|  |
| --- |
|  |
| Solution Design Document  Thingworx |
| ELGi Equipments |
| Ankit Pandya  apandya@ptc.com |

Document Properties

|  |  |
| --- | --- |
| File Name | Status |
| SolutionDesignDocument-Elgi\_v03\_1 | In-Process |

Change History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Description |
| JAN/14/2020 | Ankit Pandya | 1.0 | Initial draft of the document. |
| MAR/10/2020 | Ajay Valvi | 3.0 | Solution Design |
| JUN/22/2020 | Ajay Valvi |Ketan Magdum | 4.0 | Updated to include Rules, SCM and Architecture |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

PTC Product Version

|  |  |
| --- | --- |
| PTC Product | Version |
| Thingworx Platform | 8.5.x |
|  |  |

Distribution List

This section lists all receivers of this document. Solution Design Document does not require sign-off as it is provided as guidance to the customer. Solution Blueprint document sets the outline of the future architecture and is a live document that will continue to capture the Solution Specification in greater detail as the Solution Development proceeds.

|  |  |  |
| --- | --- | --- |
| Name | Company | Title / Role |
| Ankit Pandya | PTC | Technical Lead/Architect |
| Ajay Valvi | PTC | Customer Success Manager |
| Ketan Magdum | PTC | Technical Lead/Architect |
| Kiran Kumar Jasti | ELGI | Project Manager |
| Kalimuthu C | ELGI | Lead Specialist |
|  |  |  |

Table of Content

[1. Introduction 6](#_Toc34749308)

[1.1 Purpose 6](#_Toc34749309)

[1.2 Document Overview 6](#_Toc34749310)

[1.3 Reference Documents 6](#_Toc34749311)

[1.4 Definitions, Acronyms and Notation 6](#_Toc34749312)

[1.4.1 Definitions 6](#_Toc34749313)

[1.4.2 Acronyms 7](#_Toc34749314)

[2. Project Overview 8](#_Toc34749315)

[2.1 ELGi Overview 8](#_Toc34749316)

[2.2 Business Drivers and Project Goals 8](#_Toc34749317)

[2.3 Dependencies 8](#_Toc34749318)

[2.4 Assumptions 8](#_Toc34749319)

[3. Solution Design 9](#_Toc34749320)

[3.1 Data Model 9](#_Toc34749321)

[3.1.1 Component Classification 9](#_Toc34749322)

[3.1.2 Data Model Definition 12](#_Toc34749323)

[3.1.3 Model Hierarchy 14](#_Toc34749324)

[3.2 Device Model 17](#_Toc34749325)

[3.2.1 Device Onboarding 17](#_Toc34749326)

[3.2.2 Device Connectivity 20](#_Toc34749327)

[3.2.3 Asset Management 21](#_Toc34749328)

[3.2.4 Change Management 22](#_Toc34749329)

[3.2.5 Device Decommissioning 22](#_Toc34749330)

[3.3 Alert Management 23](#_Toc34749331)

[3.4 Organization and User Setup Strategy 23](#_Toc34749332)

[3.4.1 Organizations and Business Units 23](#_Toc34749333)

[3.4.2 Visibility restrictions 24](#_Toc34749334)

[3.4.3 User roles 24](#_Toc34749335)

[3.5 System Integration 26](#_Toc34749336)

[3.5.1 LN/PLM Integration 27](#_Toc34749337)

[3.5.2 CRM Integration 27](#_Toc34749338)

[3.5.3 CCS Integration 29](#_Toc34749339)

[3.6 User Interface 31](#_Toc34749340)

[3.7 Security Model 31](#_Toc34749341)

[3.7.1 Platform Security 31](#_Toc34749342)

[3.7.2 Server Security 32](#_Toc34749343)

[3.8 High Level Architecture 32](#_Toc34749344)

[3.8.1 Functional Architecture 32](#_Toc34749345)

[3.8.2 Cloud Deployment 33](#_Toc34749346)

[3.8.3 Data Storage 33](#_Toc34749347)

[4. Solution Artifacts 35](#_Toc34749348)

[4.1 Use Case 1: Generic Services 35](#_Toc34749349)

[4.1.1 Snippet Details 35](#_Toc34749350)

[5. Appendix 37](#_Toc34749351)

[5.1.1 On Premise Architecture Considerations 37](#_Toc34749352)

[5.1.2 On Premise Targeted Architecture 37](#_Toc34749353)

# Introduction

## Purpose

This document describes the outline of the target architecture of the solution for ELGi. Described solution will be delivered to the Customer via Thingworx project.

