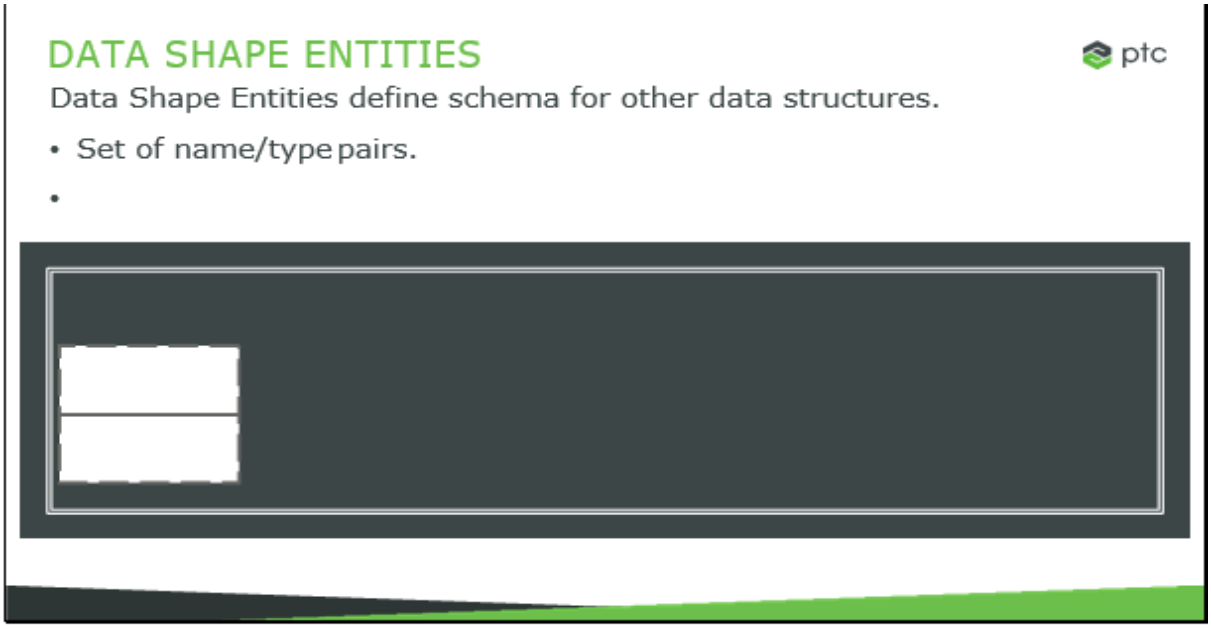
Anatomy of a Primitive



The other three components are data components. Your application is typically most interested in the value of the primitive. However, a primitive also has a timestamp of when the data was valid and a quality attribute defining how confident the sensor or device is in that value. Many applications simply ignore the timestamp and quality attributes.

Data Shape Entities

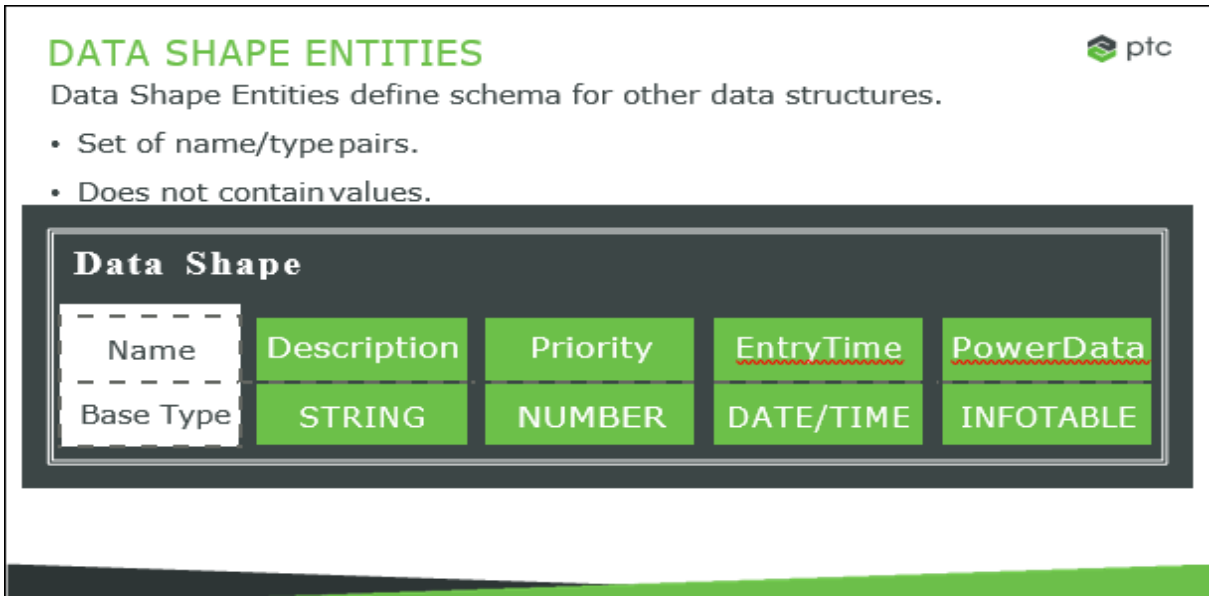




A data shape is an entity used to define the schema of other entities, such as data tables and info tables.

It has a set of valid name/type pair

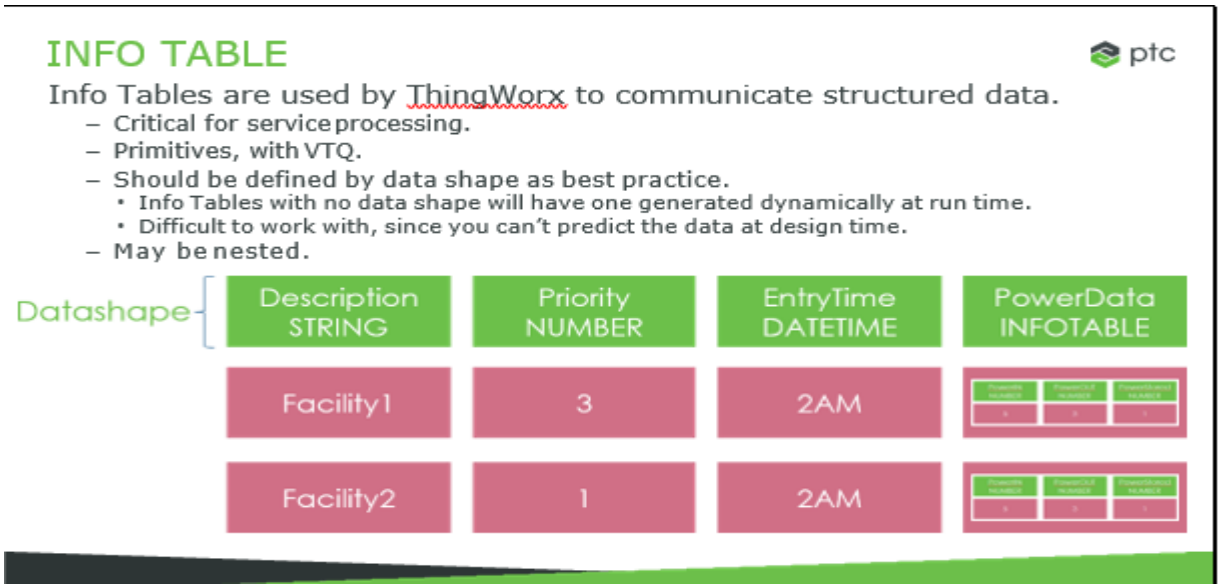
Data Shape Entities



In the example, we have a description string, a priority number, an entry time, and an info table named power data.

When you apply a data table to another entity, it makes entering any other primitive illegal. So, we couldn't add an image blob – it isn't listed in the data table. One key point is that the data table doesn't actually contain values – just name/type pairs. The values are stored in the entity you apply the data table to

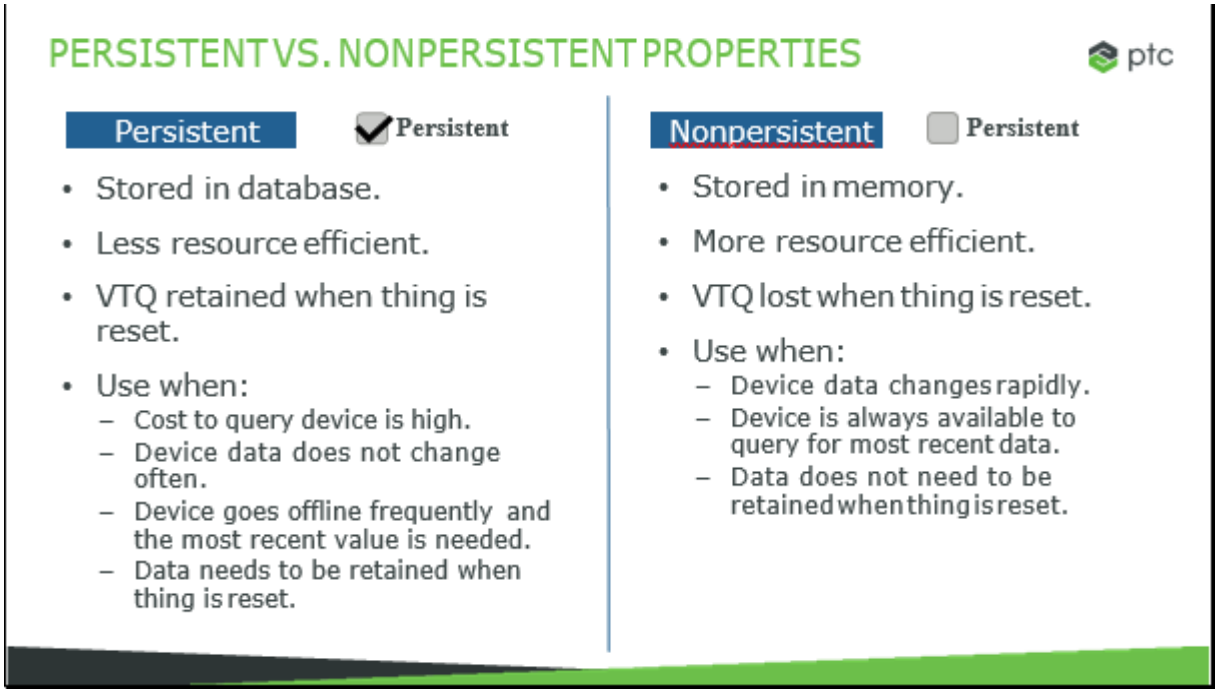
Info Table



Info Tables are critical to the understanding of ThingWorx data. They are used to communicate service results, used by the mashup builder to predict data returned from a service, to structure event and alert data, and package data for communication with the edge

They are primitives, with value, time, and quality. They should always be defined by a data shape. Without the data shape, there is no structure to tell you what is in the Info Table, which makes getting usable data out of it much harder. And, since an Info Table is a primitive, and Info Tables contain primitives, they can be nested, placing an Info Table inside and Info Table.

Persistent vs. Nonpersistent Properties



A persistent property is stored in the ThingWorx database, while a non-persistent property is stored only in memory.

There are advantages and disadvantages to both strategies.

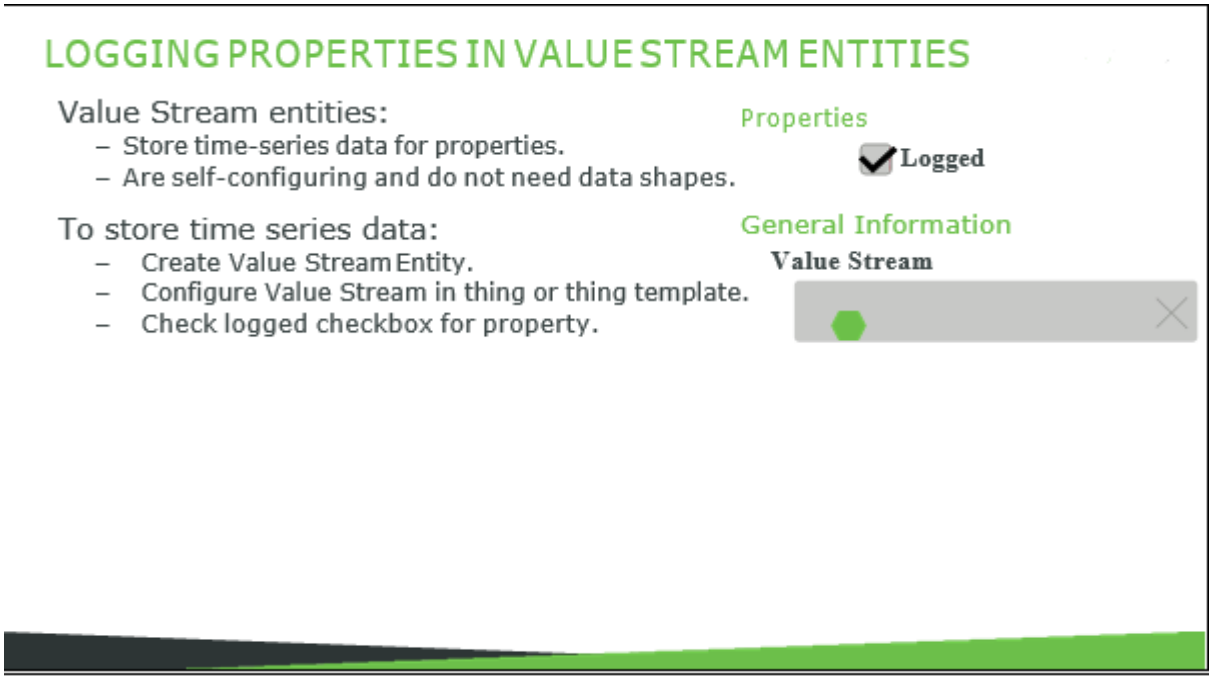
The main advantages of a persistent property Is that the value is retained locally on the ThingWorx server, even if the thing is reset. You can reset a thing by restarting the server, editing the thing, or editing anything shapes or thing template that is used by the thing.

Nonpersistent properties are more resource efficient. They don't have to write to the Thing Worx database every time they query the device. However, their value is lost if the system is reset.

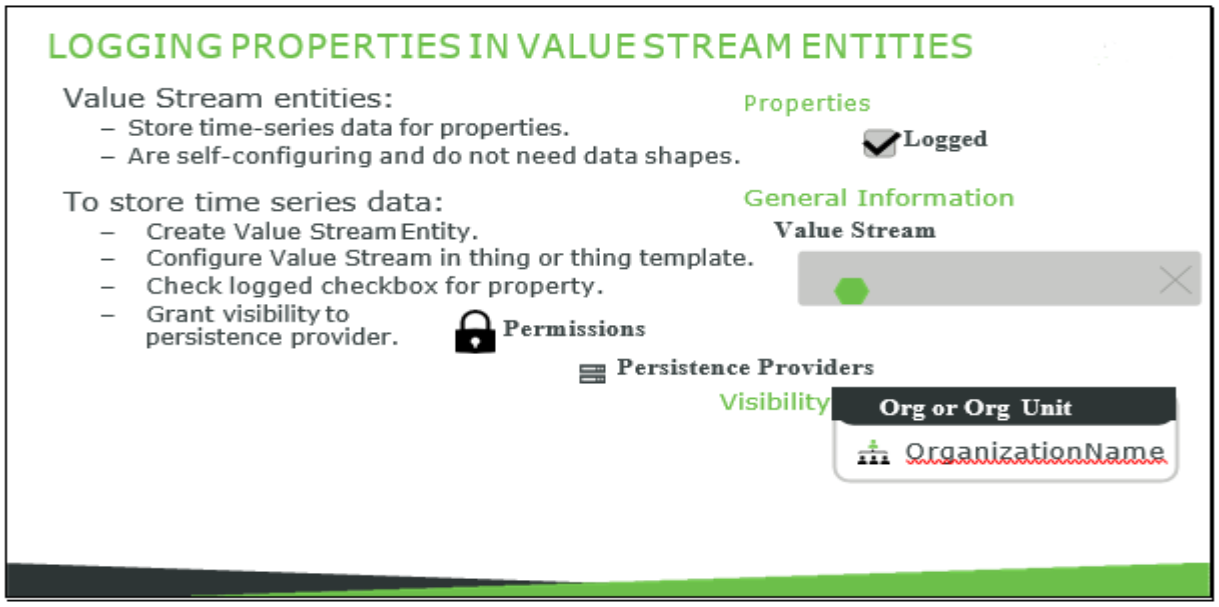
As a general rule, if you can get the value from the device at any time, there's no reason to persist the value. Thing Worx will just get it when it needs it.

However, if the device is calculated inside Thing Worx, or the device can't always be relied on to provide the data on request, you should persist the property.

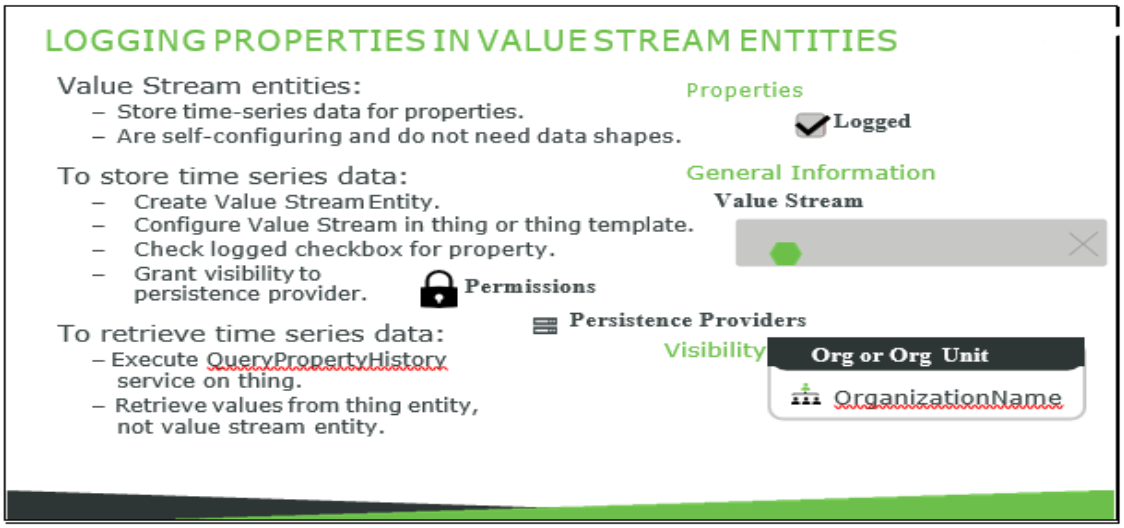
Value stream entities store time-series data for properties that have the logged check box selected.



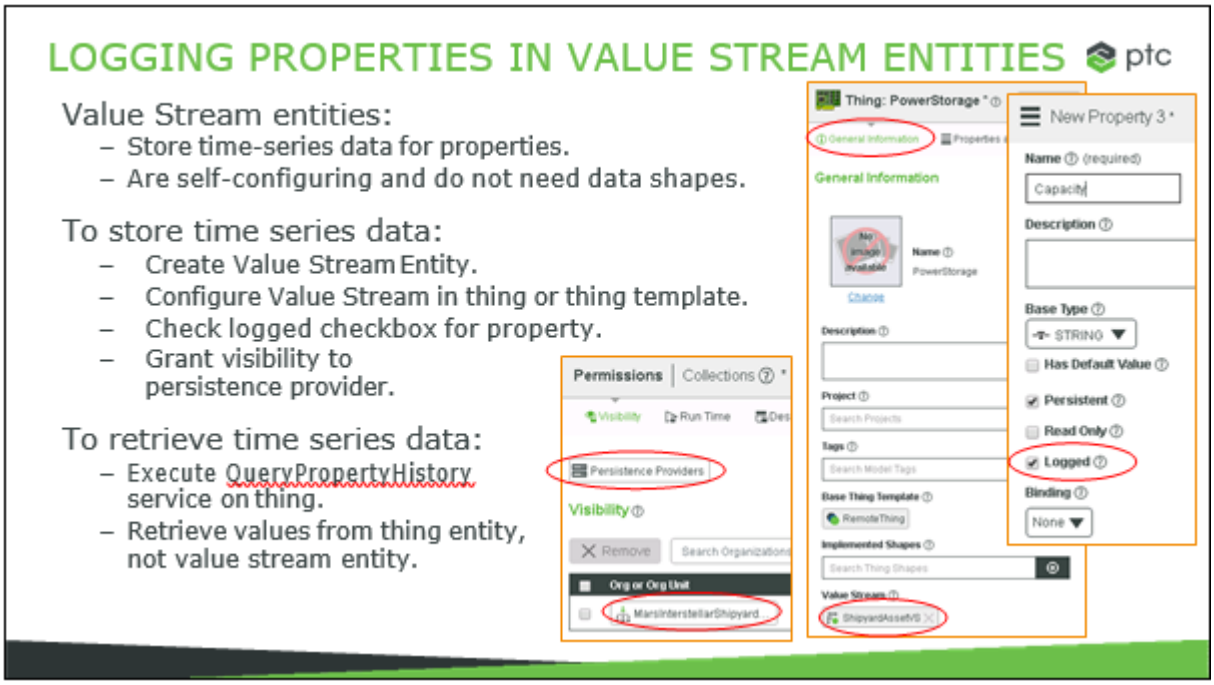
However, there is some configuration that goes into setting up a value stream. First, the value stream entity must exist. You can either create one, or use one that is already in the system. Then, on the General Information page of the thing or thing template, you have to specify which value stream to use for time-series data.



And lastly, every value stream stored its data in a persistence provider, which is basically the database where the data is stored. The user needs to have visibility to that persistence provider. The easiest way to do this is to simply grant everyone visibility to the persistence provider collection.



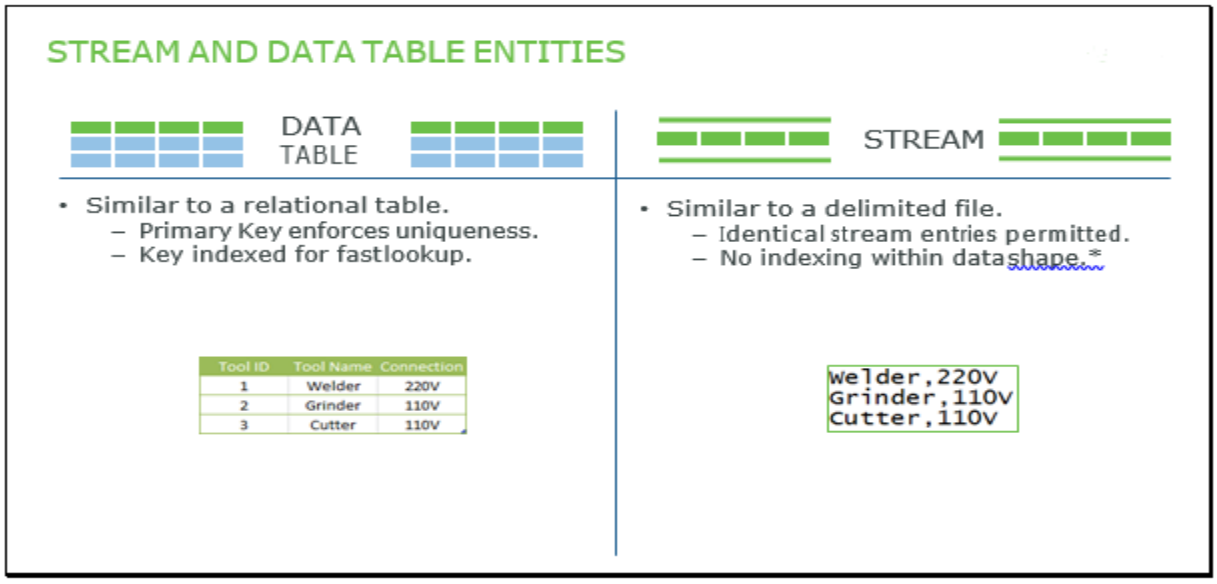
Usually it is acceptable to let the user base know a persistence provider exists – that isn't generally secured information. Since it's just visibility, it doesn't give them the ability to edit or query the persistence provider.



Value stream entities store time-series data for properties that have the logged check box selected.

However, there is some configuration that goes into setting up a value stream. First, the value stream entity must exist. You can either create one, or use one that is already in the system. Then, on the General Information page of the thing or thing template, you have to specify which value stream to use for time-series data. And lastly, every value stream stored its data in a persistence provider, which is basically the database where the data is stored. The user needs to have visibility to that persistence provider. The easiest way to do this is to simply grant everyone visibility to the persistence provider collection. Usually it is acceptable to let the user base know a persistence provider exists – that isn't generally secured information. Since it's just visibility, it doesn't give them the ability to edit or query the persistence provider.

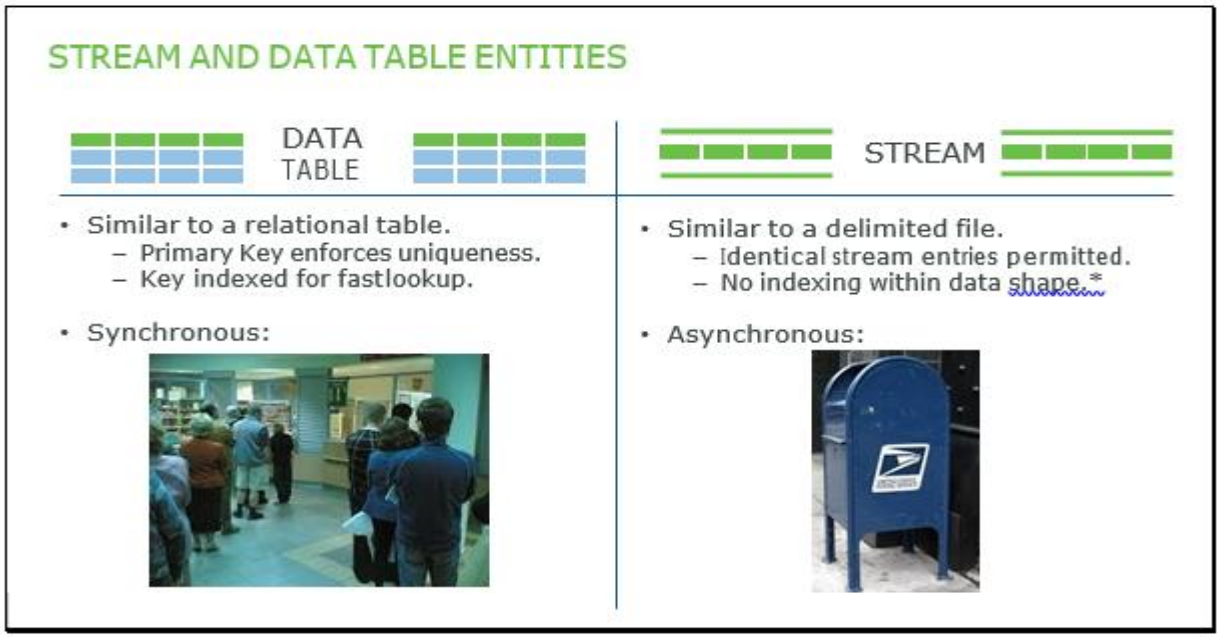
Streams and data tables are two different entities designed to hold structured data.



Deciding between a stream and a data table isn't always an obvious choice. In many cases, either data structure will serve your needs equally well.

However, we will go over some of the differences between the two data structures to help decide which may be better for your use case.

STREAM AND DATA TABLE ENTITIES

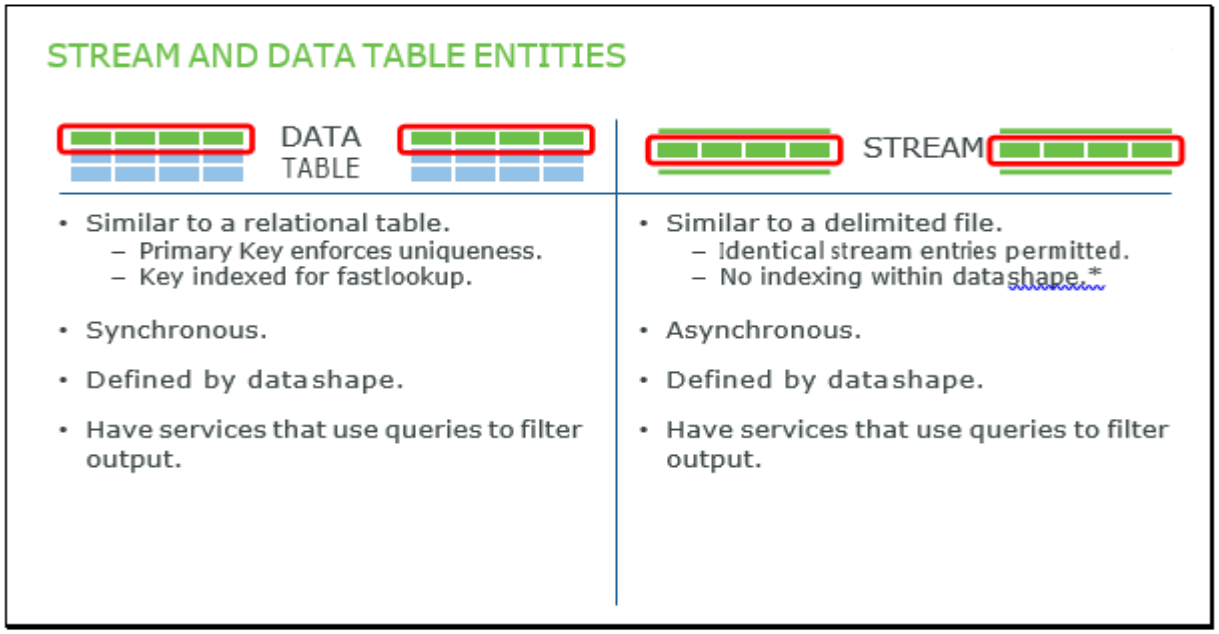


•A data table is intended to be similar to a relational table. It has a primary key which is indexed, which can make lookups using that key much faster; looking up Tool ID number 347 would be very fast.

•A stream is more like a delimited file. None of the data fields are indexed, so lookups can be slower. Even if there was a tool ID number, it wouldn't be indexed and the stream would be searched sequentially. However, streams have other lookup advantages that we will cover in a moment.

•So, if you are doing frequent lookups against a key or need to enforce uniqueness, use a data table.

STREAM AND DATA TABLE ENTITIES

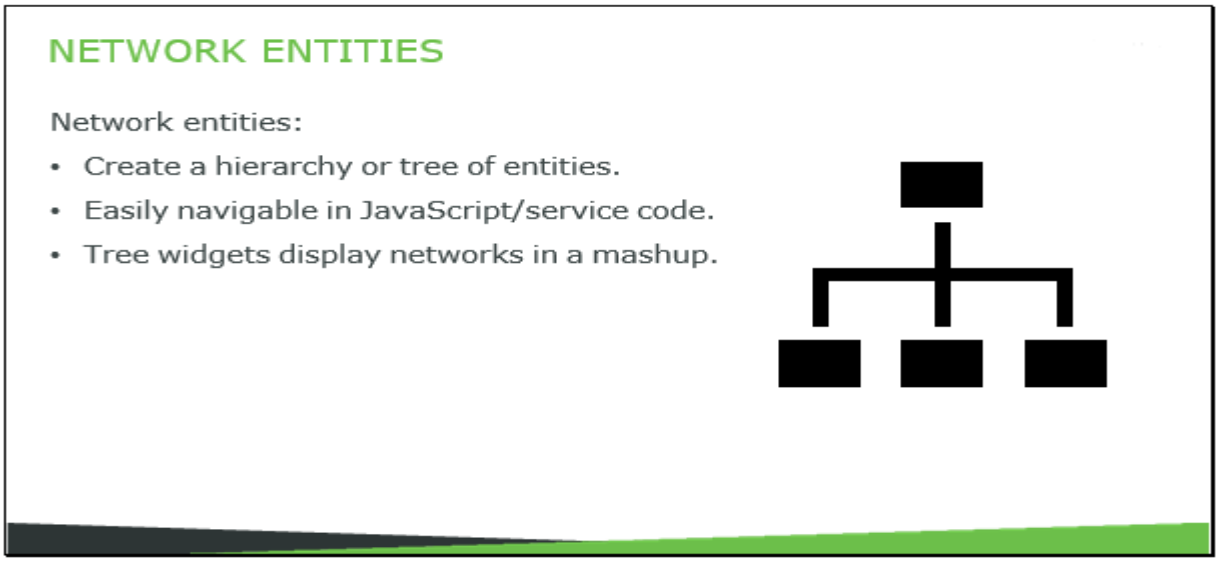


Writing an entry to a data table is synchronous, like waiting in line at the post office. Everybody waits their turn until the postal clerk can help you. The data table locks the entry so nobody else can edit it, takes the entry, and provides confirmation the entry was processed.

Writing an entry to a stream is asynchronous, more like dropping off a piece of mail; you just drop it off. There is no guarantee that another letter won't be processed before yours. However, you don't have to wait for a confirmation, making the data entry must faster. Although it is less reliable than a data table, it is still extremely reliable.

So, if it is critical that entries be processed in a specific order, or you need confirmation that the entry was processed, you should use a data table. If not, consider a stream as the more efficient option.

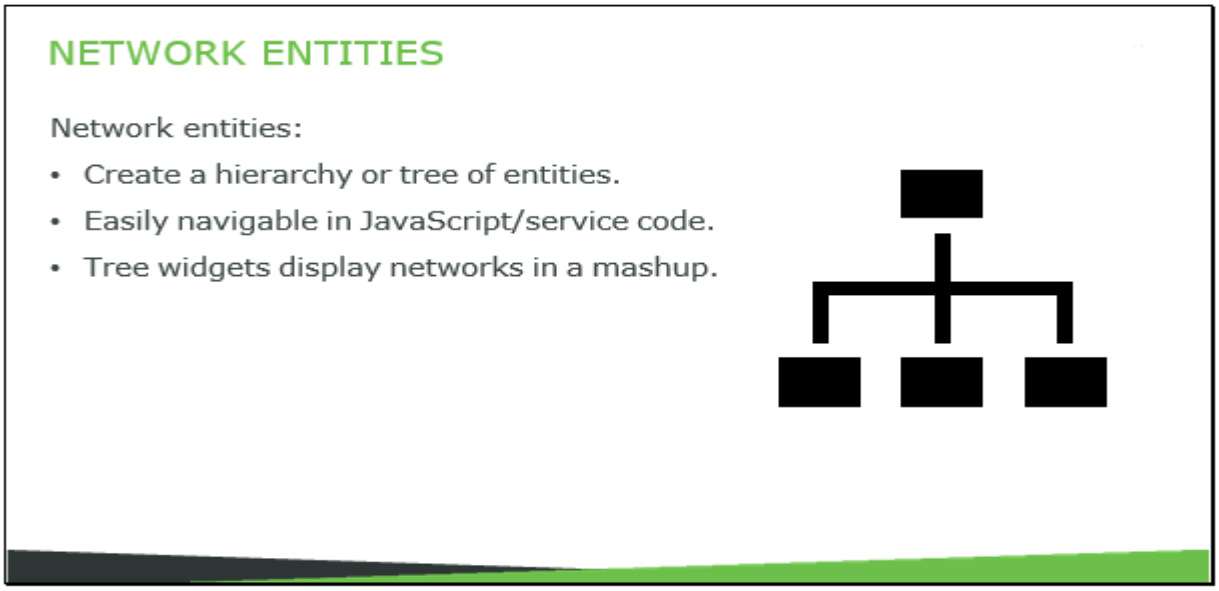
STREAM AND DATA TABLE ENTITIES



Both data tables and streams require a data shape to define the data that may be entered into them. In our graphics of the data structures, we represent the data shape as the green row

Both types of data structures also have services available that enable you to use services to filter the query output.

Network Entities



Network Entities are a tree of related entities. They can be used to represent any tree structure, from entities at given locations, to a bill of materials.

The advantage of the Network entity is that it's easy to work with and navigate from entity to entity. We have widgets to display trees from networks in mashups, and APIs for navigating networks.

Let's create a network entity for Boltron, our giant construction robot.

Exercise 1: Creating Networks: Form Boltron Network

In this exercise, you will create a Network entity to store the current configuration of a giant modular

construction robot named Boltron. Boltron is a gigantic robot which has sub-robots that attach to it that

perform separate tasks, such as welding, grinding, drilling, and more. It can also have sub-robots that contain

smaller sub-robots for smaller drilling or construction jobs.

Objectives:

After successfully completing this exercise, you will be able to:

• Create a network entity.

• Add things to a network.

Task 1: Create the Boltron network

1. Create a new Network entity with the name Boltron Network belonging to the Mars Interstellar Shipyard Project.

Task 2: Add things to the Boltron network

3. Click the Network Hierarchy tab.
4. Type Bool in the search field.

Note: Boltron is the top-level entity in the network. This giant robot has one top-level entity. However, other networks may have more than one top-level entity.

5. Drag Boltron from the left panel to the right panel.
6. Select the BlueBoltron, GreenBoltron, RedBoltron, and YellowBoltron check boxes.
7. Drag all four things to Boltron in the right panel. The network should now look similar to this:

Note: The BlazingArcWelder can only be used by Red Boltron.

8. Drag Boltron Blazing Arc Welder from the left panel to RedBoltron.

Note: Green Boltron is responsible for connecting the Dyno herms and InfraCells.

9. Select the Boltron Dyno Therms and BoltronsInfraCells check boxes.
10. Drag both things to Green Boltron. The network should now look similar to this:
11. Click Save.

Exercise 2: Creating Properties: Power System

In this exercise, you will create the properties that track power produced, consumed, and stored. This three

properties are in three separate entities. Power stored is in our power storage thing, while power produced and

power consumed are in their respective thing shapes. You will log these properties to the value stream.

Objectives:

After successfully completing this exercise, you will be able to:

• Create properties.

• Log properties to value streams.

• View historical values of logged properties.

• Set run time access to properties.

Task 1: Create the Power Stored property for the power storage thing.

1.Edit the Power Storage thing.

2. Add a property with the following details:

Name	Base Type	Units	Min Value	Max Value	Persistent	Logged
Power Stored	NUMBER	kWh	0	500000000 (eight zeroes)	True/Checked	True/Checked

2.Expand Advanced Settings and set the following details:

Data Change Type	Data Change Threshold
Value	100

Instructor Note:

Communicate that the power storage facilities capacity for measuring power stored is based on its current voltage, a mechanism which isn't completely accurate. Changes of less than 100 kWh are likely to be sensor inaccuracy, so they aren't counted as data changes.

4. Click Done.

5. Click Save.

Task 2: Configure the power stored property to log changes to a value stream

Note: We have a logged property, so the thing needs to be associated with a value stream. We'll use the VS value stream which has been created for you.

6. Select the General Information tab.

7. In the Value Stream field, type/select VS.

Instructor Note:

Tell the students to remember that for a value stream to work, the user needs visibility to the persistence provider the value stream is using. This has already been configured for them in ACM.

8. Click Save.

Note: Our thing now has a property. If anyone is to use that property, they need run time access to it. We'll grant the Power. General and Shipyard AdminGrp permission to read all properties for the power storage thing.

Task 3: Configure run time permissions

9. Navigate to the Run Time permissions page for the Power Storage thing.

Instructor Note:

This is the students first time setting run time permissions. Be more verbose here.

10. Under All Properties, Events, and Services, in the Search Users or Groups field, click +.

11. Type/select Power. General.

12. Click the + again.

13. Type/select Shipyard AdminGrp.

14. Set the Property Read permission for both groups to Allow.

15. Click Save.

Task 4: View logged properties

Note: We have a logged property. Let's test that and make sure logging is working as expected. We'll set the value of Power Stored a few times to create log entries.

16. Navigate to the Properties and Alerts page of the Power Storage thing.

17. In the My Properties table, for the Power Stored property in the Value column, click to edit the value.

18. Type 11000 in the Set value of property field.

19. Click.

20. Click for the Power Stored property again.

21. Type 10000 in the Set value of property field.

22. Click.

23. Click for the Power Stored property a third time.

24. Type 10003 in the Set value of property field.

25. Click.

Note: We'll execute the Query Property History service to view the log.

Instructor Note:

Remind students that you view the log from a service on the thing. You can't view it from the value stream.

26. Select the thing's Services tab.

27. Click Generic at the bottom to expand the Generic Services section.

Instructor Note:

Remind students that Generic services come from the Generic Thing template, which is the root template for all things, so everything has them

28. Use the scroll bar to scroll down to the Query Property History service.

29. Click the Execute service button for the Query Property History service.

30. Click Execute in the pop-up window.

31. Click Done.

Instructor Note:

Mention that the change to 10003 wasn't logged because we set a data change threshold of 100.

Note: We are done with the power storage property. Now, let's create the power produced property on

the power producer thing shape.

Task 5: Create the Power Produced property for the power producers

32. Edit the PowerProducerTS thing shape.

33. Add a property with the following details:

Name	Base Type	Units	Min Value	Persistent	Logged
Power Produced	NUMBER	kW	0	True/Checked	True/Checked

34. Click Done.

35. Click Save.

36. Navigate to the Run Time permissions page for the PowerProducerTS thing shape.

Task 6: Set run time permissions

Note: We'll grant run time permissions to view properties to the Power. General group. The shipyard only needs to see properties as an aggregate, not individually, so we won't grant permission to the Shipyard AdminGrp.

37. Under All Properties, Events, and Services, in the Search Users or Groups field, click +.

38. Type/select Power.

39. Set the Property Read permission to Allow.

40. Click Save.

Task 7: Configure solar collectors to log changes to a value stream

Note: You can't set the value stream on a thing shape – only from a thing template or thing. So, we'll set

it on the solar collector thing template.

41. Edit the SolarCollectorTT thing template.

42. In the Value Stream field, type/select VS.

43. Click Save.

Task 8: Create the Power Consumed property for power consumers

44. Edit the PowerConsumerTS thing shape.

45. Add a property with the following details:

Name	Base Type	Units	Min Value	Persistent	Logged
Power Consumed	NUMBER	kW	0	True/Checked	True/Checked

46. Click Done.

47. Click Save.

Task 9: Set run time permissions

Note: The Power. General group should be able to view the property of every power producer, so we'll set both run time permissions and run time instance permissions.

48. Navigate to the Run Time permissions page.

49. Under All Properties, Events, and Services, in the Search Users or Groups field, click +.

50. Select Power. General.

51. Set the Property Read permission to Allow.

52. Click the Instance of Entity button to set visibility instance permissions.

53. Under All Properties, Events, and Services, in the Search Users or Groups field, click +.

54. Select Power. General.

55. Set the Property Read permission to Allow.

56. Click Save.

Task 10: Configure rovers to log to a value stream

57. Edit the Base Rover Template thing template.

58. In the Value Stream field, type/select VS.

59. Click Save.

Task 11: View logged properties

Note: Let's test logging for crew rover one.

60. Open the CrewRover1 thing.

61. Navigate to the Properties and Alerts.

62. For the Power Consumed property, in the Value column, click to edit the value.

Note: You may need to refresh your browser window to see the updated properties.

63. Type 5 in the Set value of property field.

64. Click.

65. Click for the Power Consumed property again.

66. Type 2 in the Set value of property field.

67. Click.

68. Select the Services tab.

69. Click Generic to expand the Generic Services section.

70. Scroll down to the Query Property History service.

71. Click the Execute service button for the Query Property History service.

72. Click Execute in the pop-up window.

73. After observing the data, click Done.

Exercise 3: Wrapped Services: Power System

In this exercise, you will write the two services Karen will need to get power consumer and power producer

data. These services don't belong to any specific thing but are system-wide utilities. So, before authoring us

properties, you will create an application services thing to store them.

Once you have your thing, you create the get power producers and get power consumers services, which will

each return a list of power assets in the system, along with their data. Of course, lastly, you set access control

for those services and test them.

Objectives:

After successfully completing this exercise, you will be able to:

• Create an application services thing.

• Author a service that returns an Info Table.

• Test a service.

• Create a data shape.

• Apply a data shape to an Info Table.

• Set run time access to properties.

• Use access control reports to verify run time access to services.

Task 1: Create the Power System Services thing

Note: Before we write the get power producers and consumers services, we need to decide which thing to put them in. This service doesn't belong to any one specific thing, so we will create a utility thing to store it.

Task 2: Author the Get Power Producers service

3. Navigate to the Services tab.

4. Click Add to add a service.

5. In the Service Info tab of the service editor, in the Name field, type Get Power Producers.

6. Select the Output tab of the service editor.

Note: This service returns structured data – more than one property of more than one power producer. Since the output of a service can only be one base

type, it needs to be one that can contain structured data. As a best practice, use the info table for this unless there is a compelling reason to use another base type.

7. From the drop-down list where NOTHING is selected, select INFOTABLE.

Instructor Note: Mention that when we return an Info Table as a result, it is a best practice to include a data shape to define the schema of the info table. Say we have a quick way to do that, which you will demo in a minute.

8. Select the Me/Entities tab of the service editor.

Instructor Note: Go through this slowly – using the me/entities tab to execute services on other things is a useful concept.

9. Select the Other Entity radio button.

10. In the Search Entities field, type/select PowerProducerTS.

11. In the Choose category drop-down list, select Queries.

12. Click Services to expand it.

13. Click the black arrow to the right of the Get Implementing Things with Data service name to add the snippet for the service.

14. Click Check syntax at the top of the Script editing area.

Task 3: Test the Get Power Producers service

Note: Next, we test the service and create a data shape for our result at the same time.

15. Click Done.

16. Click Save.

17. Click the Execute service button for the Get Power Producers service.

18. Click Execute.

Task 4: Assign a data shape to service output

19. Click + Data Shape on the upper-right side of the modal window to automatically create a data shape from this result.

20. In the Name field, type Get Power Producers DS.

21. Click Save.

22. Select the Field Definition stab to view the fields in this data shape.

Instructor Note: Do some talk here. This is the first time they created a data shape.

Note: Now that we have our data shape, we apply it to the info table.

23. Navigate to the Services page of the Power System Services thing.

24. Click the Get Power Producers service.

25. Click Output.

27. Click Done.

28. Click Save.

Task 5: Author the Get Power Consumers service

Note: It's nearly identical to Get Power Producers, so we'll use duplicate.

29. Select the Get Power Producers check box

30. Click Duplicate.

31. In the New Name field, type Get Power Consumers.

32. Click Done.

33. Click the Get Power Consumers service.

Note: We need to edit the script to use the power consumers thing shape, and we can't use the same data shape, since the power consumers thing shape won't return the same fields.

34. In the script area, replace Power Producer TSwith PowerConsumerTS.

35. Click Output.

36. In the Data Shape field, click the X to remove Get Power Producers DS.

37. Click Check syntax at the top of the Script editing area.

Note: And we create the new data shape.

38. Click Done.

39. Click Save.

Task 6: Test the Get Power Consumers service

40. For the Get Power Consumers service, click the Execute service button.

41. Click Execute.

Task 7: Assign a data shape to service output

42. Click + Data Shape on the right side of the modal window to automatically create a data shape from this result.

43. Type Get Power Consumers DS in the Name field.

44. Click Save.

45. Edit the Power System Services thing.

46. Select the Services tab.

47. Click the Get Power Consumers service.

48. Click Output.

49. In the Search Data Shape field, type/select Get Power Consumers DS.

50. Click Done.

51. Click Save.

Task 8: Set permissions to execute the services

Note: As a best practice, you should set permissions for services individually and not grant blanket service execute permissions for the entire thing. Doing so exposes dangerous services to the user that should only be part of the internal API, executed by the code that we write in a way that uses them safely and responsibly. We will give the power. General group permission to execute both services.

Instructor Note:

Emphasize this point. It's important. Really important.

52. Navigate to Run Time permissions.

53. In the Search Properties, Services, or Events field, search for Get Power Producers.

54. Expand Services.

55. Select Get Power Producers.

56. Under the Get Power Producers service, in the Search Users or Groups field, type/select Power. General.

57. Set the Service Execute permission to Allow.

58.In the Search Properties, Services, or Events field, select Get Power Consumers.

59. Under the Get Power Consumers service, in the Search Users or Groups field, type/select Power. General.

60. Set the Service Execute permission to Allow.

61. Click Save.

Task 9: Use access control reports to verify our users can execute the services

62. Click the Access Reports link.

63. In the Search Users or Groups field, type/select hconrad.

64. In the Search Entities field, type/select Power System Services.

65. Click Apply.

Note: The report correctly states that Hector was not given permission to execute any services from the Power System Services thing.

66. Remove hconrad from the User or Group field.

67. In the Search Users or Groups field, type/select Klee.

68. Click Apply.

Note: Karen does have permission to execute Get Power Producers and Get Power Consumers. Next, let's

see if she has permissions to execute the parent service from the internal API –

Get Implementing Things with Data.

69. Remove Power System Services from the Entity field.

70. In the Search Entities field, type/select Get Power Producers TS.

71. Click Apply.

Note: Karen has no run time permissions and cannot execute Get Implementing Things with Data.

Exercise 4: Create Aggregate Properties

In this exercise, you will create your aggregate properties for Hector in the Shipyard thing. These will contain the

total power produced, consumed, and stored for the entire shipyard, rather than just for a single power asset.

You will also log these properties in your value stream.

Objectives:

After successfully completing this exercise, you will be able to:

• Create properties.

• Set run time access for individual properties.

Task 1: Create aggregate properties for the Shipyard thing

1. Edit the Shipyard thing.

2. Add properties with the following details:

Name	Base Type	Units	Persistent	Logged
Power Consumed	NUMBER	kW	True/Checked	True/Checked
Power produced	NUMBER	kW	True/Checked	True/Checked
Power Stored	NUMBER	kWh	False/Unchecked	True/Checked

Instructor Note:

Power stored is not persistent because we are going to bind it to Power Stored on the Power

Storage thing. Since this property will always be in sync with the other, there is no need to retrieve it from the database.