The intended audience of this document is as follows:

* Project Managers – to understand the possible scope of the future project and the intra/inter-dependencies;
* Solution Architects and Technical Authorities – to understand the breadth and depth of the proposed solution, impact on the overall IT Landscape.

## Document Overview

This document is organized into the following sections:

* Section 1 provides an overview of the document itself, used terminology and glossary; it also provides the document life cycle description.
* Section 2 provides a summary of the project context, business goals of the project.
* Section 3 includes blueprint description. This chapter contains a catalog of the system components used in the solution and also describes the high-level architecture of the baseline state and the target state.
* Section 4 contains list of all the use cases proposed which will be implemented as part of this solution. At this stage only outline of work packages is provided.

The feature level technical design details will be expanded within the same document as the project development starts. This remains a live document until the end of the solution development. Changes to this document shall be monitored and version controlled.

## Document Lifecycle

## This document describes the outline of the target architecture of the solution for ELGi. Described solution will be delivered

## Reference Documents

Section below lists all the documents which are relevant for the audience of this specification.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Document name | Version | Date | Owner | Document description |
|  | AA\_Cloud Requirement\_finalDocument |  | <TBD> | ELGi | Provides reference to requirements documented by ELGi |
|  | Data Model Document | 1.0 | <TBD> | ELGi | Provides the data model specification and details for assets and its parameters. |
|  | Integration Document | 1.0 | <TBD> | ELGi | Provides the integration specifications for 3rd party integration with LN, CRM and CCS. |

## Definitions, Acronyms and Notation

### Definitions

|  |  |
| --- | --- |
| Term | Definition |
| Architectural View | The system is described using several architectural views, each addressing a specific set of business needs that must be satisfied in order for the system to be acceptable. |
| Asset (Device, Instrument, Connected Product, Remote Thing) | An asset is a tangible business object; the thing of value that is being tracked and/or monitored. An asset is an instance of the Thing Template (ThingWorx terminology). |
| Component | It is a collection of code and business configurations organized to accomplish a specific function or set of functions. |
| Data Item | A time stamped piece of information which is send from the asset to the platform. It can contain information describing operating environment or performance of the asset (CPU temperature, operating system name, GPS coordinates, velocity of the vehicle etc.). |
| Model Number | Model Number (also known as Model Name) defines “type” of the device. Model Number in conjunction with the Serial Number allows for precise identification of the asset in the platform. |
| Serial Number | Serial Number is an identifier of the asset. Serial Number in conjunction with the Model Number allows for precise identification of the asset in the platform. |
| Stakeholder | It is an individual, team, or organization with interests in, or concerns relative to, a solution. Stakeholders need to understand the decisions that form the basis for the architecture and the rationale for those decisions. |

### Acronyms

|  |  |  |  |
| --- | --- | --- | --- |
| Acronym | Definition | Acronym | Definition |
| ELGi | ELGi Equipments Ltd. | RA | Remote Access |
| AD | Active Directory | SDK | Software Development Kit |
| API | Application Programming Interface | SN | Serial Number |
| DI | Data Item | SSO | Single Sign On |
| EMS | Edge MicroServer | UC | Use Case |
| ID | Identifier | UI | User Interface |
| IoT | Internet of Things | MN | Model Number |
| LDAP | Lightweight Directory Access Protocol | OS | Operating System |
|  |  |  |  |

# Project Overview

The purpose of this project is to implement Thingworx solution for ELGi.

AIR~ALERT is an advanced data acquisition system to monitor the operation of air compressor systems. This system is also used for preventive maintenance and failure prediction for trouble free running of air compressors. This system has important components such as data acquisition hardware (Gateway), network, cloud-based application (AIR~ALERT) and Air compressors and auxiliary equipment. The data acquisition hardware will request and receive the inputs from various child devices like Neuron controllers, VFD, AI modules and Energy meters. etc. and stores locally for some period and sends the data to the cloud through Gateway in periodical interval as specified in the setting.

## ELGi Overview

ELGi Equipments Ltd, the leading air compressor manufacturer in India offers over 400 compressed air systems designed for all industrial needs.

ELGi is a global leader in providing sustainable compressed air solutions. ELGi designs and manufactures an extensive range of innovative and technologically advanced compressed air solutions for a variety of industry applications. Over the years, ELGi has earned worldwide accolades for designing customer centric, compressed air solutions that are sustainable, and help companies achieve their productivity goals while ensuring a lower total cost of ownership.

## Business Drivers and Project Goals

This section lists the main business drivers and project goals which ELGi Equipments wants to accomplish by executing the Thingworx project.