3. Click Advanced Settings for the Power Stored property.

4. Verify that Value is selected in the Data Change Type field.

5. Type 100 in the Data Change Threshold field.

6. Click.

7. Click Save.

Task 2: Set permissions

Note: We are going to do something different for permissions on properties this time. Hector needs to be

able to read all properties of the Shipyard thing. Karen needs to read the power properties, but there will be other properties in the future coming from other departments, such as an operational efficiency property from the manufacturing department. Karen shouldn't have access to any new properties, so we will grant her permissions to individual properties, rather than blanket permissions to all properties. However, to see any properties, Karen first needs visibility to the Shipyard thing, which she doesn't currently have.

8. Navigate to the Visibility permissions page for the Shipyard thing.

9. Grant the MarsInterstellarShipyardOrg: Power organizational unit visibility.

10. Select the Run Time tab.

Note: Let's grant blanket property read to Shipyard AdminGrp.

11. Type sh in the Search Users or Groups field under All Properties, Events and Services and select

Shipyard AdminGrp.

12. Set the Property Read permission to Allow.

Note: Now we'll grant permissions to the Power. General group individually.

13. In the Search Properties, Services or Events field, search for Power Consumed.

14. Click Properties to expand it.

15. Select Power Consumed.

16. Under the Power Consumed service, in the Search Users or Groups field type/select Power. General.

17. Set the Property Read permission to Allow.

18. In the Search Properties, Services or Events field, type/select Power Produced.

19. Under the Power Produced service, in the Search Users or Groups field, type/select Power. General.

20. Set the Property Read permission to Allow.

21. In the Search Properties, Services or Events field, type/select Power Stored.

22. Under the Power Stored service, in the Search Users or Groups field, type/select Power. General.

23. Set the Property Read permission to Allow.

24. Permissions should look like this:

25. Click Save.

Note: The shipyard thing now has logged properties, so it needs a value stream specified.

Task 3: Set the value stream for the Shipyard

26. Navigate to the General Information page of the Shipyard thing.

27. In the Value Stream field, type/select VS.

28. Click Save.

Exercise 5: Binding Properties

Your shipyard has the aggregate properties, but they aren't being populated. In this exercise, you will populate

the power stored property. It's the easiest to populate because you only have one power storage thing.

You will to bind the power stored property in the power storage thing to the property in the shipyard thing. This

will keep them in sync.

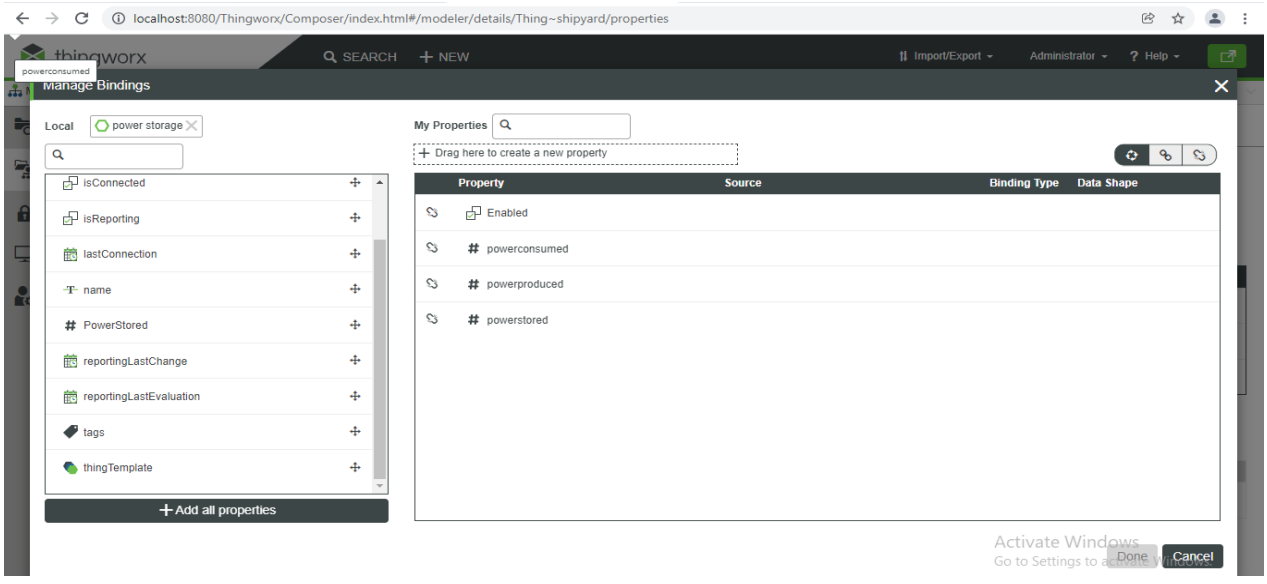
Objectives:

After successfully completing this exercise, you will be able to:

• Bind local properties.

Task 1: Bind the Power Stored property of the Power Storage thing to the Shipyard

1. Navigate to the Properties and Alerts page of the Shipyard thing.
2. Click Manage Bindings.



3. In the Search Things field, type/select Power Storage from the list.

4. Cursor over symbol of the Power Stored property and drag Power Stored property for the

Power Storage thing in the left panel to the Power Stored property for the Shipyard thing in the right

panel.

5. Click Done.

6. Click Save.

The aggregate power produced and power consumed properties are still not being populated. Unfortunately, you can't fix this with a local binding because there are many power producers and power consumers, and your shipyard is constantly adding and removing power assets. So, you will create a rollup service on the shipyard to do the job. It will gather up all the power assets, total them, and set the power consumed and power produced properties. It will also return an info table containing

the totals.

Objectives:

After successfully completing this exercise, you will be able to:

• Create a data shape manually.

• Author a service that manipulates Info Tables.

Task 1: Create a data shape for the rollup service

Note: Our rollup service will return three values – power produced, consumed, and stored. That means

we're going to need to return an info table, and that info table will need a data shape. So, we'll create

the data shape first. Unlike the last time, we need to create this data shape manually, since there is no

other service that returns exactly the fields we need.

1. Create a data shape entity with the following details:
2. Select the Field Definitions tab.
3. Click Add to add the field definition.

4. Type power Produced in the Name field.

5. Type/select NUMBER for the Base Type field.

6. Type kW in the Units field.

7. Create two more field definitions with the following details:

Name	Base Type	Units
Power Consumed	NUMBER	kW
Power Stored	NUMBER	kWh

Note: Now that we have our data shape, we can author the rollup service.

Task 2: Author the rollup service

10. Navigate to the Services page of the Shipyard thing.

11. Click Add.

12. Type Rollup in the Name field.

13. Click Output.

14. From the drop-down list where NOTHING is selected, select INFOTABLE.

15. In the Search Data Shape field, type/select Rollup DS.

16. Type the following code in the Script editing area.

Instructor Note:

This code will be saved in the following folder on the virtual machine:

D:\Student\Code Snippets\

The file name is:

Model_M6_Ex06_Shipyard_Rollup.txt

This is also a good time to go through the code walkthrough slides if you want to. Many students are

curious about this, and you are welcome to do so, but it's optional. A code walkthrough is not

included in the WBT.

```
/*************************************************************
******

Log the beginning of the service call

*************************************************************
*****

logger. Warn ("Beginning Shipyard Rollup Service.");

/*************************************************************
******

Using our GetPowerConsumers () service, get all power

consumers and store them in an info Table

*************************************************************
*****/

var sysServThingName = "PowerSystemServices";
```

```
var power Consumers = Things[sysServThingName]. GetPowerConsumers ();
var total Power Consumed = 0;
/*************************************************************
******
Iterate through the table one row at a time, adding to
totalPowerConsumed with each row. This code was derived
from Snippets > Info table for loop
*************************************************************
*****/
var table Length = powerConsumers.rows. length;
for (var x = 0; x < table Length; x++) {
var row = powerConsumers.rows[x];
totalPowerConsumed = totalPowerConsumed + row. power Consumed;
}
//Set the Shipyard Power Consumed property
me. power Consumed = totalPowerConsumed;
/*************************************************************
******
Using our GetPowerProducers () service, get all power
consumers and store them in an info Table
*************************************************************
*****/
var power Producers = Things[sysServThingName]. GetPowerProducers ();
```

```
/**************************************************************
******

Use the Aggregate function, available from

Snippets > InfoTableFunctions > Aggregate

to sum the column of the totalPowerProduced

Info Table.

**************************************************************
*****/

var prams = {

t: power Producers /* INFOTABLE */,

columns: "Power Produced" /* STRING */,

aggregates: "SUM" /* STRING */,

group by Columns: undefined /* STRING */

};

var result Table = Resources["InfoTableFunctions"]. Aggregate(prams);

var totalPowerProduced = resultTable.SUM_PowerProduced;

//Set the Shipyard Power Produced property

me. Power Produced = totalPowerProduced;

/**************************************************************
******

Return info table of power produced/consumed/stored

Used Snippets > Create Info table from Data shape and Snippets > Create

info table
```

entry from data shape as a basis for this code.

****
*****/

```
prams = {
infoTableName: "Info Table",
dataShapeName: "Rollups"
};
var result =
Resources["InfoTableFunctions"].CreateInfoTableFromDataShape(params);
var new Entry = new Object ();
new Entry. power Consumed = totalPowerConsumed; // NUMBER
newEntry.powerProduced = totalPowerProduced; // NUMBER
var powStorageThingname = "Power Storage";
newEntry.powerStored = Things[powStorageThingname]. Power Stored; // NUMBER
result. Add Row (new Entry);
logger. Warn ("Shipyard Rollup: " + me. power Consumed + "C-" +
 empower Produced + "P-" + me. Power Stored + "S");
```

17. Click Check Syntax at the top of the Script editing area.

18. Click Done.

19. Click Save.

Task 3: Test the rollup service

Note: Normally, you test a service immediately after writing it. But first, let's look at our power assets so

we know what the result of our rollup service should be.

20. Navigate to the Services tab of the PowerSystemServices thing.

21. In the GetPowerProducers row, click the Execute Service button.

22. Click Execute.

23. Click Done.

Note: Let's set each of the solar collectors to produce 1000 kW, for a total of 2000 kW for the shipyard, and power produced to 10003 kWh.

24. Navigate to the Properties and Alerts tab of SolarCollector-1.

25. In the Power Produced property row, click.

26. Edit the Set value of property field to 1000.

27. Click.

28. Set or verify the following property values:

Entity	Property	Property Value
SolarCollector-2	Power Produced	1000 kW
Power Storage	Power Stored	10003 kWh

Note: Earlier, you set power consumed for CrewRover1 to 2 kW. Let's verify that's still the case.

29. Navigate to the Services tab of the PowerSystemServices thing.

30. For the GetPowerConsumers service, click the Execute Service button.

31. Click Execute.

32. Verify that at least one of your power producers is consuming more than 0 kW of power. If not, set a

property so at least one is producing some power.

33. Add up and remember the total power produced for the shipyard. In the example above, that would be

2 kW.

34. Click Done.

Note: We know our shipyard is producing 2000 kW, consuming 2 kW, and has 10,003 kWh stored. Now

let's make sure the Rollup service reflects that.

35. Navigate to the Services page of the Shipyard thing.

36. For the Rollup service, click the Execute Service button.

37. Click Execute.

Note: The result info table is correct. However, our service also sets properties on the shipyard. Let's

verify that worked as well.

38. Click Done.

39. Select the Properties and Alerts tab.

40. Verify that the Power Consumed property is set to 2, Power Produced property is set to 2000, and Power Stored property is set to 10003. If the property values are not reflecting, then click the Refresh button to verify.

Exercise 7: Subscribing to a Timer

Your rollup service can update the power consumed and power produced properties, but there is currently no mechanism to execute the service. You will subscribe to the shipyard's timer event, so the rollup service will update every ten seconds, assuring that these properties are never more than ten seconds old.

You are also following your best practice regarding timers, where the Shipyard Thing is a timer, listening to its

own timer event. This keeps your event queue clean and empty.

Objectives:

After successfully completing this exercise, you will be able to:

• Write a subscription.

• Configure a timer.

Task 1: Configure the timer event to occur every ten seconds

1. Navigate to the Configuration tab of the Shipyard thing.

2. Change the value of the Update Rate field to 10000.

3. Verify the enabled check box is selected.

Task 2: Author a subscription to the timer event

4. Select the Subscriptions tab.

5. Click Add to add your subscription.

6. In the Subscription Info tab of the subscription editor:

• Verify that Me is selected.

• Select the Enabled check box.

7. Select Inputs.

8. In the Event drop-down list, select Timer.

9. Type the following code in the Script editing area. This code calls the Rollup service.

var result = me. Rollup ();

10. Click Done.

11. Click Save.

Note: To test this subscription, we will change a few power properties and verify that the shipyard updated appropriately.

Task 3: Test the subscription

12.Set the following property values:

Thing Name	Property	Property Value
Power Storage	Power Stored	12000 kWh
SolarCollector-1	Power Produced	300 kW

In the last exercise, you wrote a long service. In this exercise, you will use script logs to troubleshoot that service.

Objectives:

After successfully completing this exercise, you will be able to:

• Read ThingWorx logs.

• Configure a log.

• Filter a log.

• Use logs to troubleshoot scripts.

Task 1: View the script log

1. Select the Monitoring tab.

Instructor Note:

You may want to go slowly and add content to this exercise. The ability to troubleshoot logs and your services are important, and this is the only time during the specialization that it is covered.

2. Select Script Log.

Note: The Rollup service has two log output lines. Since this service is executing every ten seconds, it adds to the log frequently. This is actually very poor design, as logging does consume system resources. However, logs are helpful when troubleshooting issues with the system or your scripts.

Task 2: Examine how to configure a log

3. Click Configure.

4. Expand the Log Level drop-down list.

Note: Setting a log level here will prevent any log entries below that level from being inserted into the script log. Any such entries are never recorded; they are permanently lost. So, if our rollup script had debug logs while this is set to warn, those debug log messages would never be recorded. We won't change this.

5. Click Cancel.

Task 3: Filter the script log

6. Click the Filter Options icon.

7. Click + in the Search Users field and select Administrator. This prevents log entries from other users from appearing in the table.

8. Select Error in the first Level Range drop-down list.

9. Click Apply.

Task 4: Create an error entry in the script log

Note: When we filter the log instead of configuring it, we change the log entries that are shown in the table. However, we don't change what is being stored in

the actual log file. In this case, warn level logs are still being entered into the log and can be retrieved by simply changing the filter back. Let's create an error.

10. Navigate to the Services tab of the Shipyard thing.

11. Click the Rollup service.

12. Add the following as the last line of code in the service, which will generate an error because the referenced thing doesn't exist: Things["NonexistantThing"]. RunFakeService ();

Note: Our rollup service will still work because the error is on the last line. All the useful code has executed before this line is interpreted.

13. Click Done.

14. Click Save.

15. For the Rollup service, click the Execute Service button.

16. Click Execute.

17. Click Done.

18. Navigate back to the Script Log.

19. Click the Refresh button to refresh the script log. The error is being emitted every ten seconds.

Task 5: Set the service and log filter back to their original state

20. Click the Filter Options icon.

21. In the first Level Range drop-down list, select All.

22. Click Apply.

Note: We see three log entries every ten seconds, but we don't see entries below WARN level because our system is configured to ignore them. Let's fix our Rollup service.

23. Navigate to the Services tab of the Shipyard thing.

24. Click the Rollup service.

25. Remove this last line of code causing the error:

Things["NonexistantThing"]. RunFakeService ();

26. Click Done.

27. Click Save.

Exercise 9: Automate Use Case: Create Properties

Control is accomplished primarily through services, but you need a few more properties to support yours automate and control use cases.

In this exercise, you will create a Boolean on the shipyard thing to indicate when the system is in a low- power emergency state.

Karen also needs the ability to prevent the system from entering a low-power emergency until a certain time. You will create a date/time property to store when that certain time is.

Lastly, in a low-power emergency, non-critical power consumers shut down. You need a property to indicate which power producers are critical. You will add a Boolean property to the power

consumer thing shape to do that. That way, every power consumer will have a Boolean indicating whether it is critical or not.

Objectives:

After successfully completing this exercise, you will be able to:

•Create properties.

•Set run time permissions.

Task 1: Create the properties

1.Create the following properties:

Instructor Note:

Go through this quickly. Creating properties is not a challenge for the students at this point

Thing Name	Property Name	Base Type	Persistent	Logged
Shipyard	LowPowerState	BOOLEAN	True/Checked	True/Checked
Shipyard	NoLowPowerStateUntil	DATETIME	True/Checked	True/Checked
PowerConsumerTS	High Priority	BOOLEAN	True/Checked	True/Checked

Task 2: Set the permissions

Note: Let's consider access control on these properties. Our shipyard administrator already has run time read on the shipyard, so he has access. Karen is a more difficult case. She needs to be able to set all these properties, but we can't give her blanket control of all properties in the Shipyard. It will have other properties in the future that are unrelated to Karen's job, such as supply and logistics information.

2.Allow the following permissions:

Instructor Note:

Make sure to select Instance, or you won't find the High Priority property.

Instructor Note:

Power. General already has run time instance property read on all PowerConsumerTS properties from an earlier exercise, so we don't need to add it here.

Exercise 10: Writing Services

In this exercise, you will create four services to support your use cases.

First, you need to be able to control power consumers by shutting them down. But, each power consumer is different, and responds to the shutdown command a different way. For example, a lamp would just shut off, but a large robot arm would have a lengthy shutdown procedure with safety checks to make sure it didn't drop anything. So, you'll create an over ridable service on the power consumer thing shape. This makes sure every power consumer has a shutdown service, but the specific code for that service will be overridden by the thing shape or thing template.

Then, you will override the shutdown service in the base rover thing template. Rovers respond to the shutdown event in an unusual way. Instead of shutting down, they go to a charging station and use the remaining power in their battery to feed the power grid, going into power producing mode.

Next, you will create a service that calculates the number of hours' power will remain on under current conditions. Then, you'll create two setter services, one to exit a low power state emergency, and another to enter one.

Objectives:

After successfully completing this exercise, you will be able to:

•Write an over ridable service.

•Override a service.

•Author setter services.

Task 1: Author the over ridable shutdown service

1.Edit PowerConsumerTS.

2.Navigate to the Services tab.

3.Click Add and select Local (JavaScript).

4.Type Shutdown in the Name field.

5.In the Description field, type This is a generic service intended to be overridden by the thing template or thing that implements the PowerConsumerTS thing shape. It is designed to shut down the power in a safe way to conserve power.

6.Select the Allow Override checkbox.

7.Select the Sync checkbox.

Note: This is an asynchronous service; therefore, it requires no input and there is no output. Also, it does not require any code in the script area. We can't test it directly, since it has no code to test.

8.Click Done.

9.Click Save.

Note: Next, we'll write the shutdown service for rovers, placing it on the base rover template. This service will send the rover to the charging station, so it can use the remaining power in its battery to supplement the power grid.

Task 2: Override the shutdown service in the Base Rover Thing Template

10.Edit the Base Rover Template thing template.

11.Navigate to the Services tab.

12.For the Shutdown service, click the Override icon.

13.Select the Me/Entities tab.

14.Click to expand Properties.

15.Click the black arrow to the right of the Destination property.

16.Alter the end of the code so it reads:

me. Destination = "Charging Station";

17.Click Done.

18.Click Save.

Task 3: Test the override service on a rover

Note: Let's test this service on crew rover one.

19.Edit the CrewRover1 thing.

20.Navigate to the Services tab.

21.Click Execute Service for the Shutdown service. If necessary, click Save to see the Shutdown service.

22.Click Execute.

Note: There is never a result from an asynchronous service, so this no result is normal.

23.Click Done.

24.Select the Properties and Alerts tab.

25.Notice the Destination property under CrewRoverTemplate is set to Charging Station. If not, click. Refresh to reflect the Destination property values.

Task 4: Reset the rover location

Note: We'll change the destination because if it is Charging Station, we won't know if a shutdown ran successfully.

26.For the Destination property, click Set value of property .

27.Delete Charging Station and type Something Else in the Set value of property field.

28.Click

Task 5: Write the estimated time to power outage service

29.Edit the PowerSystemServices thing.

30.Navigate to the Services tab.

31.Click Add.

32.Type EstHrsToPowerOutage in the Name field.

33.Click Output.

34.From the drop-down list where NOTHING is selected, select NUMBER.

35.Type the following code in the Script editing area:

```
/*****************************************************************
********* Get the shipyard name
*****************************************************************
********/
 var shipyardEntityName = "Shipyard";
/*****************************************************************
*       Start by doing rollup to get up-to-the-second data
*****************************************************************
********/ Things[shipyardEntityName]. Rollup ();
/*****************************************************************
*********
*       Calculate hours to dead battery.
*       power Deficit is (power Consumed - Power Produced)
*       Power Stored / power Deficit = hrs to dead battery
*****************************************************************
********/
varpowerDeficit=Things[shipyardEntityName].powerConsumed -
Things[shipyardEntityName]. Power Produced;
var hrsToPowerOut;
if (power Deficit <= 0) {//Set to 10,000 if there is no power Deficit.
hrsToPowerOut = 10000;
```

}

else {// Calculate hours if deficit is positive

hrsToPowerOut = Things[shipyardEntityName]. Power Stored / power Deficit;

}

result = hrsToPowerOut;

36.Click Check syntax at the top of the Script editing area.

37.Click Done.

38.Click Save.

Task 6: Test the estimated time to power outage service

Note: Before we test this service, let's set properties so we know the result. We'll set the system so there is 12 hours of remaining power.

39.Set or verify the following property values:

Thing Name	Property	Property Value
Power Storage	Power Stored	12000 kWh
CrewRover1	Power Consumed	2000 kW
CrewRover2	Power Consumed	0 kW
CrewRover3	power Consumed	0 kW
SolarCollector-1	Power Produced	1000 kW
SolarCollector-2	Power Produced	0 kW

Note: 12,000 kWh stored / (1000 kW produced – 2000 kW consumed) = 12 hours

40.Open PowerSystemServices.

41.Navigate to the Services tab.

42.For the EstHrsToPowerOutage service, click the Execute Service button.

43.Click Execute.

Note: We should have 12 hours of energy remaining.

44.Click Done.

Task 7: Write the service to exit a low power emergency state

45.Click Add.

46.In the Name field, type Exit Low Power State.

Note: This service will take a number as input. This number is the number of hours before a new low power emergency cannot occur. This way, Karen can prevent the automated system from re-starting the low power emergency.

47.Click Inputs.

48.Click Add.

49.Type hrsNoLowPowerState in the Name field.

50.Select NUMBER in the Base Type field.

51.Click Done.

52.Click Output.

53.Select STRING from the drop-down list where NOTHING is selected.

54.Type the following code in the Script editing area:

```
/**************************************************************
****** Get the shipyard entity name
**************************************************************
*****/ var shipyardEntityName = "Shipyard";

/**************************************************************
******
```

*If system is not in low power state, don't change state

*and begin creation of result message

****
*****/

var message;

if (Things[shipyardEntityName].LowPowerState == false) {message = "Attempted to exit low power state while " +

"shipyard is not in a low power state. ";

}

else {

//Otherwise, set low power state to false and begin creation of result message

Things[shipyardEntityName]. Low Power State = false; message = "System removed from low power state. ";

}

/**********
******

*Set hrsNoLowPowerState to current date/time + input

*parameter. Complete message

****
*****/ var temp Date = Date. Now ();

Temp Date= dateAddHours (tempDate, hrsNoLowPowerState); Things[shipyardEntityName]. NoLowPowerStateUntil = tempDate; message = message +

" System will not enter low power state again before " + Things[shipyardEntityName]. NoLowPowerStateUntil;

```
/**************************************************************
******

*       Send message to result string and script log.

**************************************************************
*****/ logger. Warn(message);

result = message;
```

55. Click Check syntax at the top of the Script editing area.

Task 8: Test the exit low power state service

Note: We can test the service directly from this page.

56.Below the script window, in the # hrsNoLowPowerState field, type 0.

57.Click Save and Execute.

Note: Notice the date and time listed in the result message. It should match the property in the Shipyard thing. Let's make sure the shipyard is now in a low power state. Now let's set the shipyard to a low power emergency state and try again.

58.Open the Shipyard thing.

59.Select the Properties and Alerts tab.

60.Verify that the LowPowerState property is false and the NoLowPowerStateUntil property matches the results of the service execution. If not, then click Refresh and then verify.

61.For the LowPowerState property, click the Set value of property icon.

62.Select True in the Set value of property field.

63.Click .

64.Open the PowerSystemServices thing.

65.Navigate to the Services tab.

66.If necessary, click Cancel to view all the services instead of just the ExitLowPowerState service.

67.For the ExitLowPowerState service, click the Execute Serviceicon.

68.Type 0 in the hrsNoLowPowerState input parameter.

69.Click Execute.

Note: Notice the date and time listed in the result message. It should match the property in the Shipyard thing.

70.Click Done.

71.Open the Shipyard thing.

72.Select the Properties and Alerts tab.

73.Verify that the LowPowerState property is false and the NoLowPowerStateUntil property matches the results of the service execution. If not, then click Refresh and then verify.

Task 9: Write the service to enter a low power emergency state

74.Open the PowerSystemServices thing

75.Navigate to the Services tab.

76.Click Add.

77.Type EnterLowPowerState in the Name field.

78.Click Output.

79.From the drop-down list where NOTHING is selected, select STRING.

80.Type the following code in the Script editing area:

/**********
****** Get the entity names

****
*****/ var shipyardEntityName = "Shipyard";

/**********
******

* If already in low power state, do nothing

* except set result message

****
*****/ var now Date = Date. Now ();

if (me. LowPowerState) {

message = "Attempted to enter low power state while already " + "in low power state. No action taken.";

}

else {

//if date is later than NoLowPowerState, set low power state to true

//Set result message

if (now Date > Things[shipyardEntityName]. NoLowPowerStateUntil) {Things[shipyardEntityName].LowPowerState=true;

message = "Entered low power state";

}

else {

//else, don't enter low power state because doing so is blocked by NoLowPowerStateUntil

//Set result message.

message = "Cannot enter low power state until " + Things[shipyardEntityName]. NoLowPowerStateUntil + "Try again later or reset Shipyard.";

```
}

}

/*****************************************************************
******

*       send message to result string and script log

*****************************************************************
*****/ logger. Warn(message);

result = message;
```

81.Click Check syntax at the top of the Script editing area.

Task 10: Test the enter low power state service

82.Click the Save and Execute button to test the service.

83.Click Execute.

Note: We get an entered low power state message. Let's exit the low power state and set the system so it can't re-enter the low power state for one hour.

84.Click Done.

85.For the ExitLowPowerState service, click the Execute Serviceicon.

86.Type 1 in the hrsNoLowPowerState input parameter.

87.Click Execute.

88.Click Done.

Note: Now let's try to re-enter the low power state and verify that we can't.

89.For the EnterLowPowerState service, click the Execute Service icon.

90.Click Execute.

Note: We get a message that we can't enter a low power state. Next, we'll run the ExitLowPowerState service again to reset the NoLowPowerStateUntil property.

91.Click Done.

92.For the ExitLowPowerState service, click the Execute Serviceicon.

93.Type 0 in the hrsNoLowPowerState input parameter.

94.Click Execute.

Note: The result should be a message saying that you attempted to exit low power state while shipyard is not in a low power state.

95.Click Done.

96.Click Save.

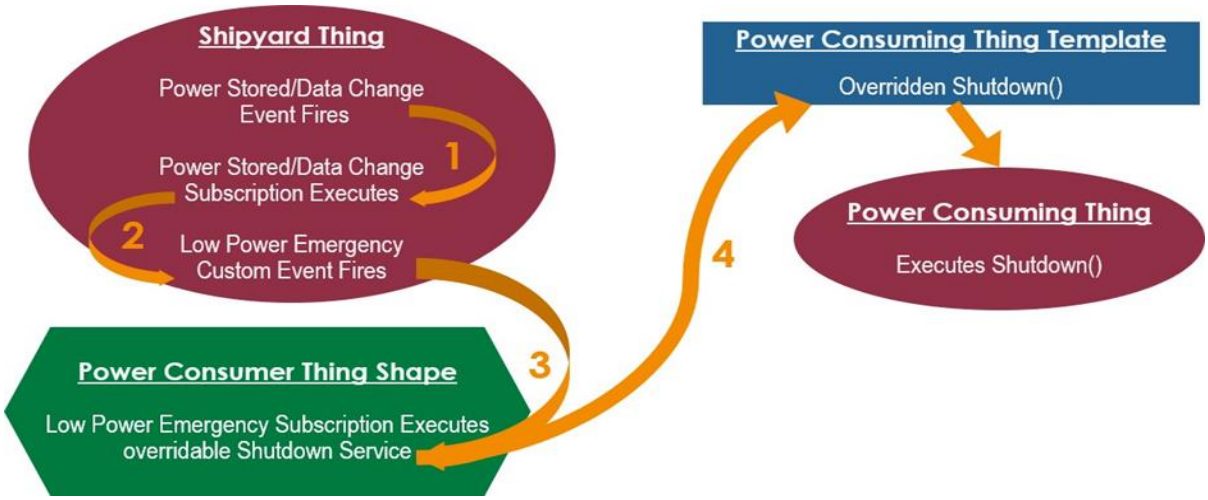
Task 11: Set run time permissions on the services

97.Set the following permissions:

Thing Name	Service	Permission Type	Groups
PowerSystemServices	EstHrsToPowerOutage	Run Time Service Execute	Power. General & ShipyardAdminGrp
PowerSystemServices	EnterLowPowerState	Run Time Service Execute	Power. Admin
PowerSystemServices	ExitLowPowerState	Run Time Service Execute	Power. Admin

Exercise 11: Events and Subscriptions

In this exercise, you will configure the events and subscriptions needed to automate your use case. Under the proper conditions, the shipyard should go into a low power state automatically. More importantly, it shouldn't go into a low power state when it's inappropriate to do so.



This process will start with the data change event on the power stored property. There can't be a power emergency unless the power reserves are being depleted, which is a change in the amount of power stored.

But, not every change in power stored is a power emergency. Most are not. You will create a subscription which analyses the data change and determines if issuing a low power emergency is warranted. If so, it emits a custom low power emergency event.

Then, a second subscription listens for that event. This subscription will shut down any non-critical power consumer.

Objectives:

After successfully completing this exercise, you will be able to:

•Write a subscription on a data change event.

•Write a custom event.

•Create a data shape for a custom event.

•Fire a custom event using subscription code.

Task 1: Create a data shape for a low power emergency custom event

Note: We need a low power emergency custom event, and every event needs a data shape to define the data that comes with it. We'll start by creating that data shape. It has four fields – The estimated time to power outage, power produced, consumed, and stored. We already have the RollupDS which has three of these fields, so we'll start by duplicating that.

1.Duplicate RollupDS.

2.In the Name field, type Low Power Emergency DS.

3.Select the Field Definitions tab.

4.Click Add to add the field definition.

5.In the Name field, type estHrsToPowerOut.

6.Select NUMBER for the Base Type field.

7.Click.

8.Click Save.

Task 2: Create a low power emergency custom event

9.Open the Shipyard thing.

10.Navigate to the Events tab.

11.Click Add to add an event.

12.In the Name field, type Low Power Emergency.

13.In the Data Shape field, type/select Low Power Emergency DS.

14.Click .

15.Click Save.

Task 3: Author a subscription that fires our custom low power emergency event

Note: Our custom event exists, but nothing is subscribed to it. Now we will start implementing the chain of events and subscriptions that cause a low power emergency, starting with the data change event on power stored.

16.Select the Subscriptions tab.

17.Click Add to add your subscription.

18.Verify that Me is selected.

19.Select the Enabled checkbox.

20.Click Inputs.

21.In the Event list, select Data Change.

22.In the Property list, select Power Stored.

23.Type the following code in the Script editing area:

```
/**************************************************************
****** Get the entity name
**************************************************************
*****/ var

Sys Serve Entity Name = "PowerSystemServices";

//Get hours to power outage and current time

var hrsToPowerOutage = Things[sysServEntityName]. EstHrsToPowerOutage ();
var now Date = Date. Now ();
```

```
/*************************************************************
*       We only act if there are less than 12
*       hours to power outrage and the system
*       can enter a lower power state according to the
*       NoLowerPowerStateUntil DATETIME parameter
*       Otherwise, this subscription does nothing.
*************************************************************
*****/
if ((hrsToPowerOutage < 12) &&
(now Date > me. NoLowPowerStateUntil)) {
//Emit the LowPowerEmergency event var prams = {
powerConsumed: me. PowerConsumed, power Stored: me. PowerStored, power Produced: me.Power Produced, estHrsToPowerOut: hrsToPowerOutage
};
me. LowPowerEmergency(params);
//Enter LowPowerState Things[sysServEntityName].EnterLowPowerState();
}
```

This code calls estHrsToPowerOutage. If the answer is less than 12 hours, it emits the LowPowerEmergency event and executes the EnterLowPowerStateservice.

24.Click Check Syntax at the top of the Script editing area.

25.Click Done.

26.Click Save.

Task 4: Test the subscription on the data change event

Note: Let's see if this works. We'll trigger the data change event by changing PowerStored. Let's put the shipyard in a state this is not a power emergency. We'll also make Crew Rover 2 a critical power producer.

27.Set or verify the following property values:

Note: You can validate power producing and consuming properties faster using the GetPowerProducers and GetPowerConsumers services than going to each entity individually in Composer. Entries in bold should only need to be validated – you set them to these values in earlier exercises.

Thing Name	Property	Property Value
Power Storage	PowerStored	13000 kWh
CrewRover1	**Power Consumed**	**2000 kW**
CrewRover2	**Power Consumed**	**0 kW**
CrewRover2	High Priority	True/Checked
CrewRover3	**Power Consumed**	**0 kW**
SolarCollector-1	**Power Produced**	**1000kW**
SolarCollector-2	**Power Produced**	**0 kW**
Shipyard	NoLowPowerStateUntil	Any time not in the future
Shipyard	**LowPowerState**	**False/Unchecked**

28.Open the PowerSystemServices thing.

29.Navigate to the Services tab.

30.For the EstHrsToPowerOutage service, click the Execute Service icon.

31.Click Execute.

Note: You will get 13 in the result. That's more than 12 hours, so we aren't in a power emergency. Let's

change power stored to 10,000, resulting in 10 hours or power left and a power emergency.

32.Click Done.

33.Set the PowerStored property for the Power Storage thing to 10,000.

34.Open the Shipyard thing.

35.Navigate to the Properties and Alerts tab.

36.Under the My Properties table, verify that the LowPowerState property value is now true (LowPowerState property check box is selected). If not, then click Refresh and then verify.

Task 5: Write the subscription that shuts down power consumers in a low power emergency

Note: The LowPowerState property was set to true by our subscription. But, non-critical power consumers didn't shut down. We'll implement that next with a subscription to the LowPowerEmergency event.

37.Open the Power ConsumersTS thing shape.

38.Navigate to the Subscriptions tab.

39.Click Add.

40.Select the Other Entity radio button.

41.In the Search Entities field, type/select Shipyard.

42.Select the Enabled check box.

43.Click Inputs.

44.From the Event drop-down, select LowPowerEmergency.

45.Type the following code to the Script area. This code calls the over ridable shutdown service for the power consumer, so a large robot can have a lengthy shutdown procedure, while our rovers can go to a charging station:

```
if (me. High Priority == false) {me. Shutdown ();
}
```

46.Click Check Syntax at the top of the Script editing area.

47.Click Done.

48.Click Save.

Task 4: Test the subscription chain

Note: Next, we'll get out of the low power state and try again. This time, our consumers should shut down.

49.In the Power Storage thing, set the Power Stored property to 13000.

50.Open the PowerSystemServices thing.

51.Navigate to the Services tab.

52.For the ExitLowPowerState service, click the Execute Service button in the Execute column.

53.Type 0 in the hrsNoLowPowerState field.

54.Click Execute.

Note: You should get a message that the System is removed from low power state.

55.Click Done.

56.In the Shipyard thing, verify that the LowPowerState property is false/unchecked.

Note: The low power state is over. Let's trigger it again.

57.In the Power Storage thing, set the Power Stored property to 10000

58.In CrewRover1, verify that the Destination property is Charging Station.

Note: This means that the CrewRover1 is headed to the Charging Station – our automation works!

59.In CrewRover2, verify that the Destination property is not Charging Station.

Note: We tagged Crew Rover 2 as a high priority power consumer that isn't shut down in a power emergency. It will do whatever it was doing before.

Wrap-Up Intro

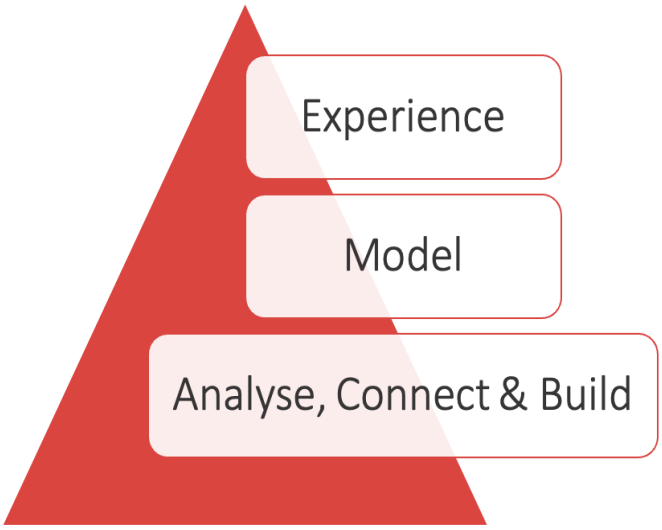
Wrap-Up Consolidated

In this module, we consolidate and practice our knowledge through the use of a new use case:

- Practice working through the ThingWorx Modelling Framework and sub-frameworks.
- Gain experience creating the entities and functionality needed to create a scalable ThingWorx model grounded in best practices.

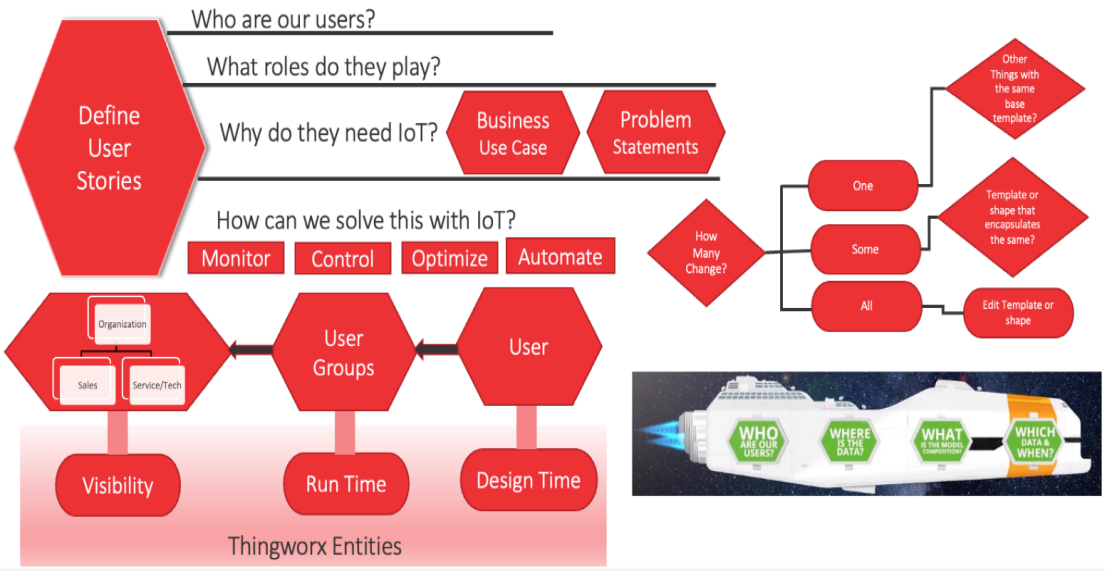
Overview of Experience

- Always Start with Experience



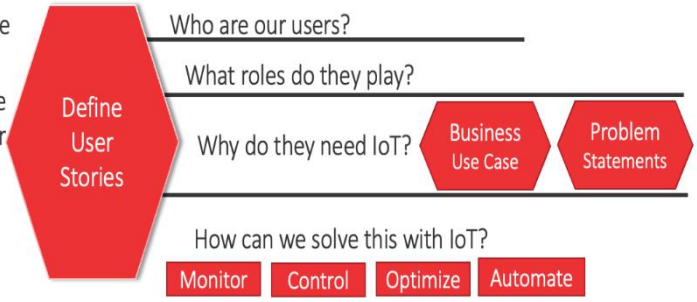
- With less human intervention, robots assemble starships.
- A technician is required to maintain or repair robot arms and other shipyard assets on a regular basis.
- Maintaining the shipyard's efficiency requires prioritising and assigning maintenance repair activities.

Closing Project – Robot Maintenance

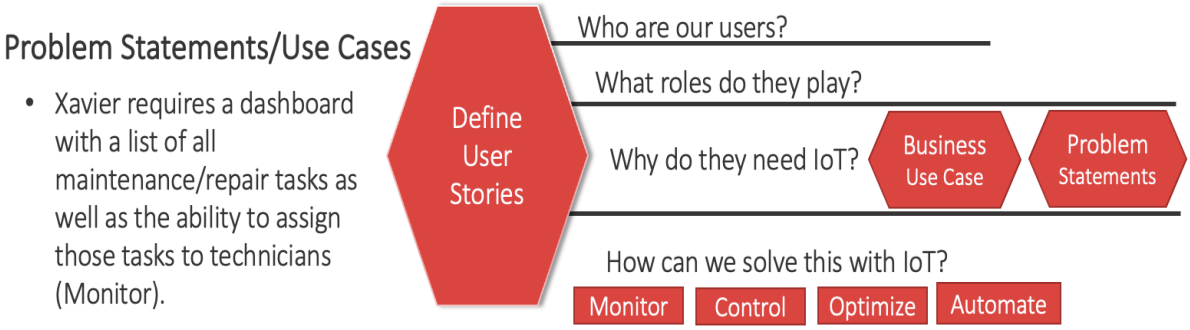


Who are our users?

- Who are our users?
  - Xavier, the maintenance lead. – Technicians
- What roles do they play?
  - Xavier is responsible for shipyard maintenance – prioritizing, assigning, and completing all maintenance and repair tasks.
  - Technicians are responsible for maintenance and repair tasks assigned to them.



Why do they need IoT?



- Robots are not connected to the ThingWorx-based IoT system.
- We have no idea if an asset needs to be maintained or repaired. This necessitates the tracking of every operable component (Monitor).
- Advanced assets can recognize when they require maintenance and repair and can generate work orders on their own (Automate).

How would you implement?

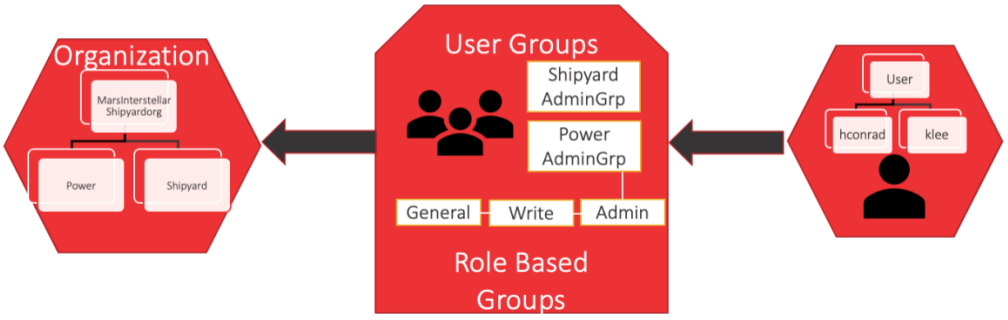
Organizational Unit: Maintenance, under MarsInterstellarShipyardOrg

Logical Groups: Maintenance Lead, Maintenance Technician

Role-Based Groups: Maintenance. Admin (Maintenance lead is a member), Maintenance. General (Maintenance Tech group is a member)

Users: Xavier (member of Maintenance Lead), Test Technician (member of Maintenance Tech)

- How would you expand our access control model for this use case?



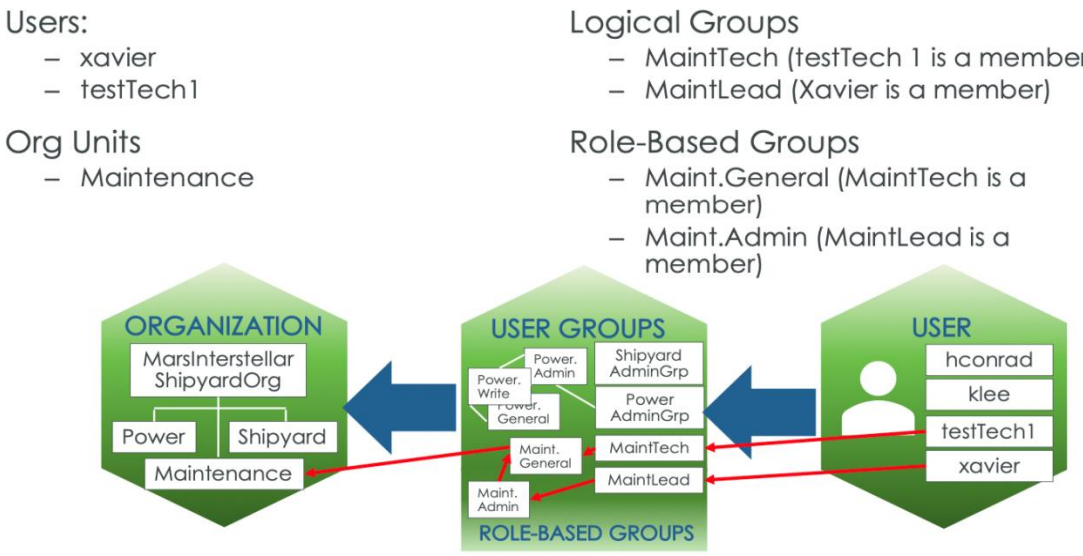
Implement the user access entities

Organizational Unit: Maintenance, under MarsInterstellarShipyardOrg

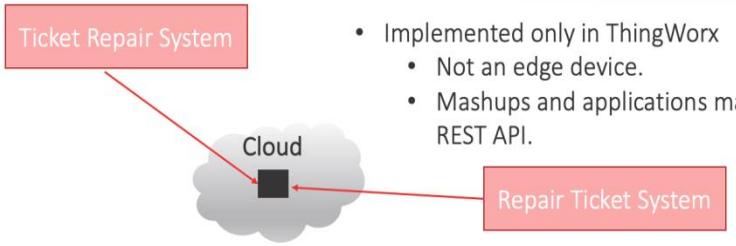
Logical Groups: Maintenance Lead, Maintenance Technician

Role-Based Groups: Maintenance. Admin (Maintenance lead is a member), Maintenance. General (Maintenance Tech group is a member)

Users: Xavier (member of Maintenance Lead), Test Technician (member of Maintenance Tech)



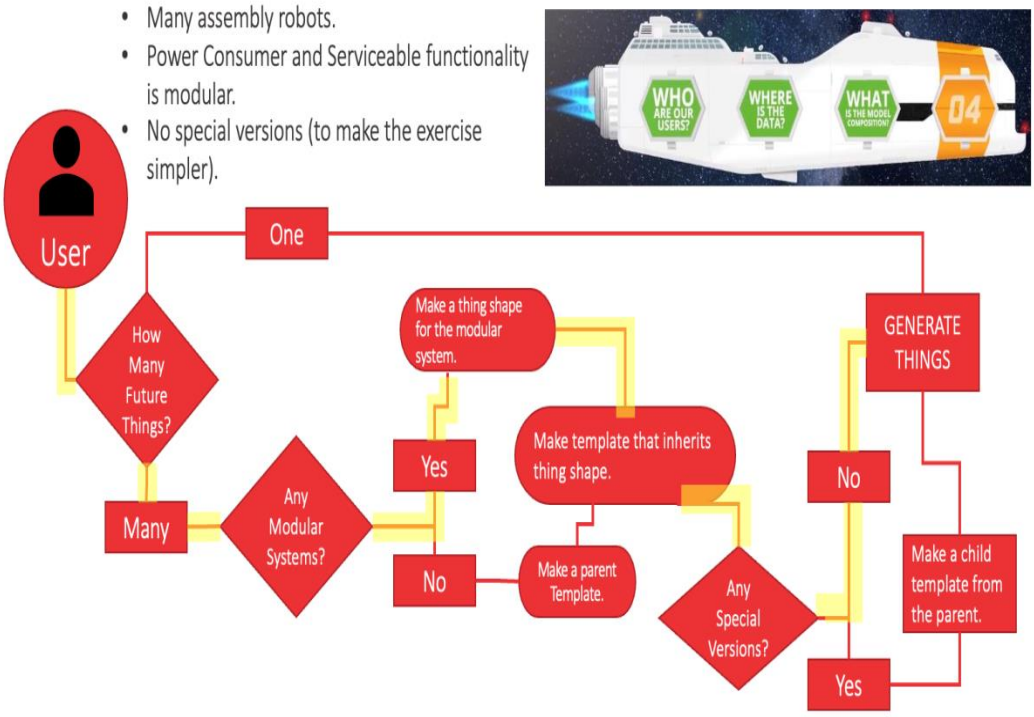
Repair Ticket System



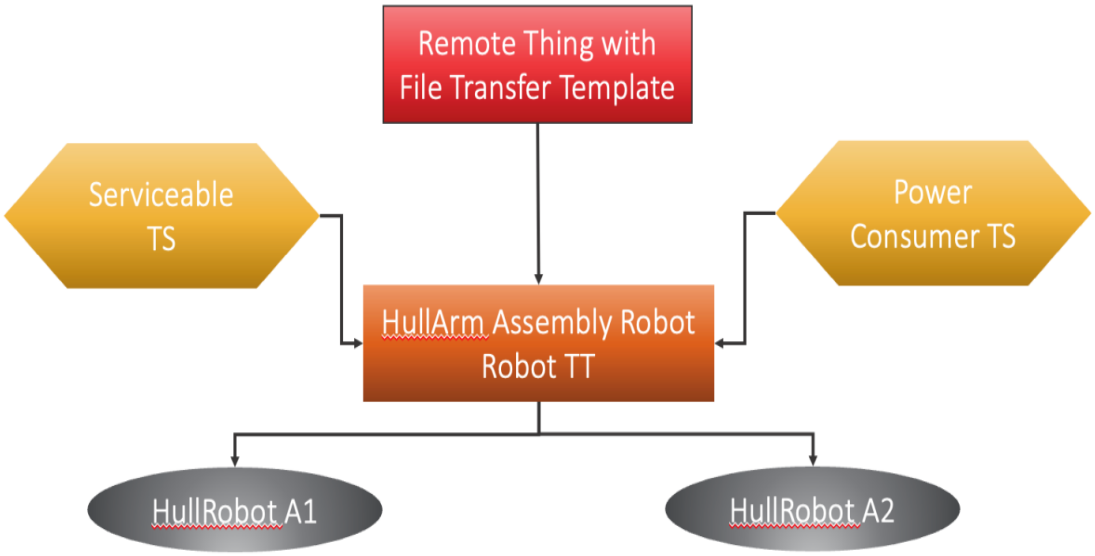
- Implemented only in ThingWorx
  - Not an edge device.
  - Mashups and applications manipulate it using REST API.