The main business drivers of this project are as follow:

* Offer end customers Value Added Services in the form of remote service, preventive maintenance, and instrument health monitoring;
* Understand how the instruments are used by the end customers via usage monitoring. This data can be used by the R&D department to improve future versions of instruments;
* Improve overall customer satisfaction with more accurate troubleshooting, reduced instrument downtimes and remote service when appropriate.

## Dependencies

Project dependencies are captured below.

|  |  |
| --- | --- |
| ID | Dependency |
|  | None |

## Assumptions

Project assumptions are captured below.

|  |  |
| --- | --- |
| ID | Assumption |
|  |  |

# Solution Design

This section of the document provides information about the solution delivered by the project – lists of components and systems involved in the solution, current as-is architecture and the target architecture.

For a long running, multi-phased projects, a transition or series of transitions architectures are included in this chapter as well.

Architecture description includes following views where applicable:

* Component view – to illustrate systems and components and main interfaces between them;
* Functional view – to illustrate clusters of functionalities and delivering components;
* Deployment view – to demonstrate physical setup of the solution.

## Data Model

This section enumerates the data modeling concepts, classification and categorization of data models that contribute to the Things in the system corresponding to the physical devices in ELGi Ecosystem.

### Component Classification

The component classification discusses about the categorization of the devices based on the features, capabilities, and the data sets they carry. The table below categorizes the major set of compressor types that are available in the ELGi eco-system which can be logically separated from each other in order to represent as a different entity type on Thingworx.

|  |  |  |
| --- | --- | --- |
| Compressor | Type | Description |
| EPSAC | C | The most basic compressor type. |
| CV | A variant of compressor with VFD. |
| CD | A basic compressor type with dry air. |
| CVD | A compressor variant with VFD and dry air. |
| OFSAC (Oil Filter) | OW | A variant of double stage compressor. Can be categorized further to C and CV variant. |
| OA | Another double stage compressor variant. Can be categorized further to C and CV variant. |
| OLW | Single stage Oil based compressor variant. Can be categorized further to C and CV variant. |
| AB Series | C | The most basic compressor type. |
| CV | A variant of compressor with VFD. |

The system also consists of different sets of parameters based on the Neuron Controller type that is fitted on the compressor. At the Thing model level, any combination of the Compressor types and controller can be represented on the data model level. The following table represents the types of neuron controller currently being used in ELGi eco-system.

|  |
| --- |
| Neuron Controller Type |
| Neuron 3 |
| Neuron 4 |
| Neuron NXT |

The following is graphical representation of compressor classifications in ELGi ecosystem.

**EPSAC**

Note: Winding temperature is currently not available in C variant, but it might be available in the future variant. The data model will be modified accordingly in the future template versions.

Also, C variant is further classified into low-pressure and high-pressure variant based on the fact weather sump pressure is available (HP) or not (LP).

Similarly, the VFD variant is also further classified based on weather winding temperature is available (CV20) or not available (CV40).

**OFSAC**

Double Stage Compressor

Single Stage Compressor

**AB Series**

C variant is further classified into high-pressure variant where the sum pressure is available.

Similarly, the VFD variant is also further classified based on weather winding temperature is available (CV20) or not available (CV40).

### Data Model Definition

A Data Model creates a uniform representation of all items that interact with one another. There are multiple benefits to such an approach. The ability to break up items and reuse components is considered standard best practice. ThingWorx has adopted this model at a high level to represent individual components of an IoT solution.

Building an IoT solution in ThingWorx begins with defining your data model, the collection of entities that represent your connected devices, business processes, and your application.

Below diagram shows what a specific Inheritance Model might look like for a compressor Thing in ELGi eco-system.

**Master Thing Templates:**

The following base Templates will be available that would act as the baseline for all the standard classifications available.

There will be several other templates available in the system that are defined in the next section of model hierarchy. However, they all will be following the baseline capabilities defined for a device model using the below mentioned templates.