Representational State Transfer Application Programming Interface more commonly known as REST API web service. It means when a RESTful API is called, the server will transfer a representation of the requested resource's state to the client system.

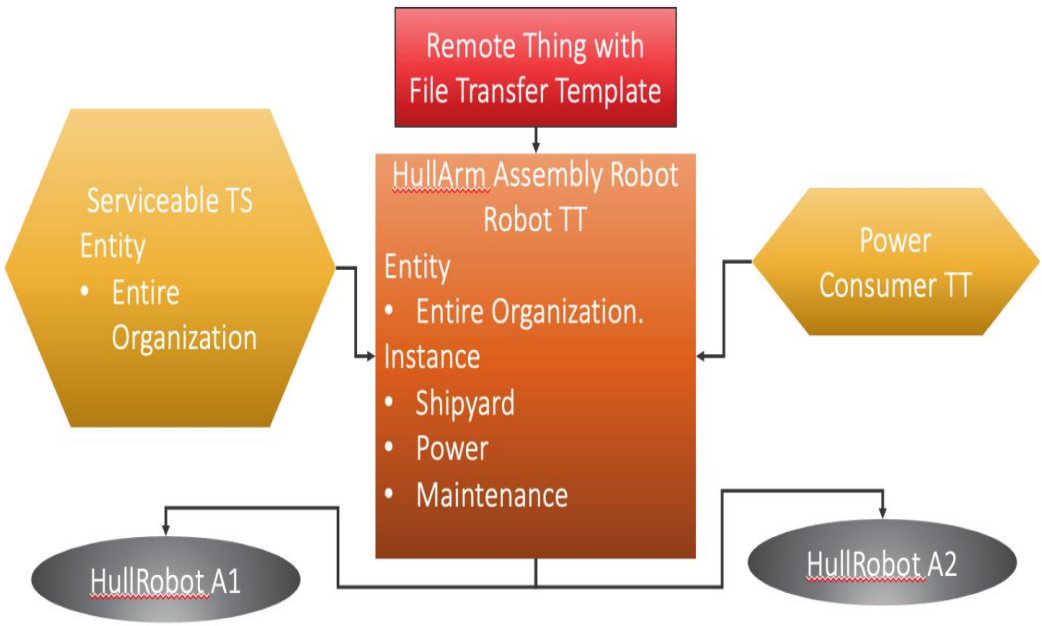
Model Composition - Assembly Robots



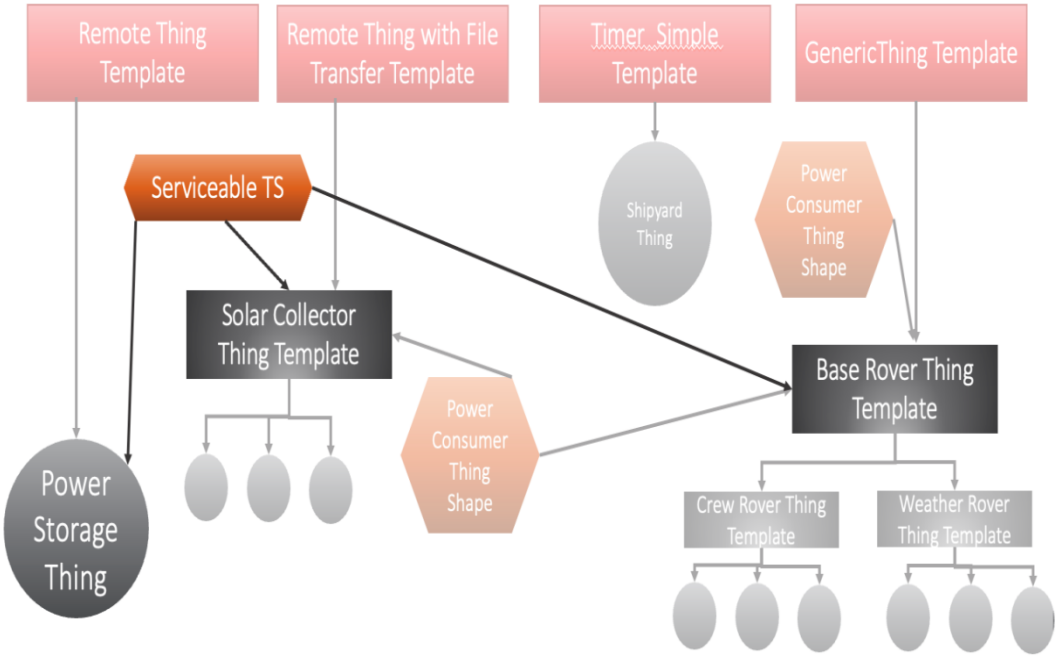
Implement Model Composition



Implement Model Visibility



Make Existing Shipyard Assets Serviceable



Implement Data Strategy

What functionality does the Serviceable Thing Shape need?

- Properties
  - Maintenance Status of asset (String).
  - Readable by Maintenance unit.

What functionality does a robot need?

- Properties
  - MfgSysData property (JSON).
  - Readable by Maintenance and unit.
- Services
  - Shutdown.
  - All power consumers need shutdown service.
  - Sets status to POWER OFF and power consumed to zero.

Exercise-1

Implement User Access Entities

STEP-1

❑ Create the users listed below.

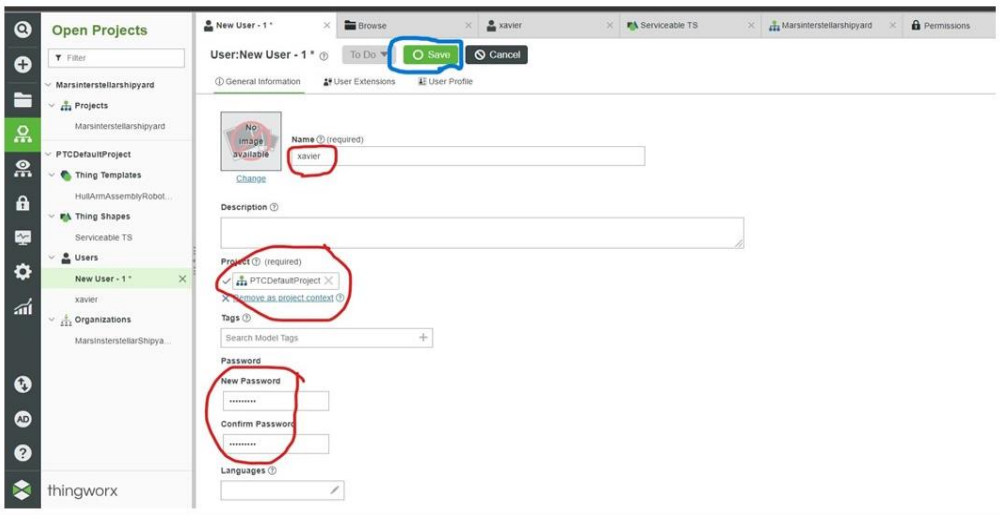
User Name	User Password
Xavier	ptcuniversityy
testTech1	ptcuniversityy

1.Open ThingWorx Developer Portal

2.Create user **Xavier** in Users

3. Create Password as" **ptcuniversityy "**

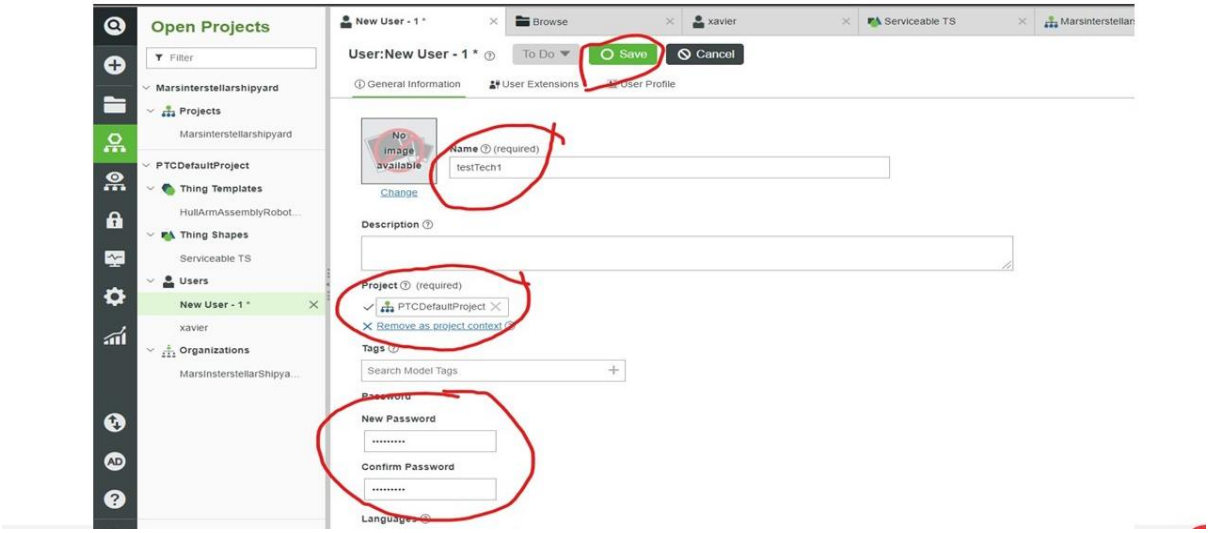
Create users Xavier & Password in thing Worx



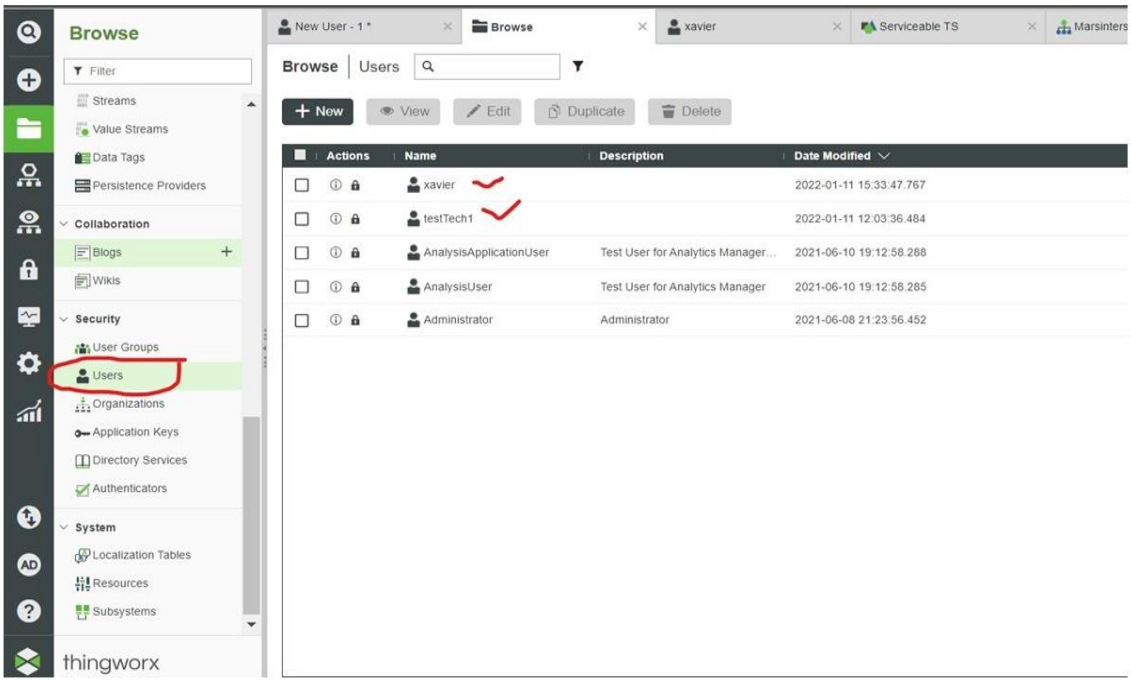
4. create one more user **testTech1** in users.

5.CreatePasswordas”**ptcuniversityy“**

Create user testTech1 & Password in thing Worx

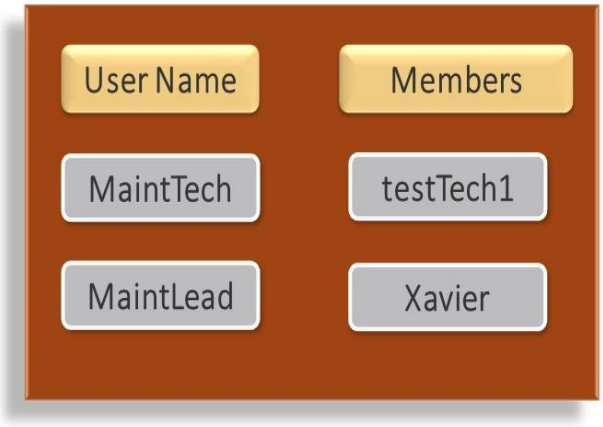


7.Now you can check the users in user list as below.

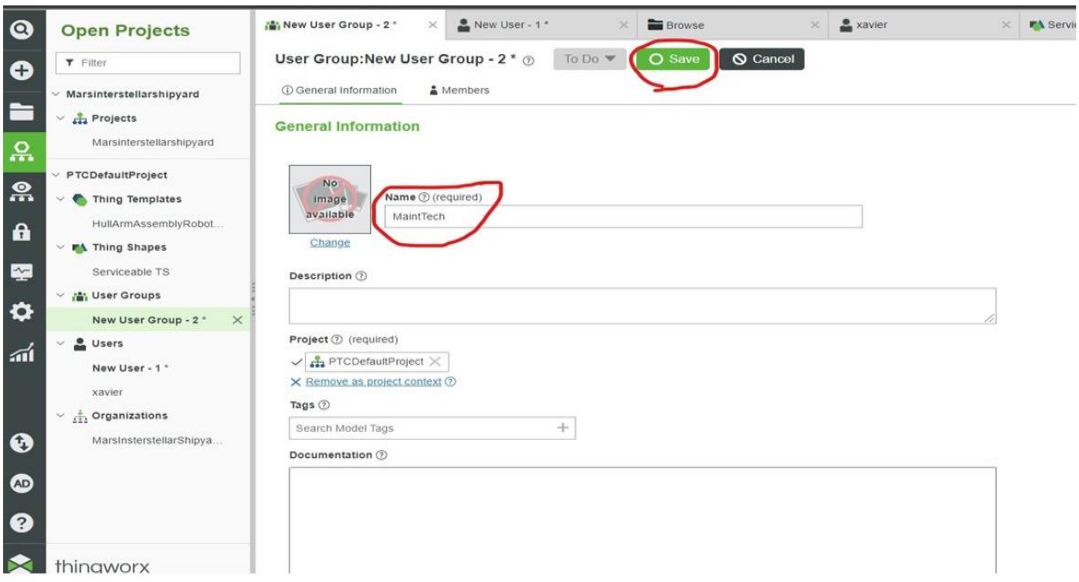


STEP-2

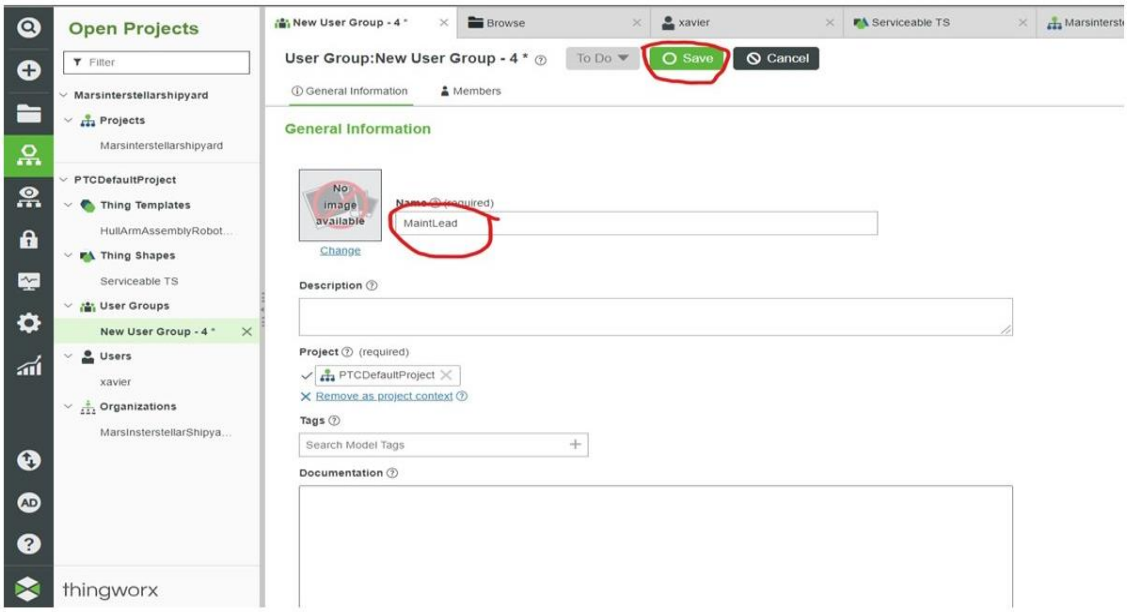
❑ Create the logical groups listed below.



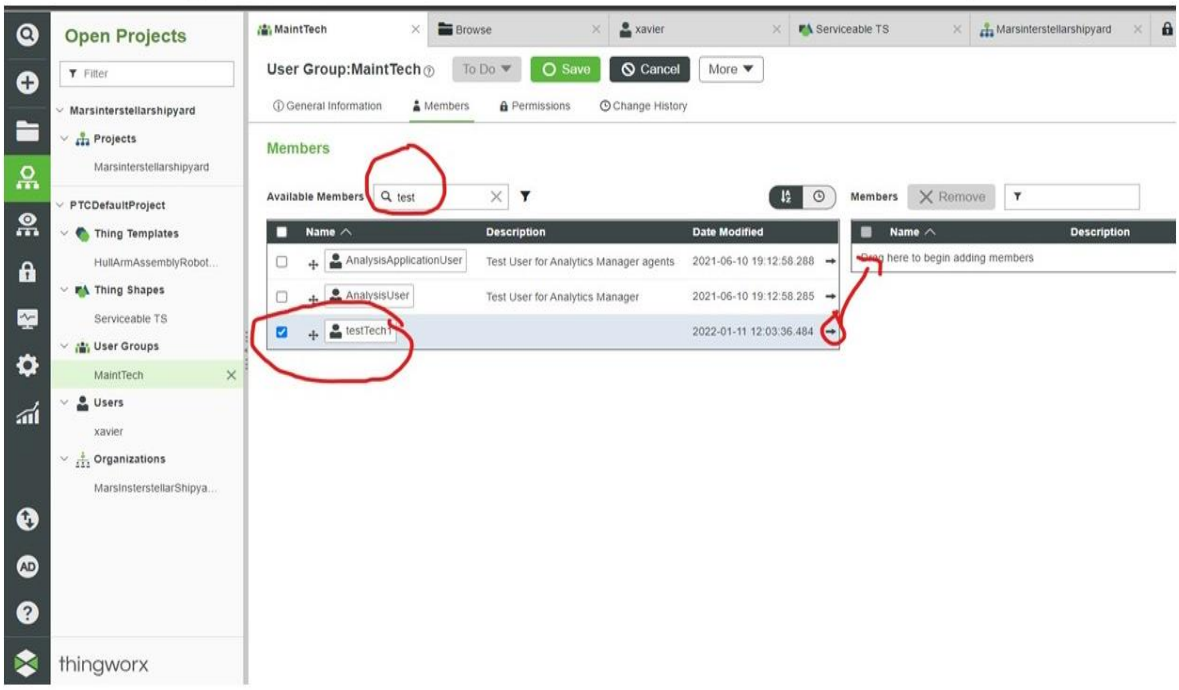
Create Logical Group for Maintech in thing Worx



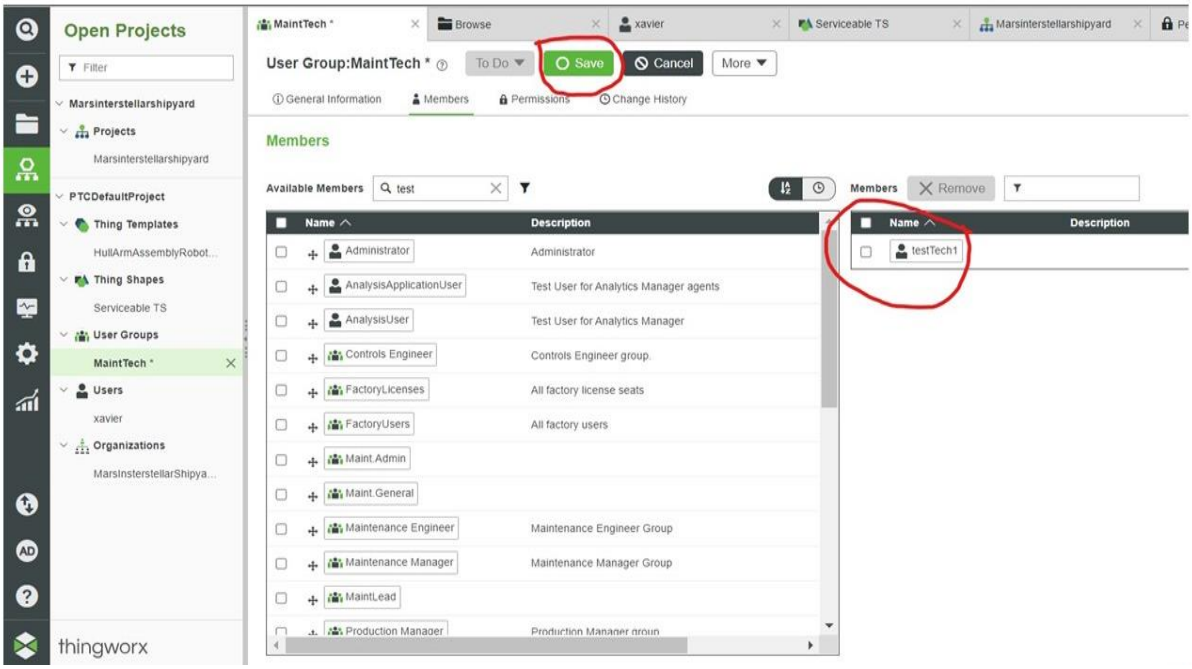
Create Logical Group for MaintLead in thing Worx



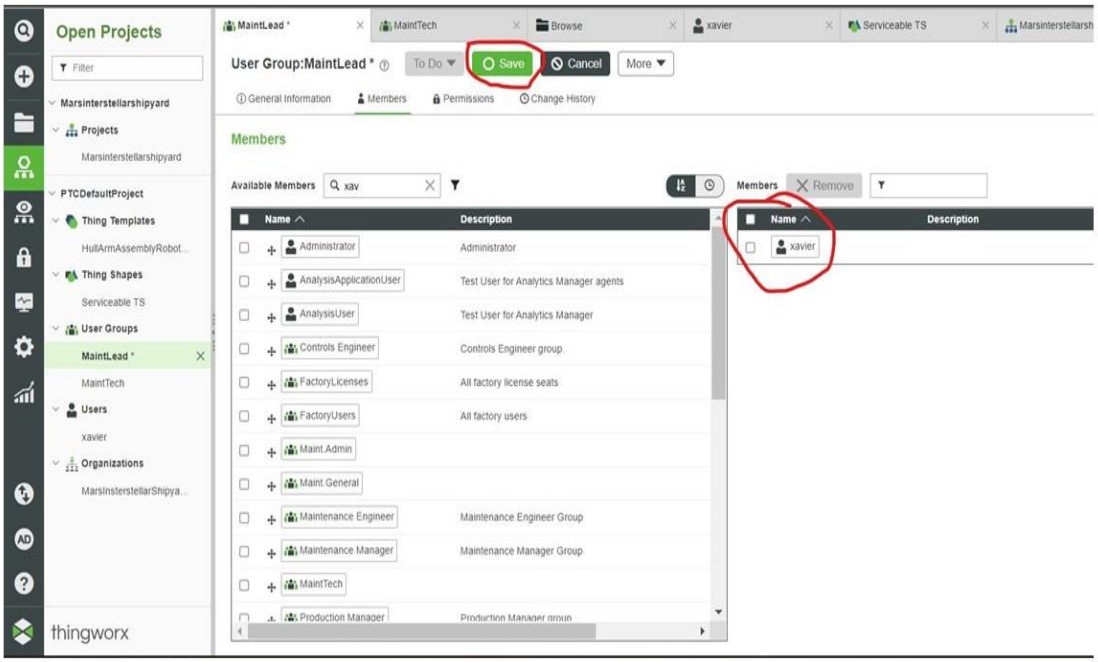
Assign as a member to testTech1 for Maintech in thing Worx



Assign as a member to testTech1 for Maintech in thing Worx

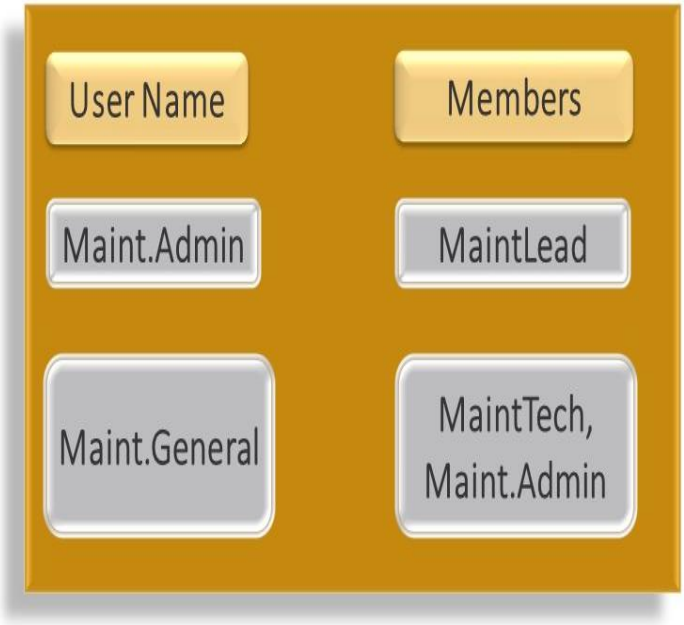


Assign as a member to Xavier for MaintLead in thing Worx

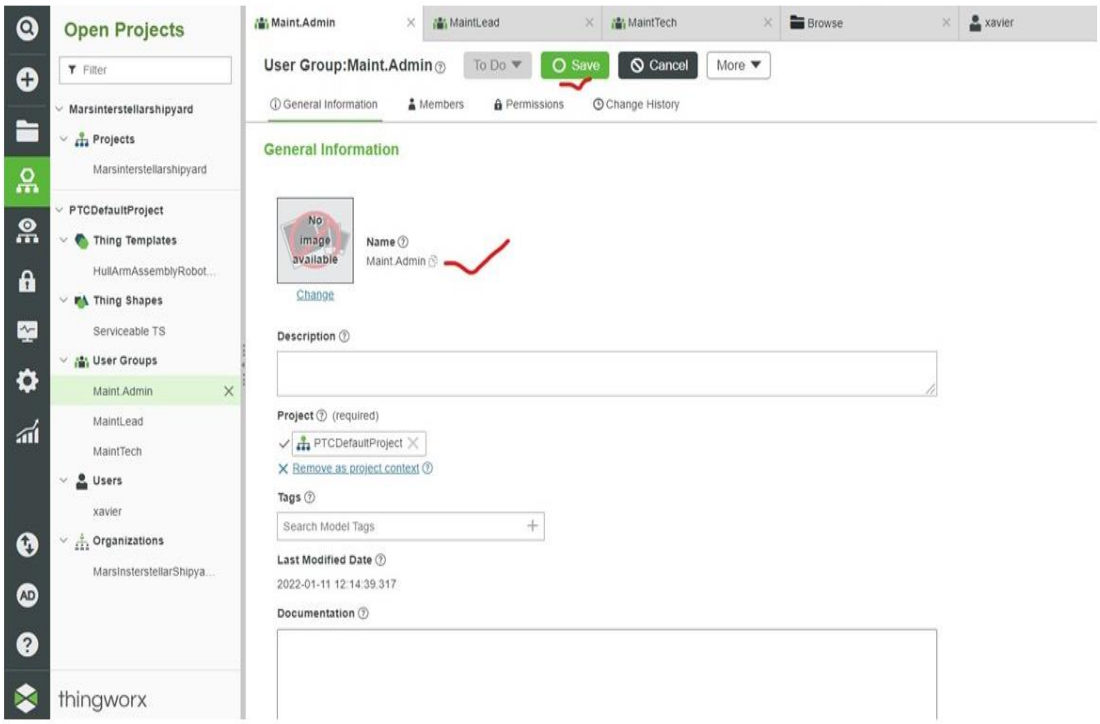


STEP-3

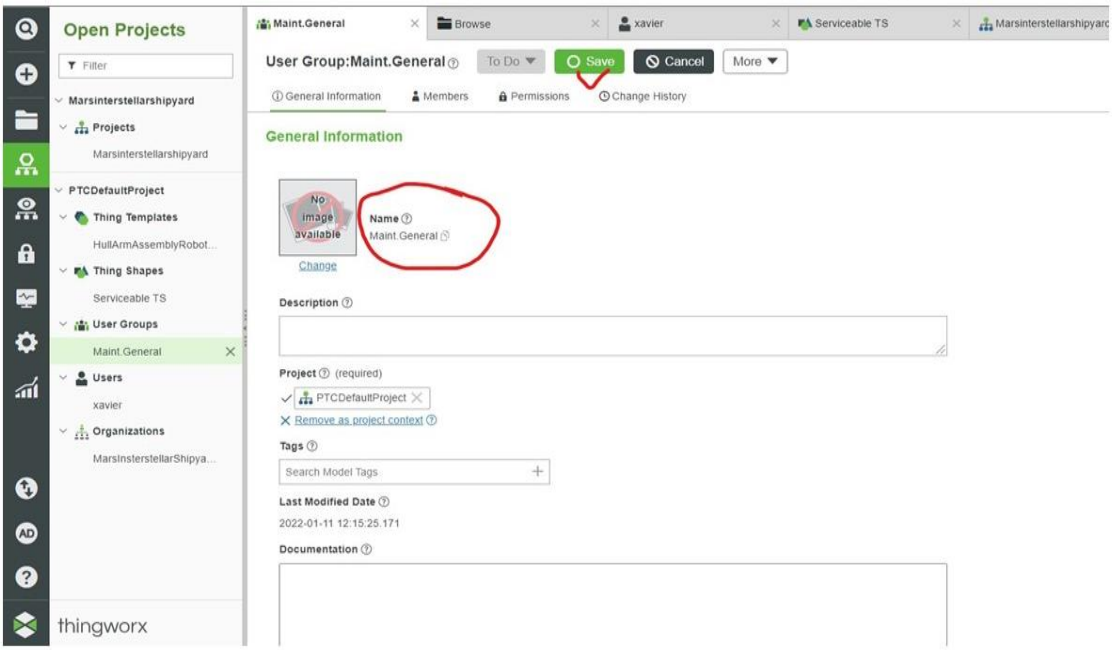
❑ Create the role-based groups listed below.



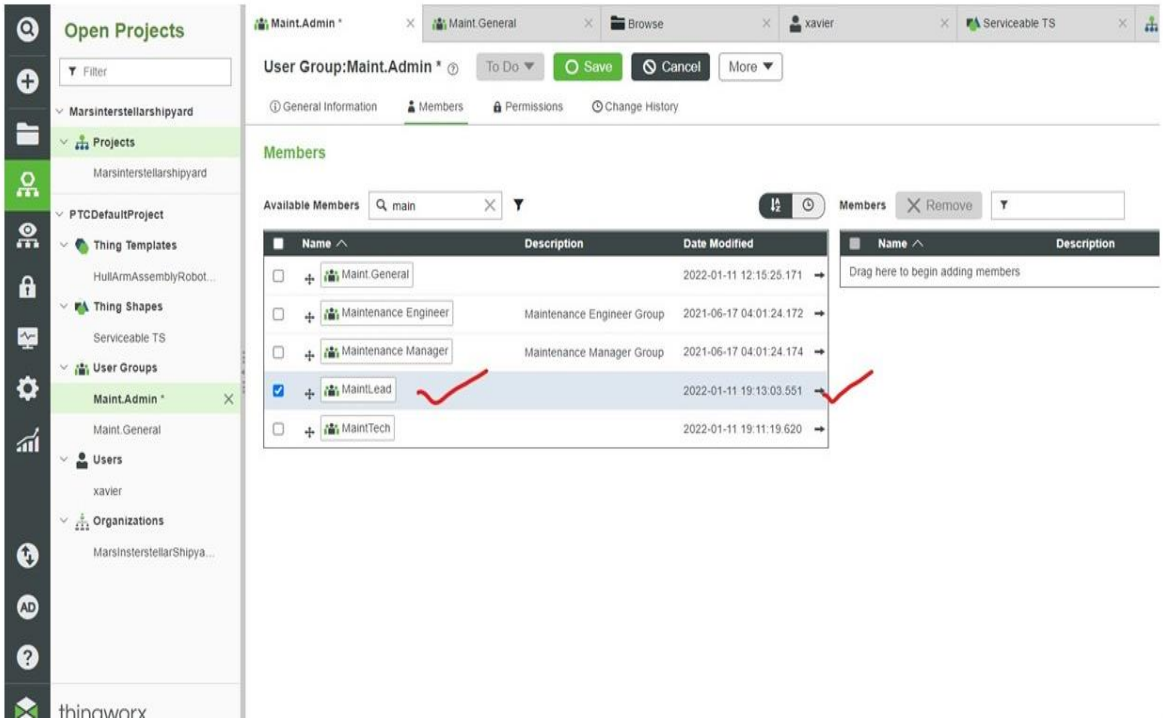
Create Role based Group for Maint.Admin in thing Worx

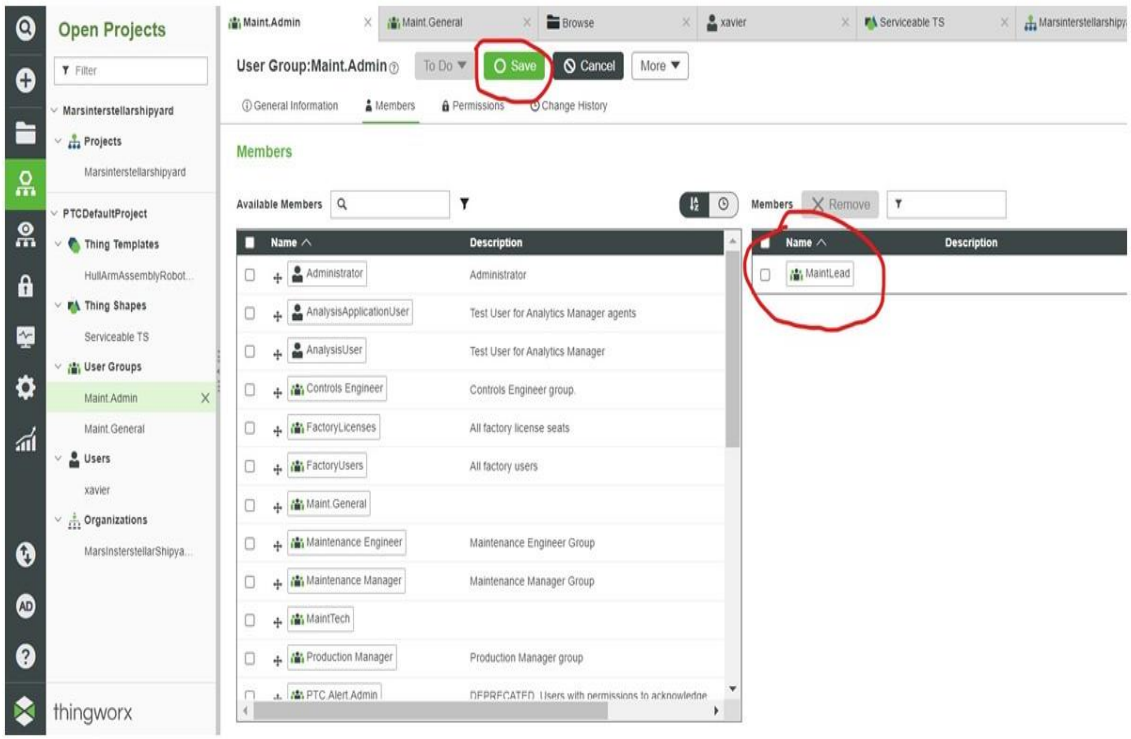


Create Role based Group for Maint.General in thing Worx

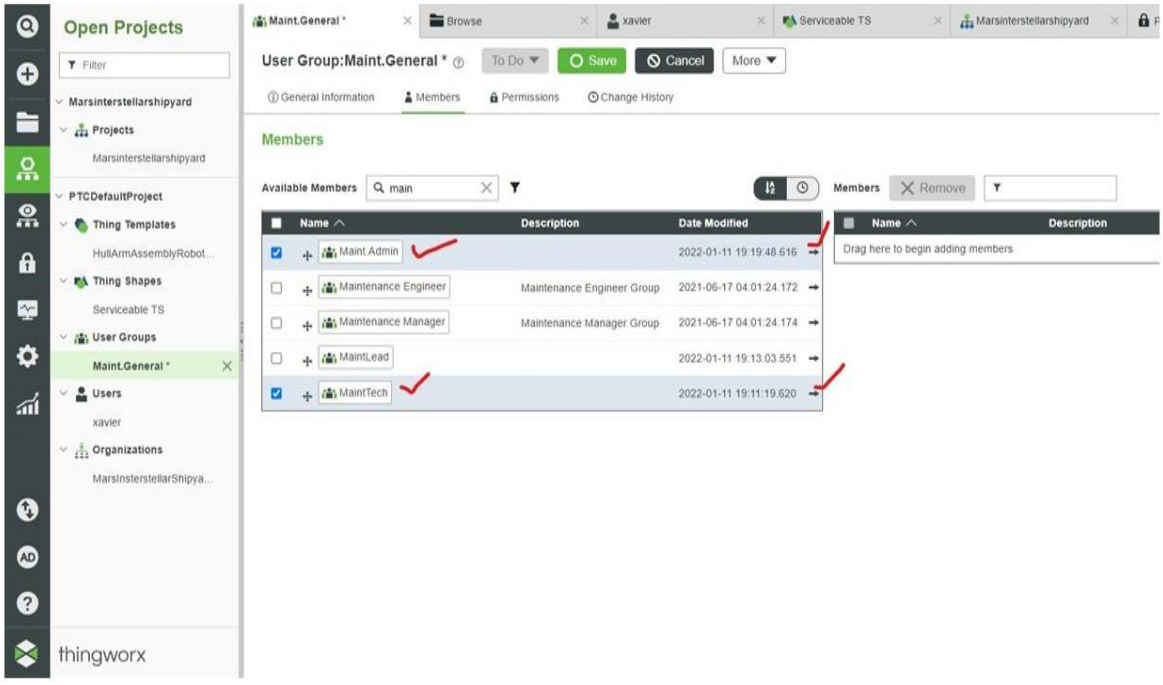


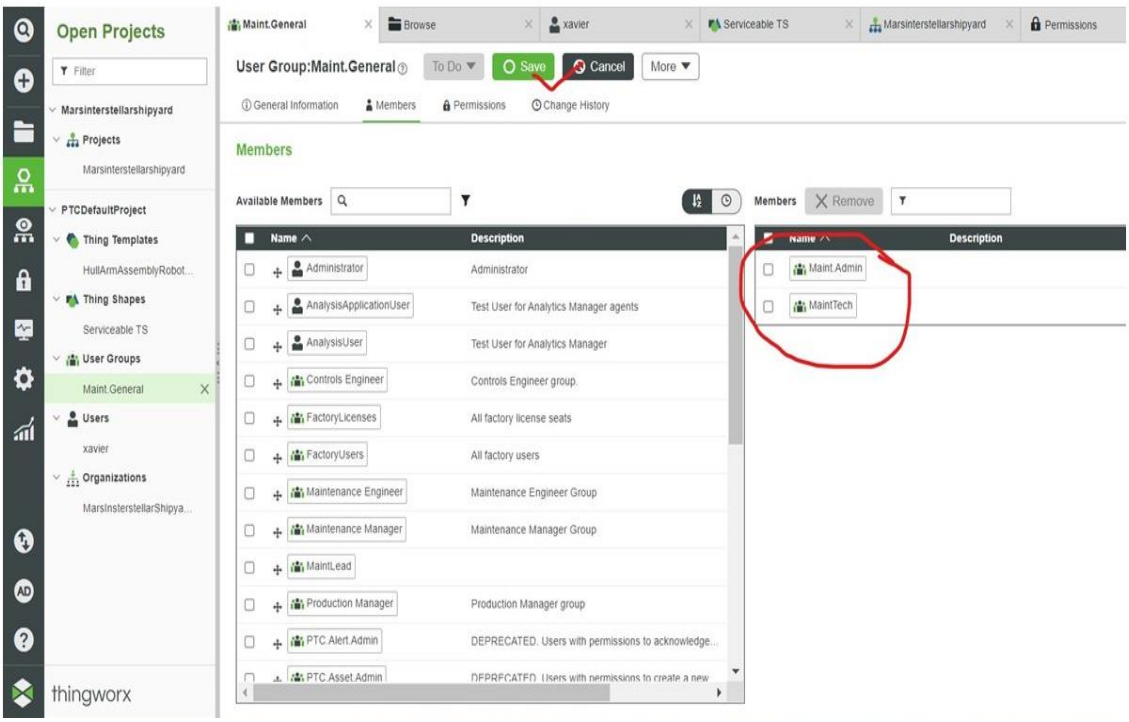
Assign as a member to MaintLead for Maint.Admin in thing Worx





Assign as a member to MaintTech, Maint.Admin for Maint.General in thing Worx



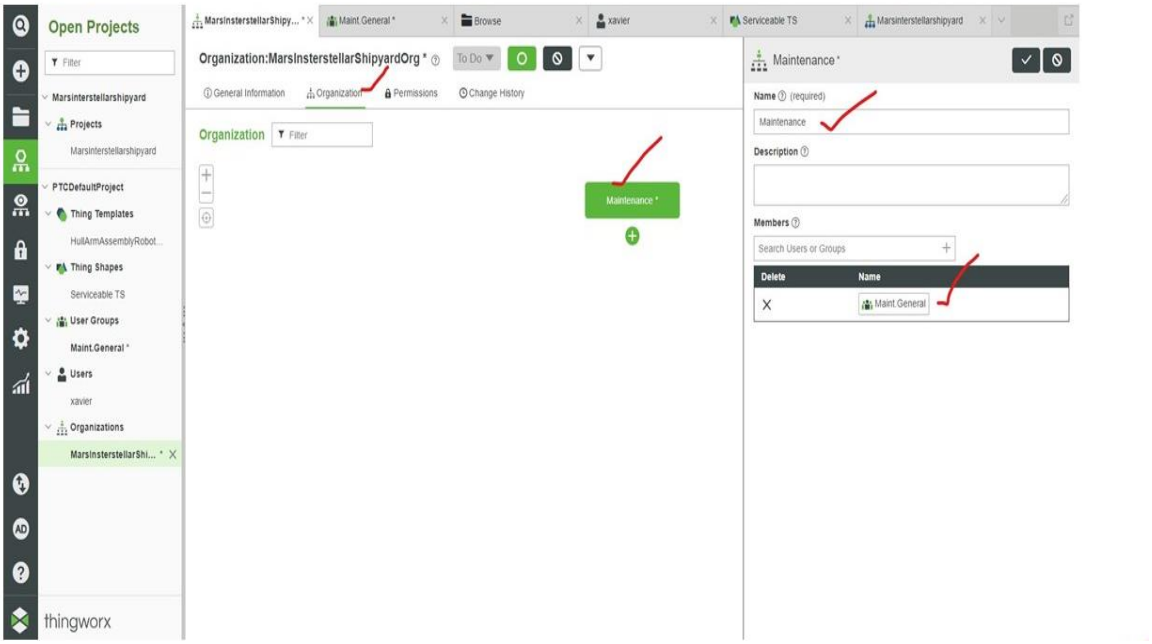


STEP-4

❑ Create the Organizational unit below.

Parent Organization	Name	Members
MarsInterstellarShipyardOrg	Maintenance	Maint.General

Create Maintenance & Assigning as a member for MarsInterstellarShipyardOrg in thingWorx



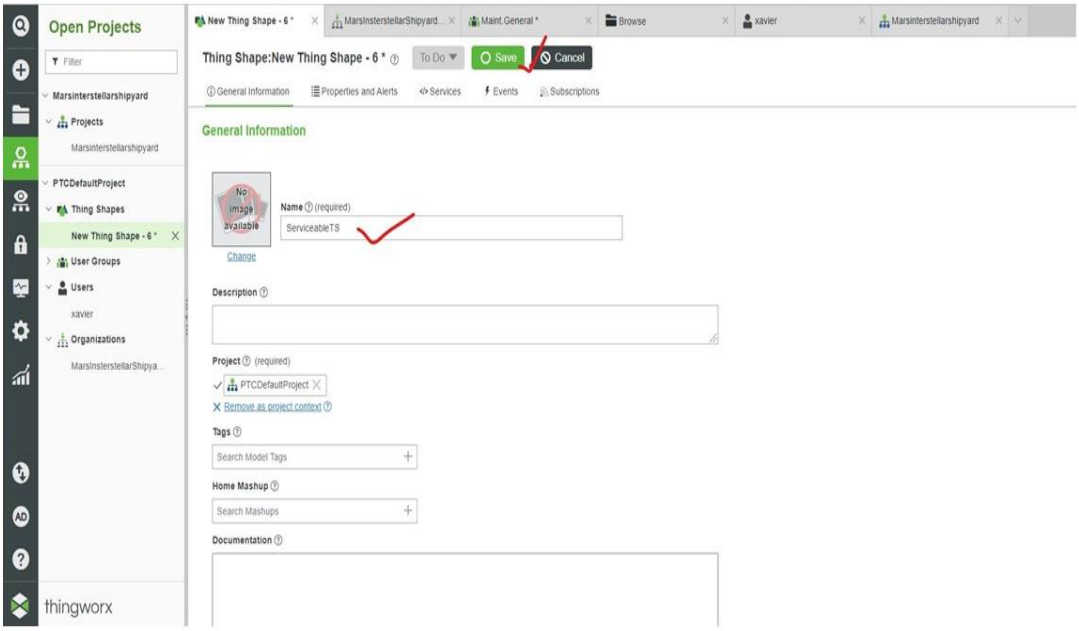
Exercise-2

Implement Model Composition

STEP-1

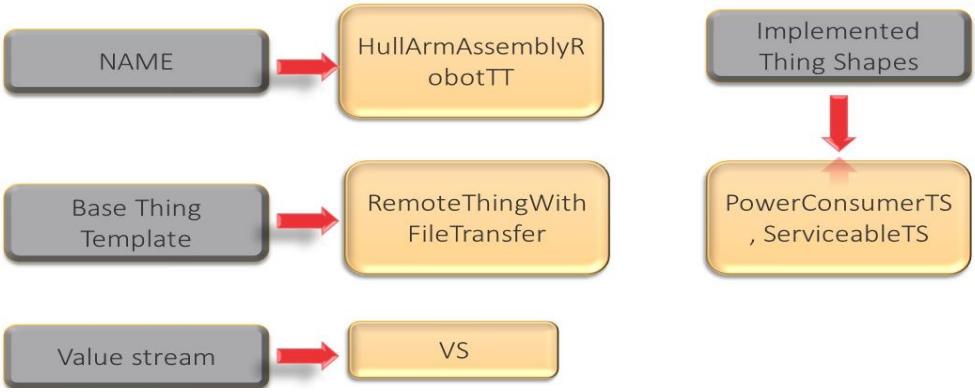
❑ Create the Thing shape below.

Create Thing shape for ServiceableTS in thing Worx

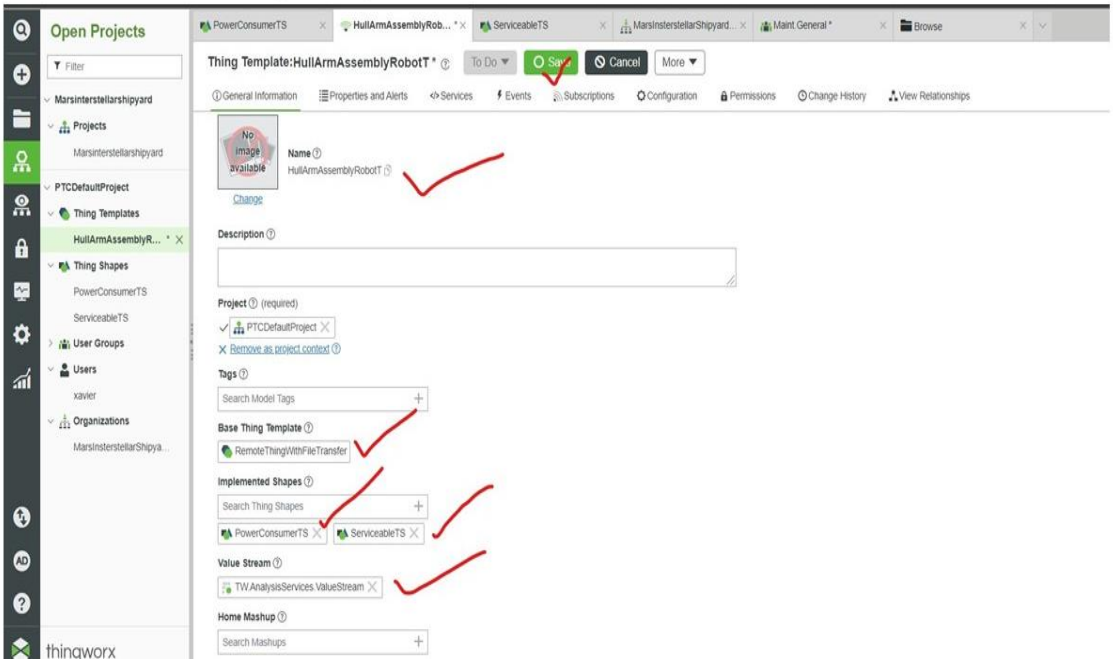


STEP-2

❑ Create the Thing Template below.