ELGI.MASTER

ELGI.MASTER.EPSAC

ELGI.MASTER.OFSAC

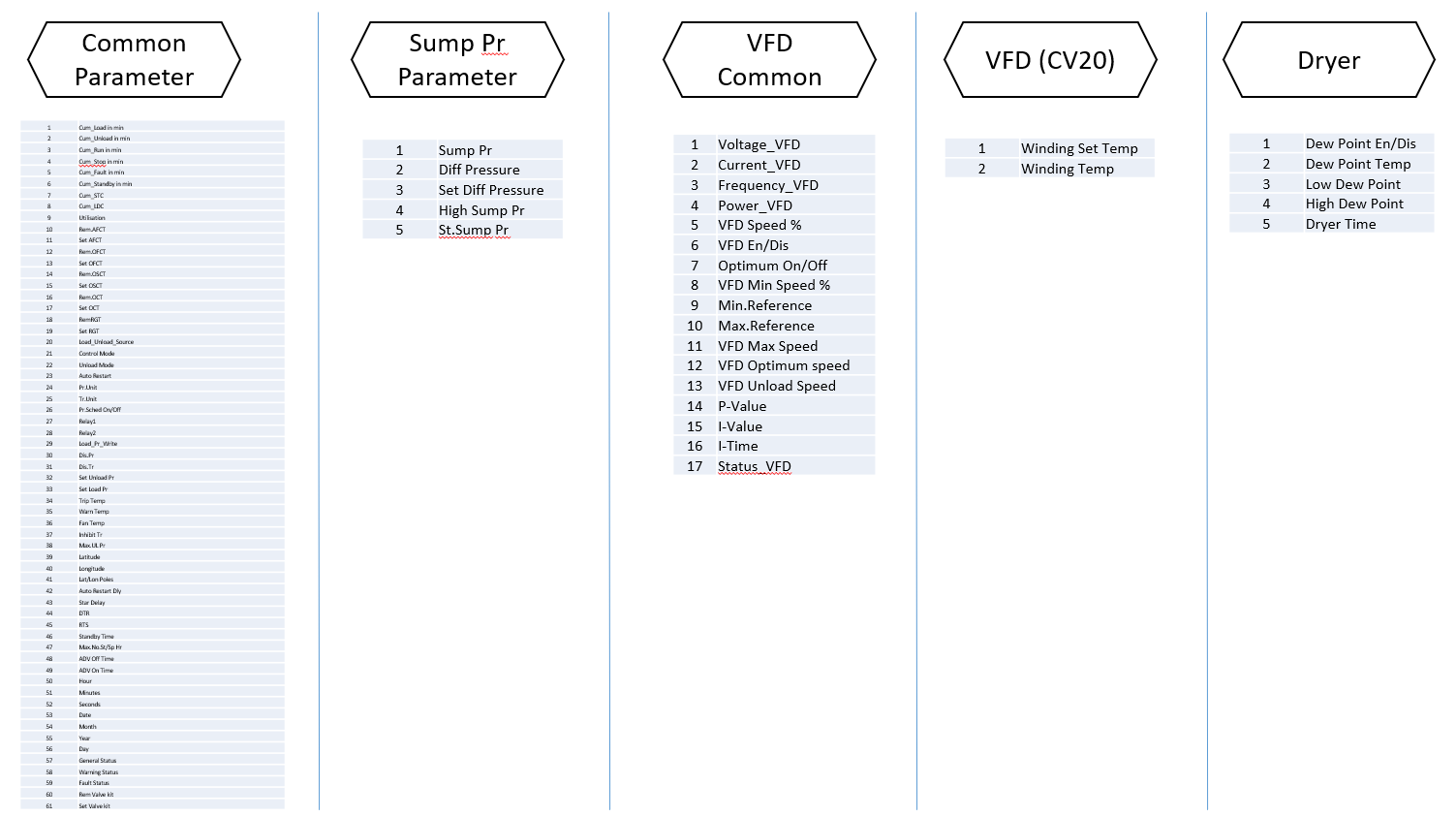
ELGI.MASTER.AB

**Thing Shapes:**

The following standard thing shapes will be available which utilizes the unique set of parameters being utilized across the several variants and classifications.

ELGI.

Calculated.TS



Neuron.TS

Neuron NXT.TS

Neuron 4.TS

Neuron 3.TS

**Data Model Breakdown:**

Once we have defined the components of Things, Thing Templates and Thing Shapes, the below diagram shows what a specific inheritance model looks like.

ELGI.MASTER.<CType>TT

ELGI.