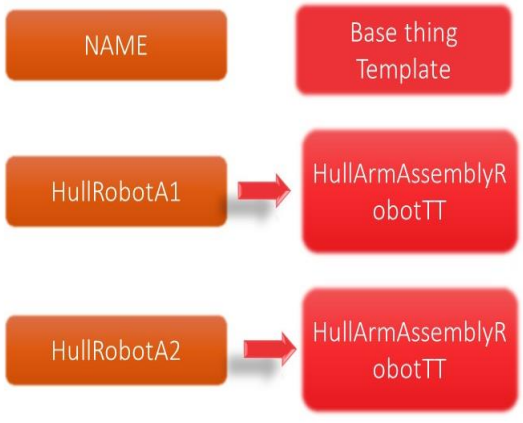


Create Thing Template

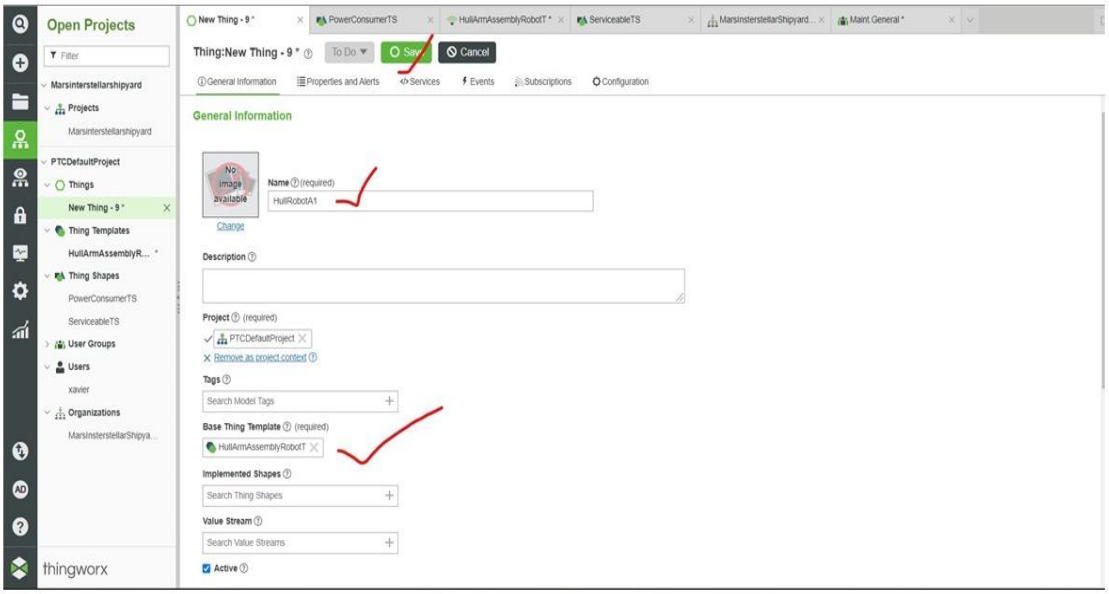


STEP-3

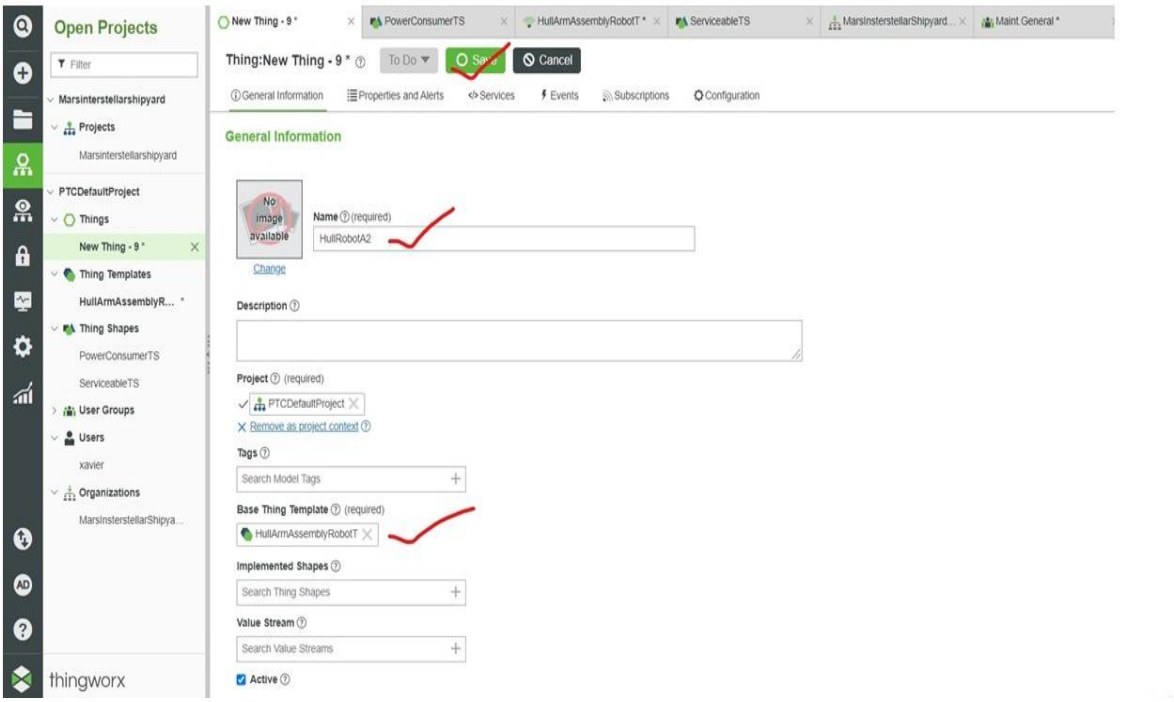
❑ Create the Things below.



Create Things for HullRobotA1



Create Things for HullRobotA2

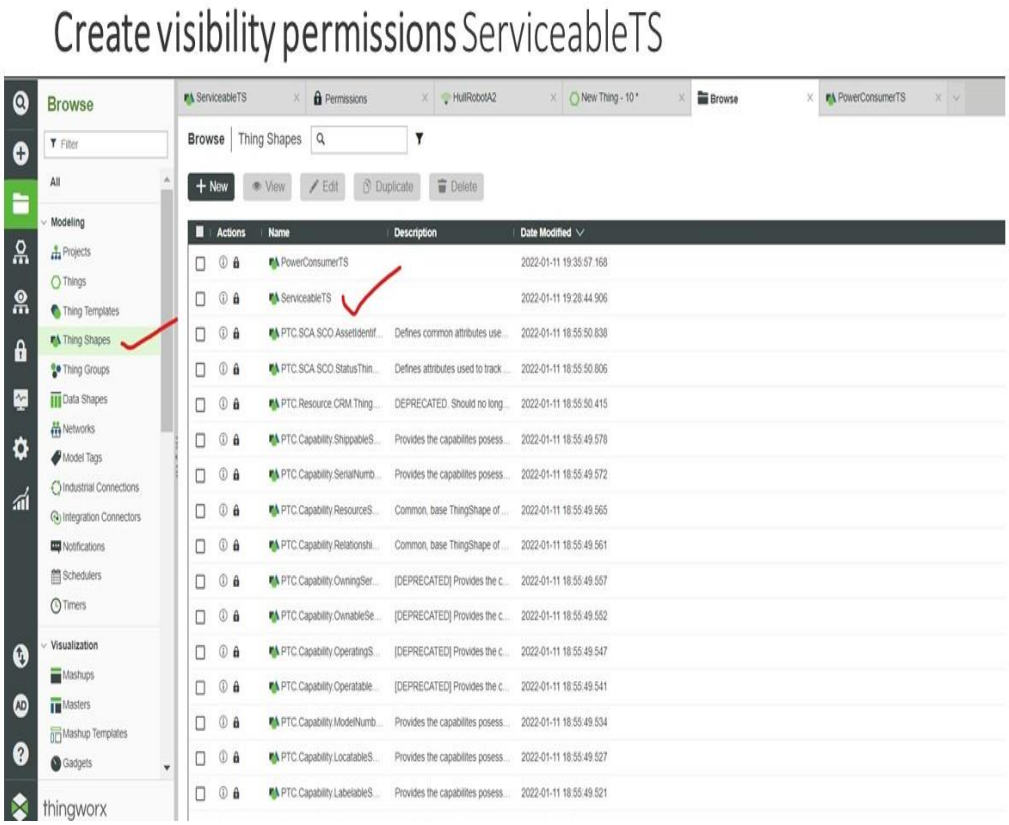


Exercise-3

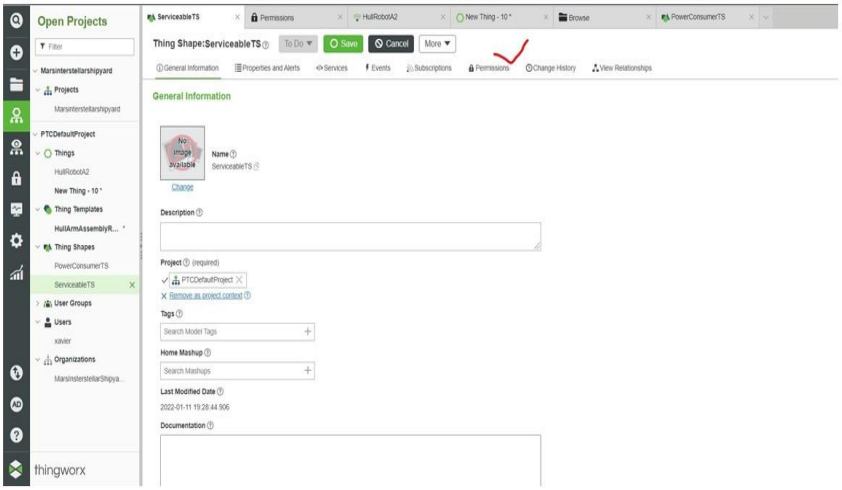
Implement Model Visibility

- Create the Visibility Permissions as per below table.

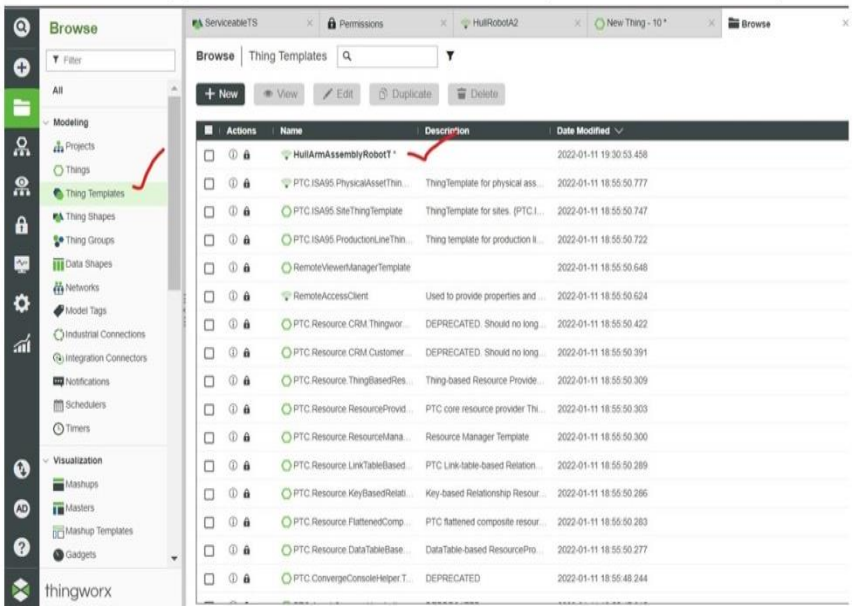
Entity	Entity or Instance?	Organizational Unit(s)
ServiceableTS	Entity	MarsInterstellarShipyard Org
HullArmAssemblyRobotT T	Entity	MarsInterstellarShipyard Org
HullArmAssemblyRobotT T	Instance	Shipyard, Power, Maintenance



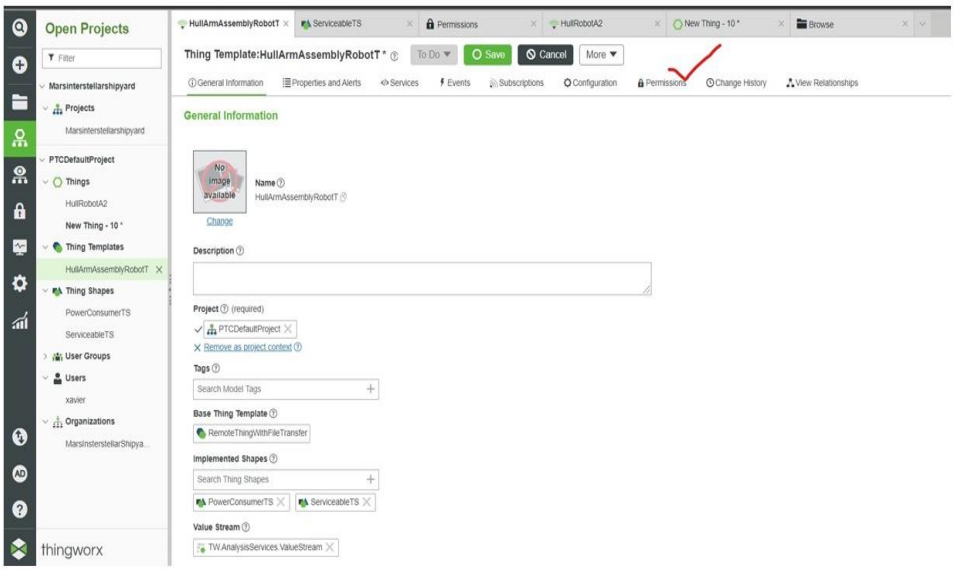
Create visibility permissions ServiceableTS



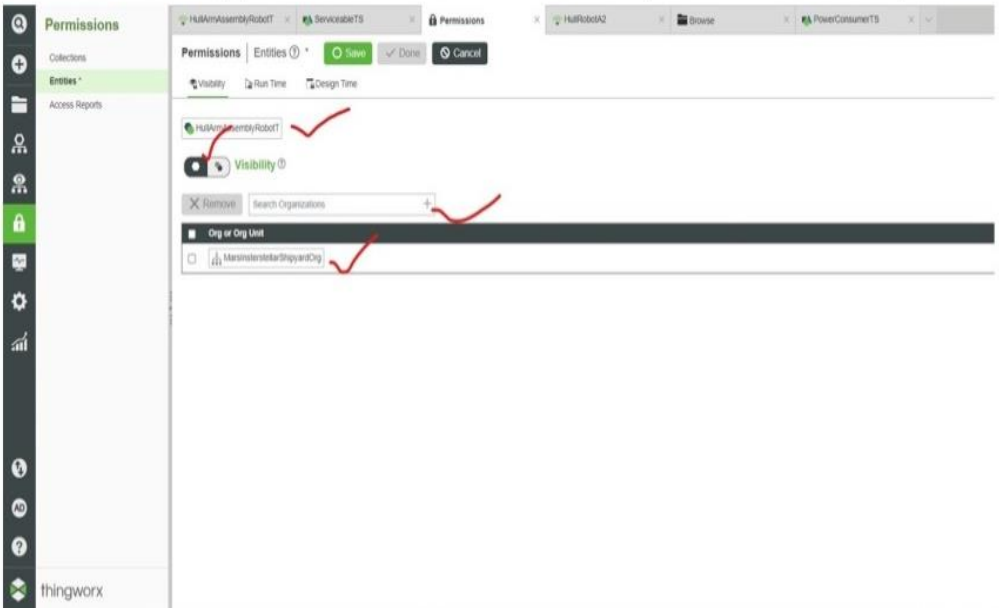
Create visibility permissions HullArmAssemblyRobotTT for Entity



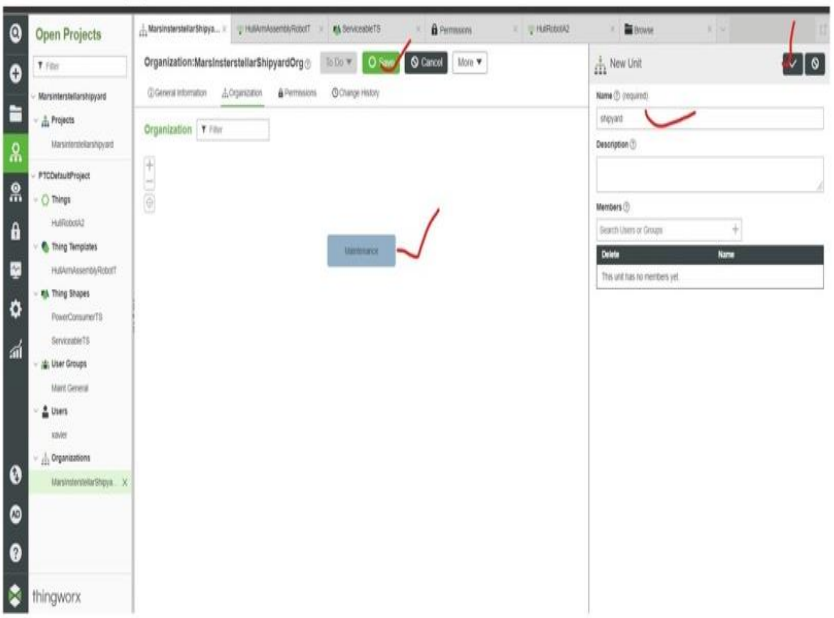
Create visibility permissions HullArmAssemblyRobotTT for Entity



Create visibility permissions HullArmAssemblyRobotTT for Entity



Create Shipyard unit



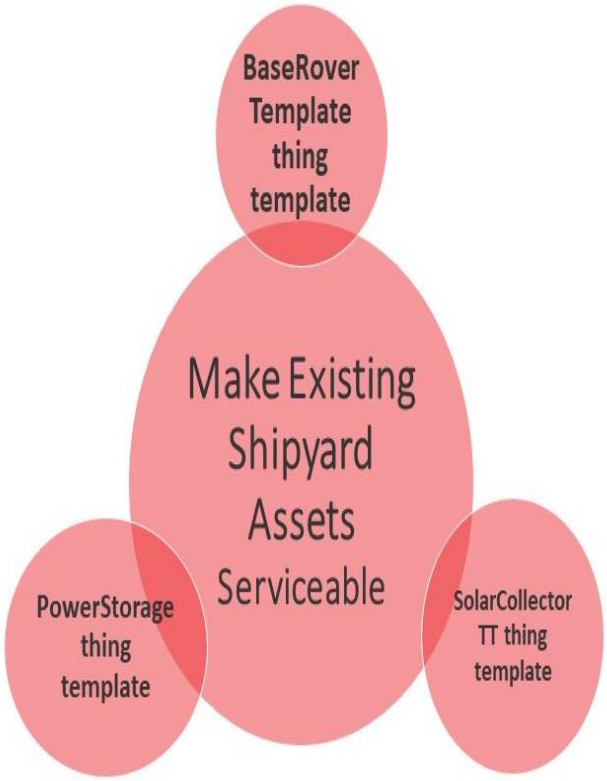
Exercise-4

Make Existing Shipyard Assets Serviceable

The ServiceableTS thing shape should be applied to the following entities:

1. Create base rover template thing template
2. Create power storage thing template
3. Create solar collector TT thing template

Refer Module -5 to apply following above entities to ServiceableTS thing shape



Exercise-5

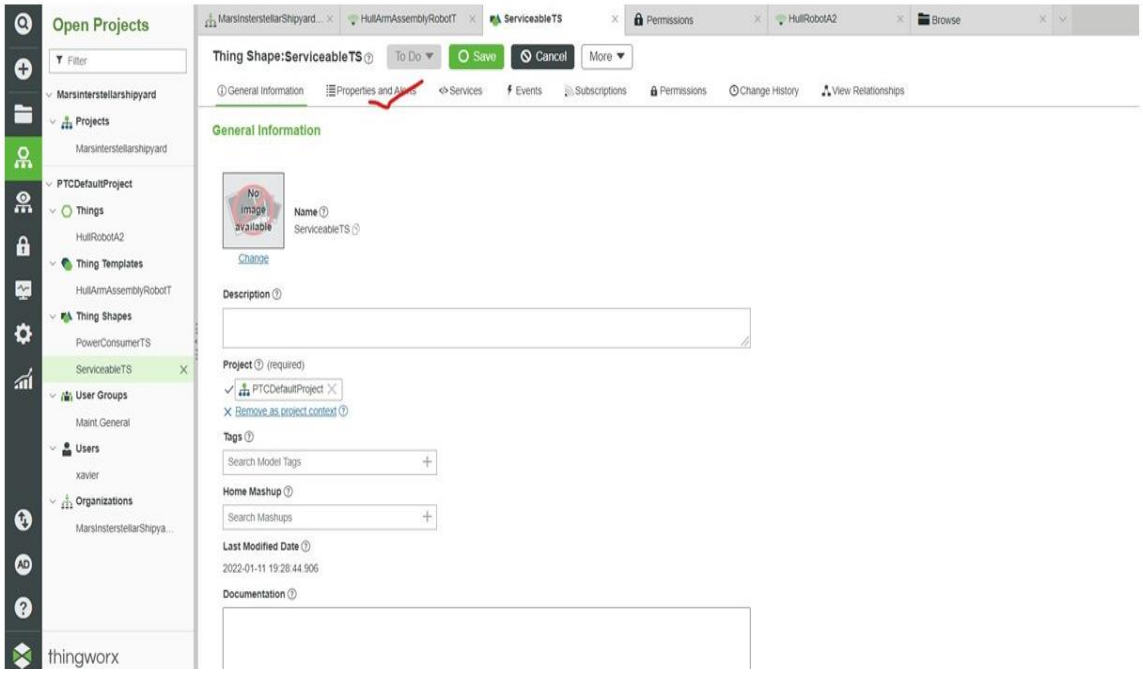
Implement Data Strategy

Step-1

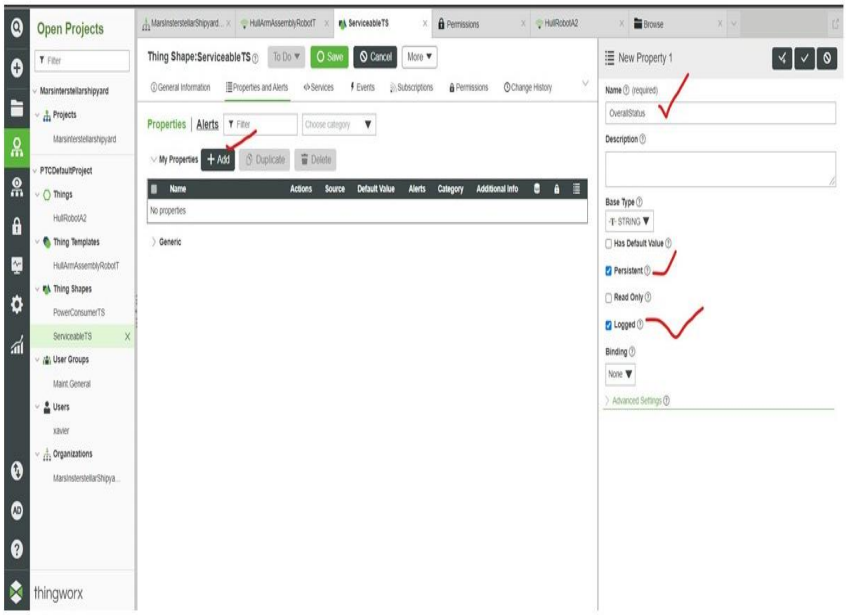
❑ Create the properties listed below.