Calculated.TS

Neuron<Type>.TS

Elgi.Compressor<UID>.Thing

Thing Template

Thing Shape

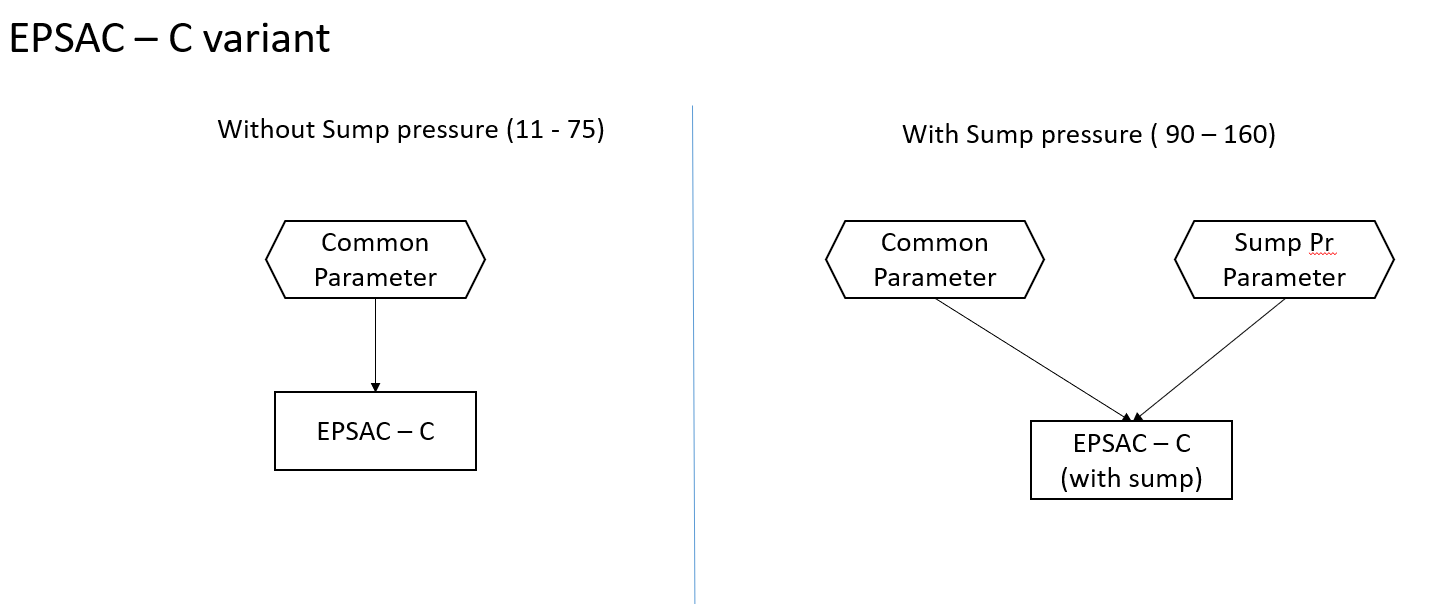
Thing

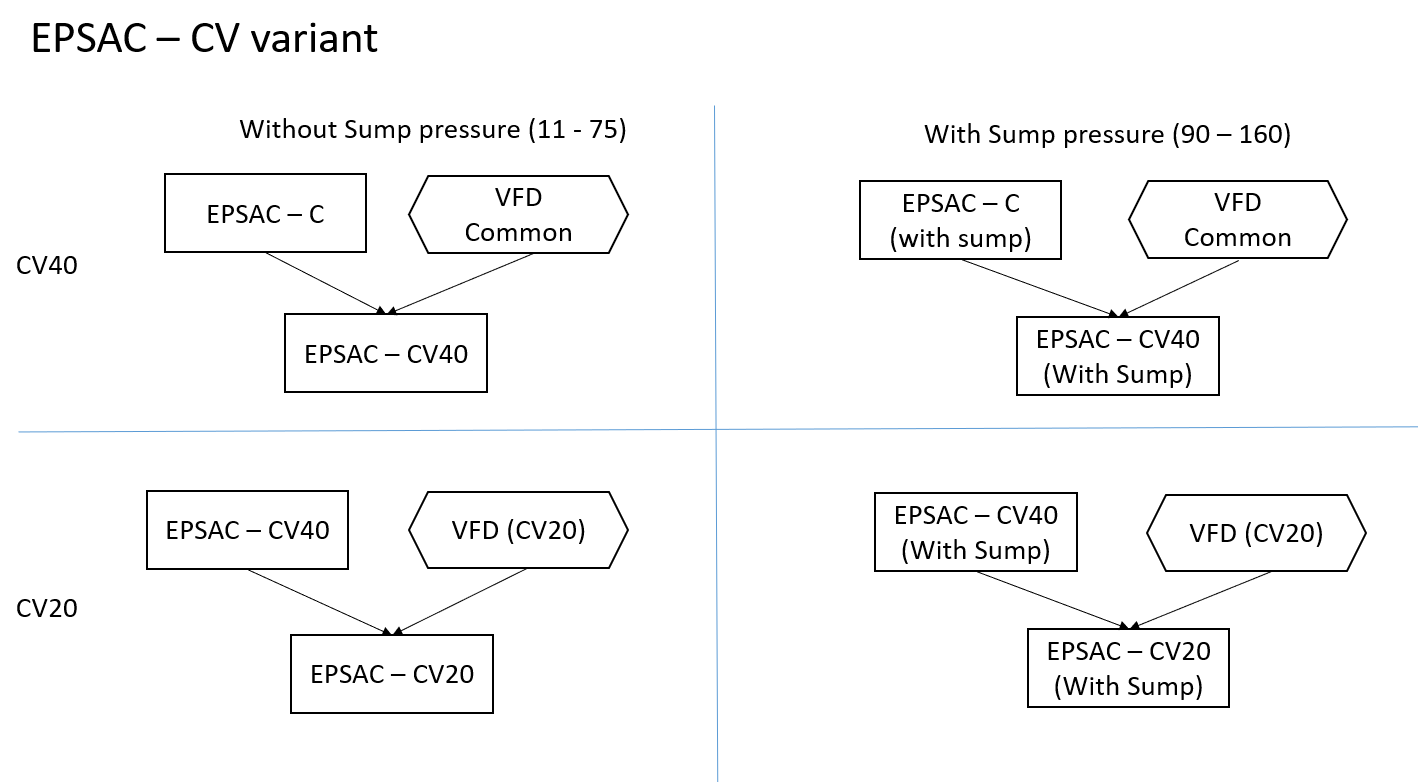
### Model Hierarchy

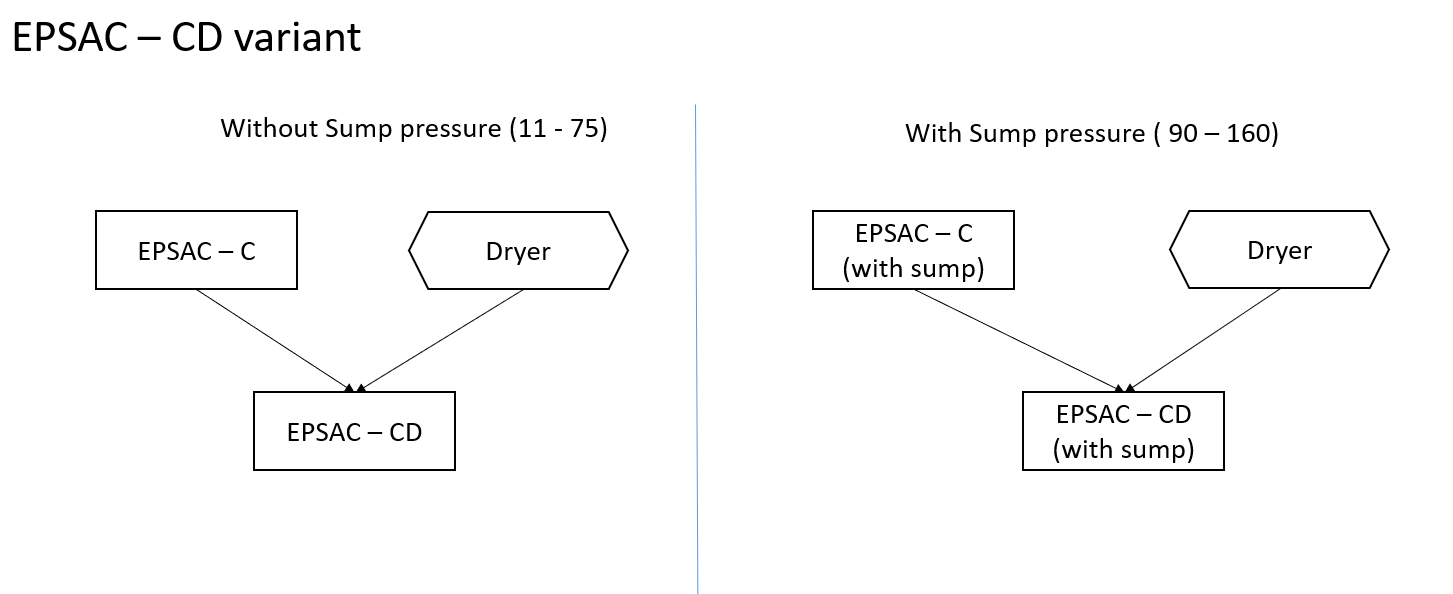
The model hierarchy defines representational hierarchy of entities and models defined in the system. As defined in the previous section, the primary or the master models (Templates and Shapes) will be utilized and respective entities will be created and used to build the actual data model upon.