Entity Name	Property Name	Base Type	Default Value	Persistent	Logged
ServiceableTS	Overall Status	STRING	Online	Yes	Yes
HullArmAssemblyRobotTT	MfgSysData	JSON	None	No	No

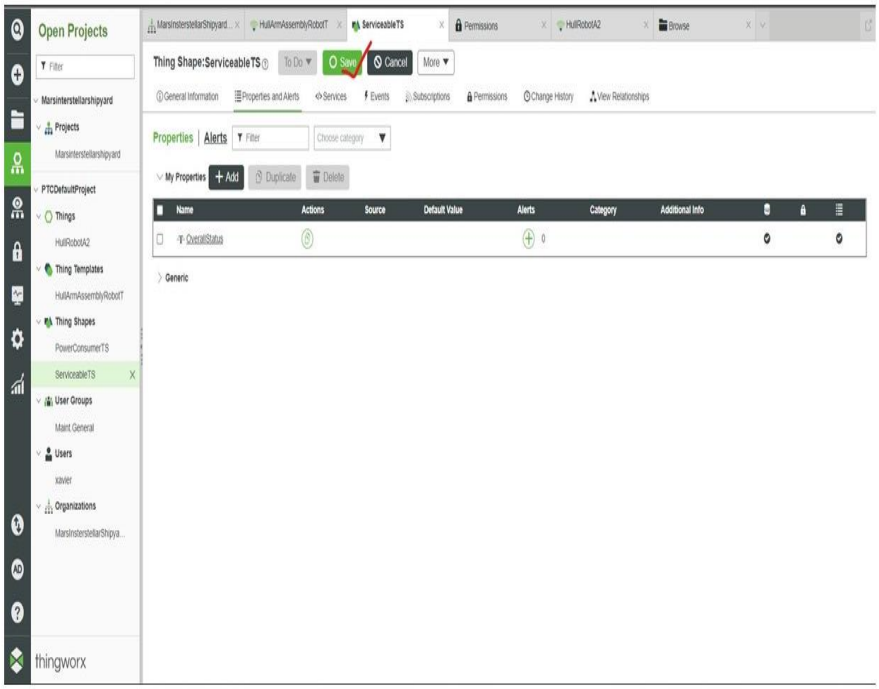
Create properties for ServiceableTS



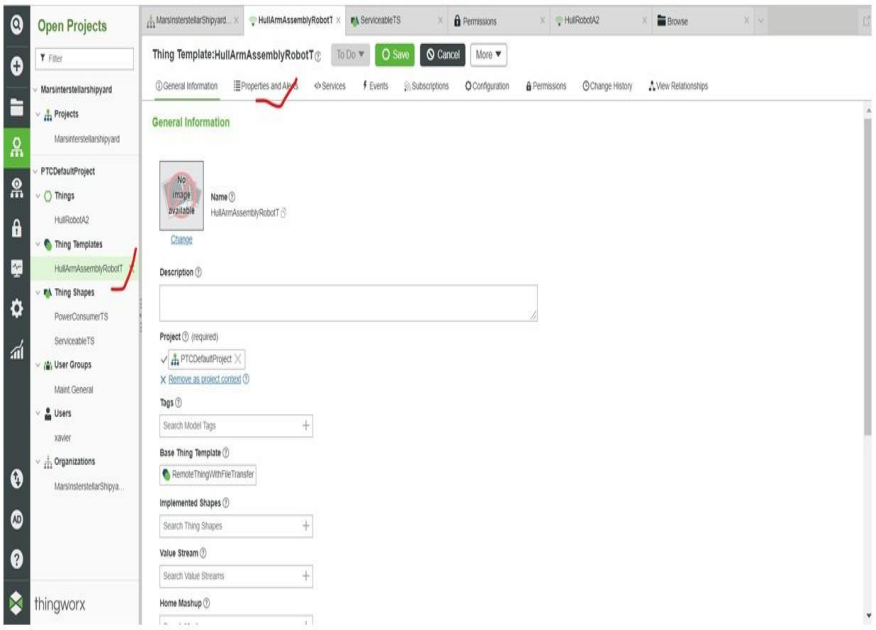
Create properties for ServiceableTS



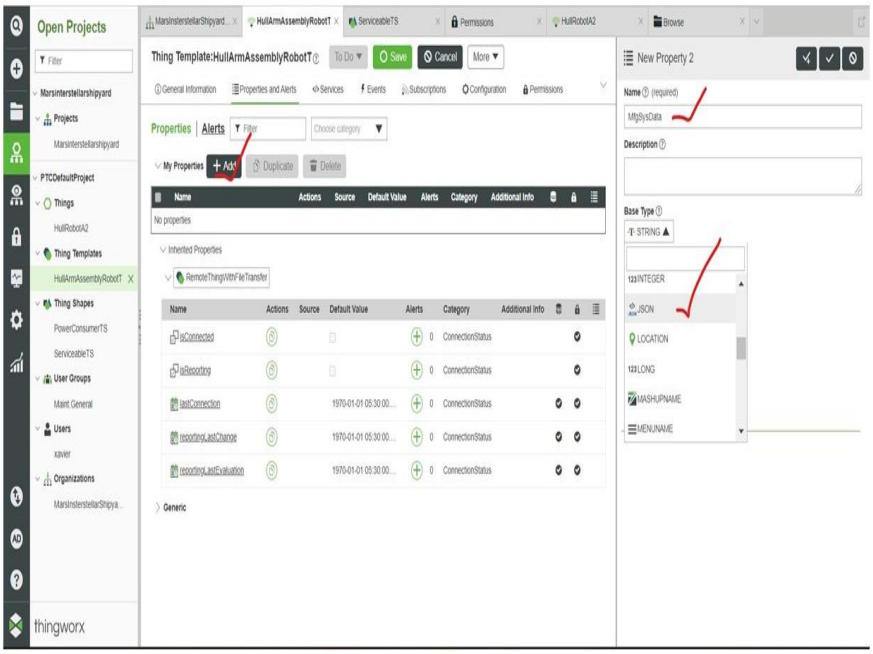
Create properties for ServiceableTS



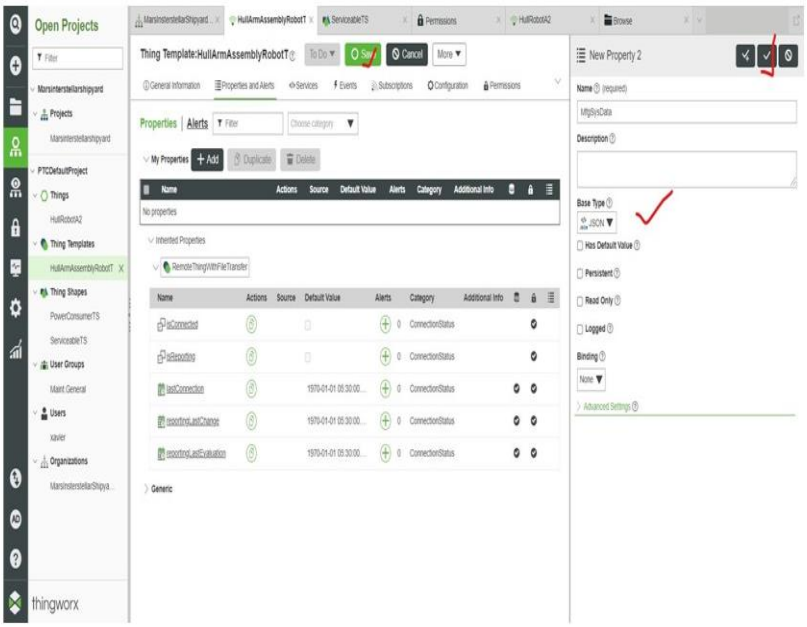
Create properties for HullArmAssemblyRobotTT



Create properties for HullArmAssemblyRobotTT



Create properties for HullArmAssemblyRobotTT

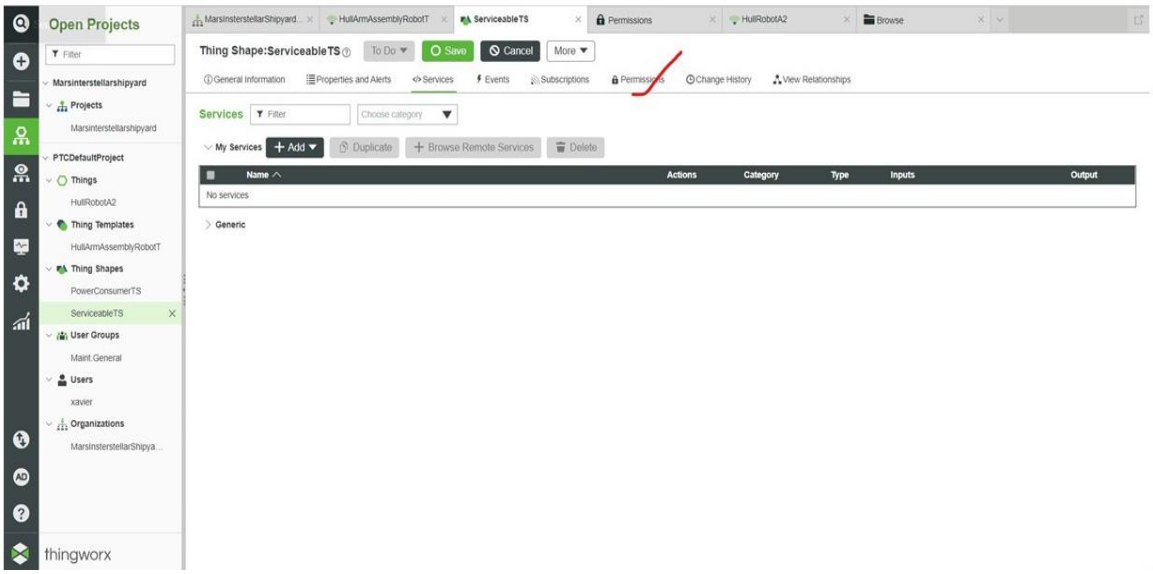


Step-2

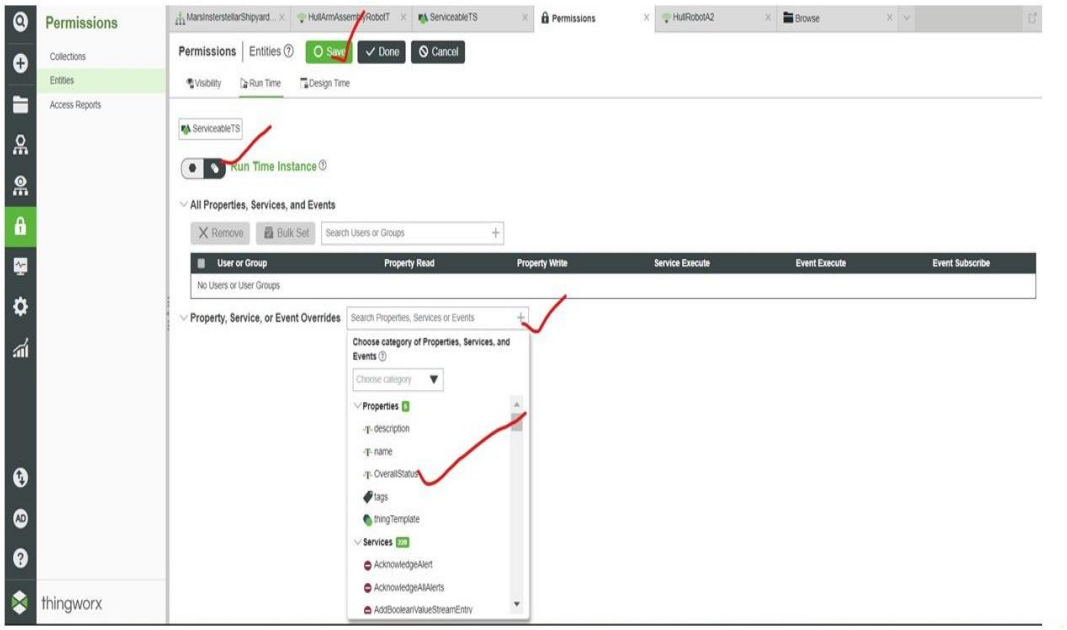
❑ Allow the following permissions during run time :

Entity Name	Property Name	Entity or Instance?	Permission	Group(s)
ServiceableTS	Overall Status	Instance	Property Read	Maint.General, ShipyardAdminGrp
ServiceableTS	Overall Status	Instance	Property Write	Maint.Admin
HullArmAssemblyRobotTT	MfgSysData	Instance	Property Read	Maint.General, ShipyardAdminGrp

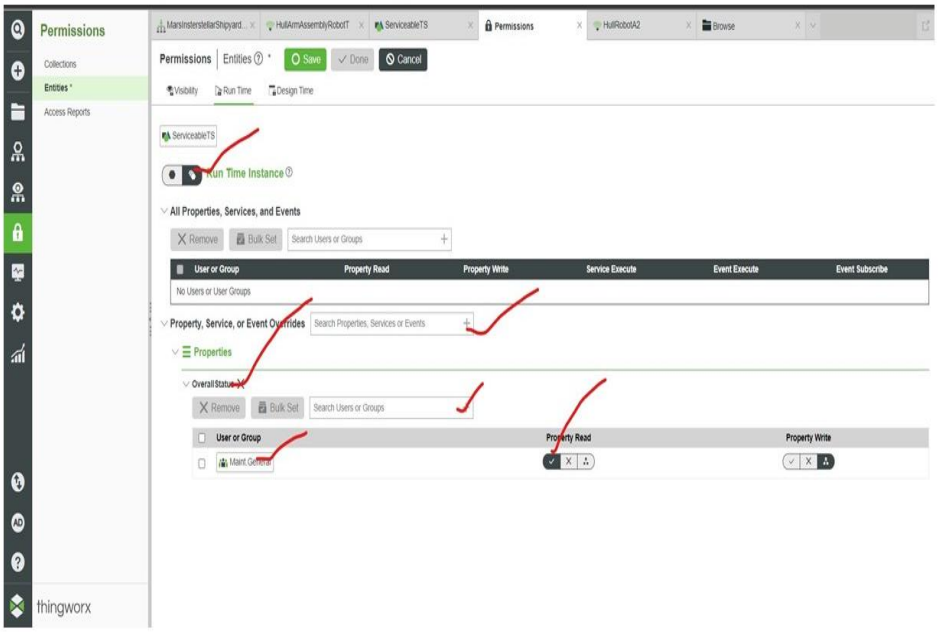
Create permissions during run time for ServiceableTS



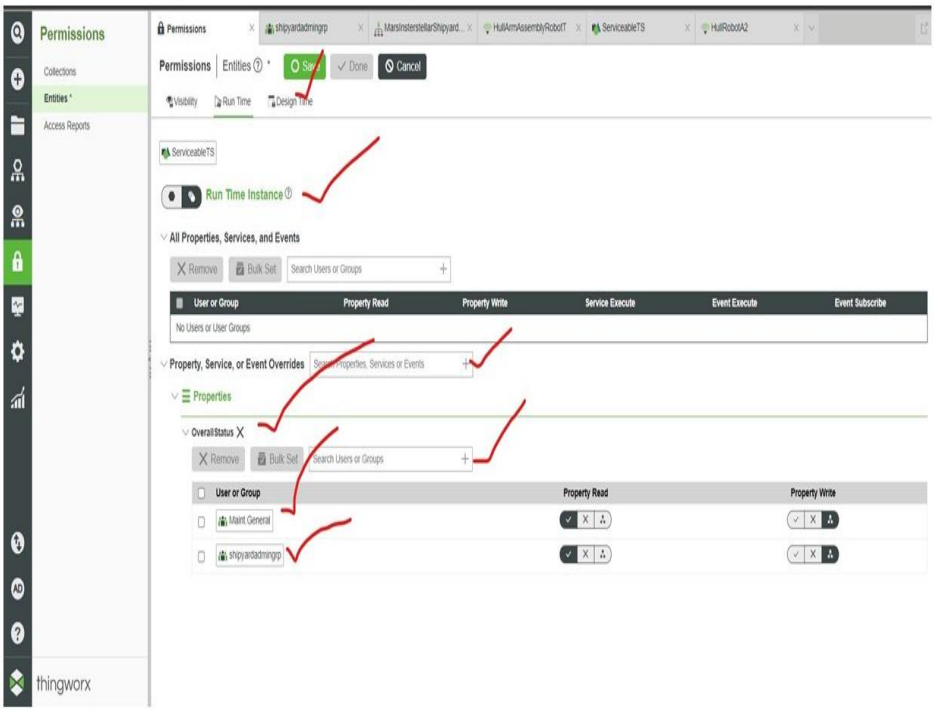
Create permissions during run time for ServiceableTS



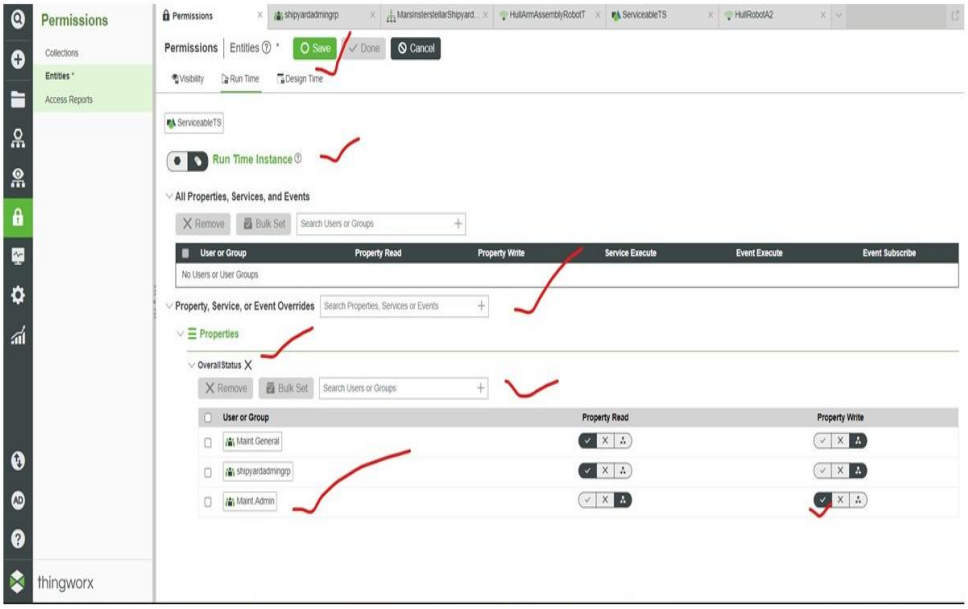
Create permissions during run time for ServiceableTS



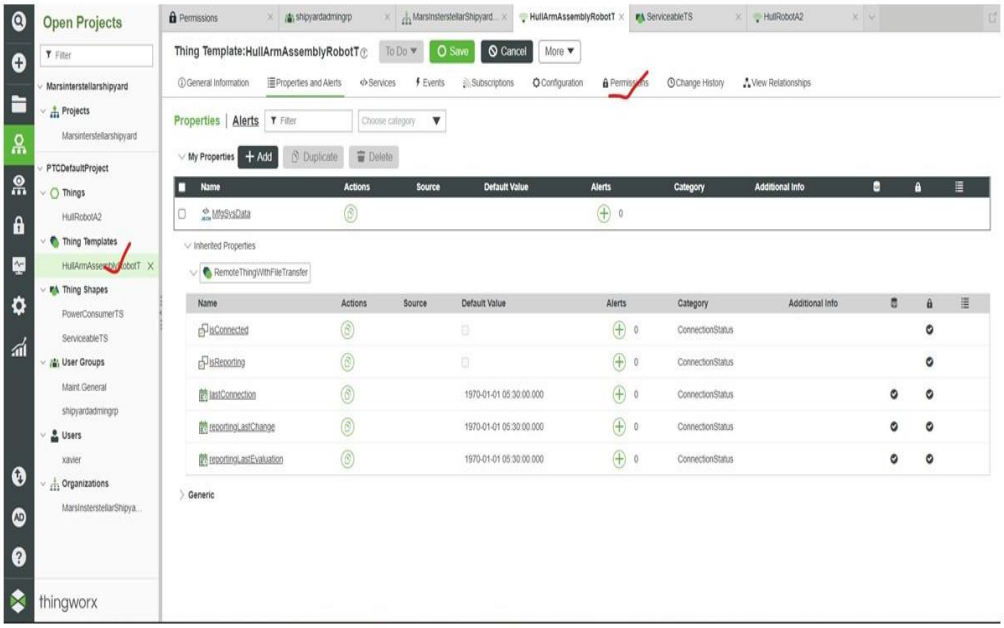
Create permissions during run time for ServiceableTS



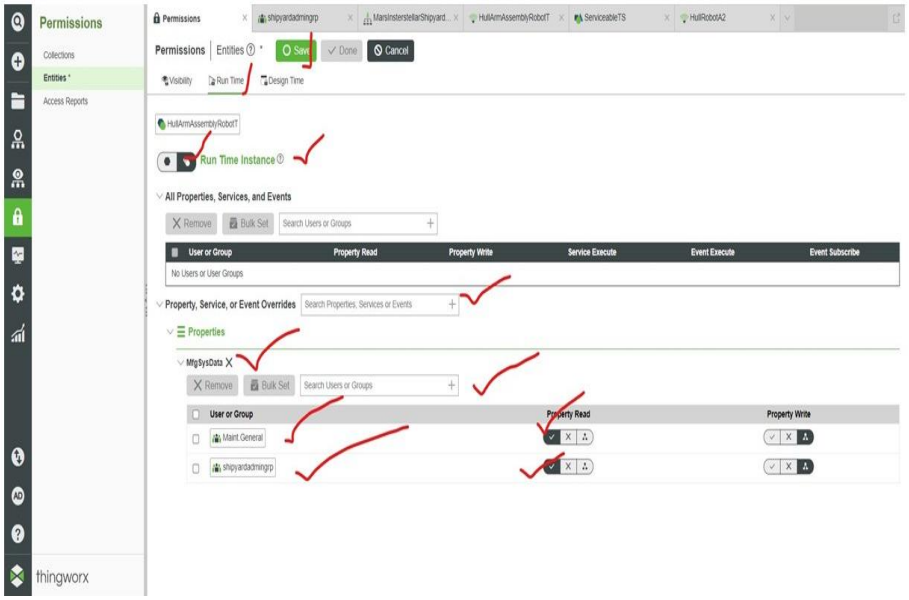
Create permissions during run time for ServiceableTS



Create permissions during run time for HullArmAssemblyRobotTT



Create permissions during run time for HullArmAssemblyRobotTT



Step-3

❑ Create the Service listed below.

Entity Name	Service Name	Code
HullArmAssemblyRobotTT	Shutdown	me.OverallStatus = "POWER OFF"; me.PowerConsumed = 0;

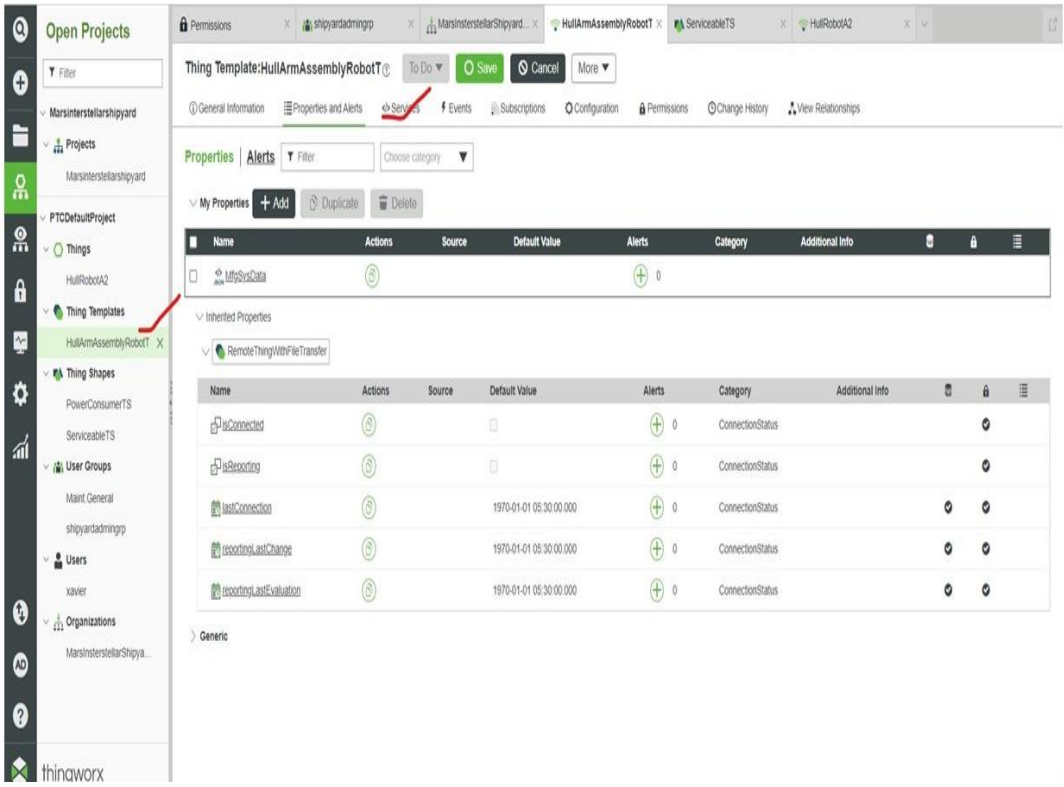
Note from the Instructor:-

This code will be saved in the virtual machine .

❑ The Path is : **D:\Student\Code Snippets**

❑ The file name is: **Model_M7_Ex5_HullAssemblyArmRobotTT_Shutdown.txt**

Create Service for HullArmAssemblyRobotTT



Create Service for HullArmAssemblyRobotTT