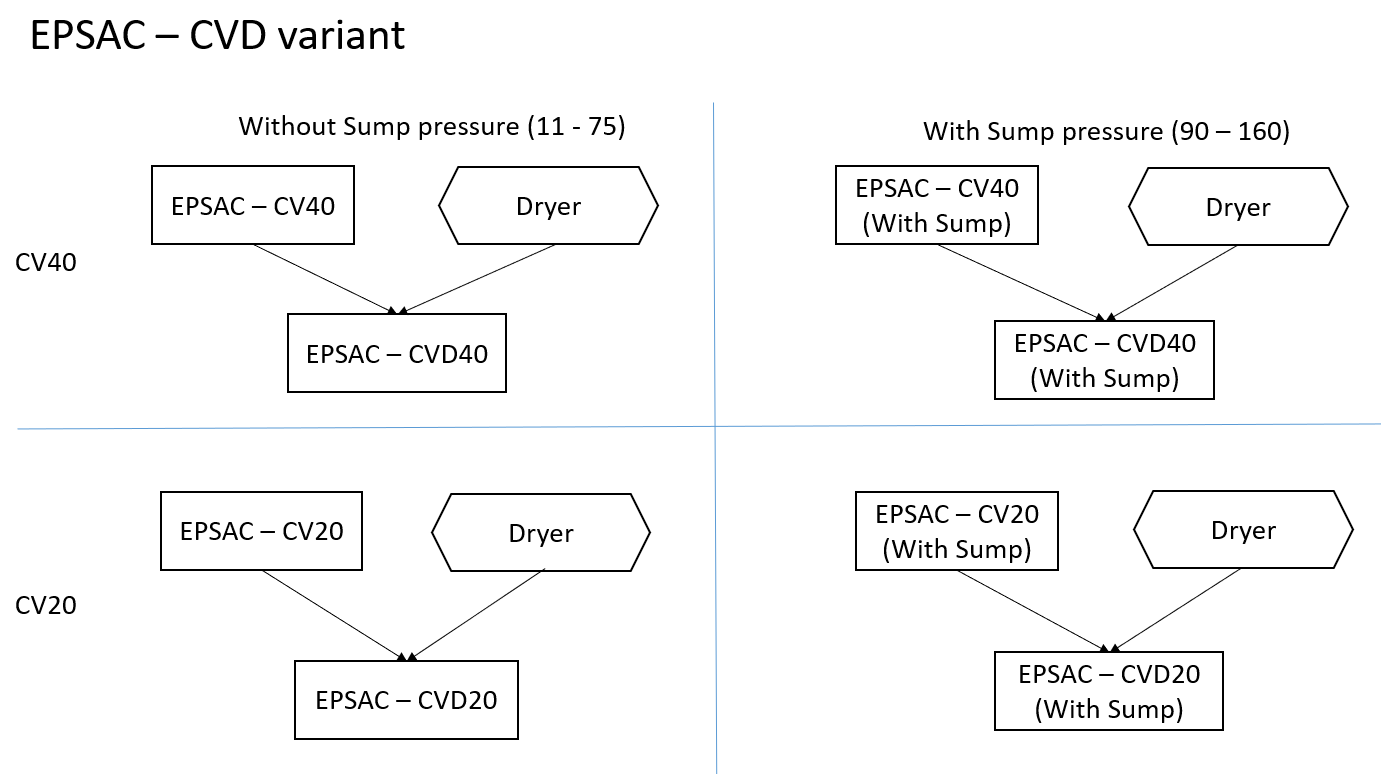
Following is the categorization of each of the data model and end representation of assets in the system. The categorization is based on the type of variants each compressor types will have in the system.

1. EPSAC

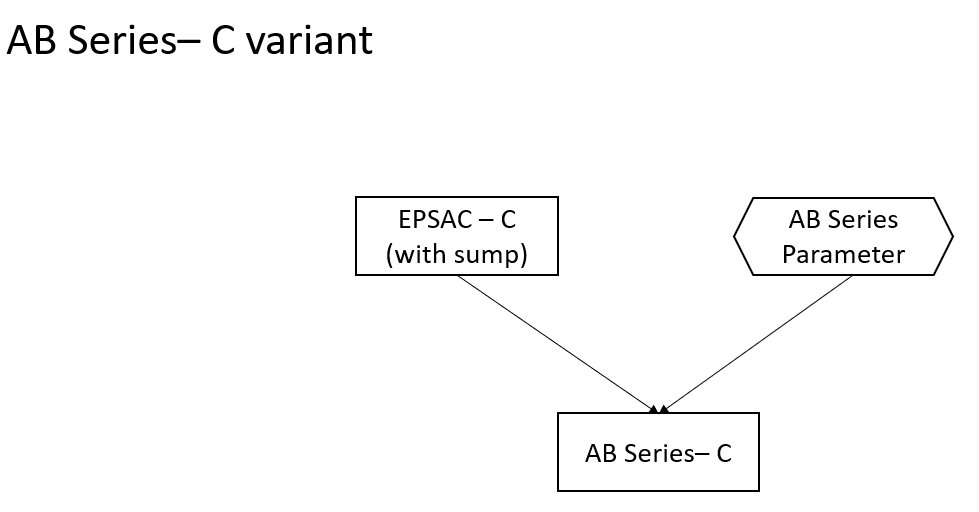


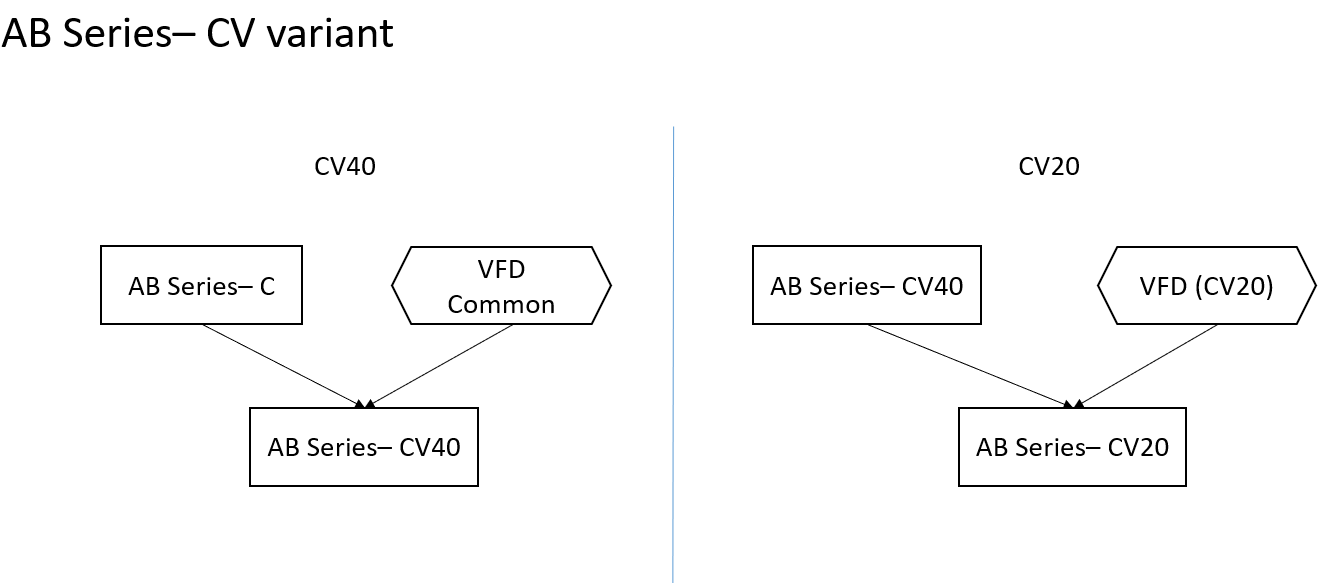






2. AB Series





3. OFSAC

ELGi to provide information.

A picture containing flower, bird

Description automatically generated

A close up of a logo

Description automatically generated

A close up of a map

Description automatically generated

A picture containing bird, flower

Description automatically generated

A close up of a map

Description automatically generated

A close up of a map

Description automatically generated

## Device Model

This section describes the architecture of a Smart Connected Product (SCP) IoT edge solution for ELGi.

The functional capabilities that are expected from the device platform connectivity module is:

* To be able to provide device onboarding capabilities.
* To be able to provide device connectivity and security.
* To be able to provide asset management and remote monitoring capabilities.
* To be able to perform Change management related operations.
* To be able to provide Health monitoring and diagnostics at asset level.
* To be able to perform device decommissioning.

The app-key based authentication will be used for device to talk to the platform. The device will have an app-key burned to it at the time of manufacturing which will be validated at the time of requesting the socket handshaking.

The solution proposed will be using C SDK module deployed on the client gateway (based on Linux operating system) to be able to communicate to the Thingworx platform.

### Device Onboarding

Device onboarding will be a sequential process consisting of the following steps.

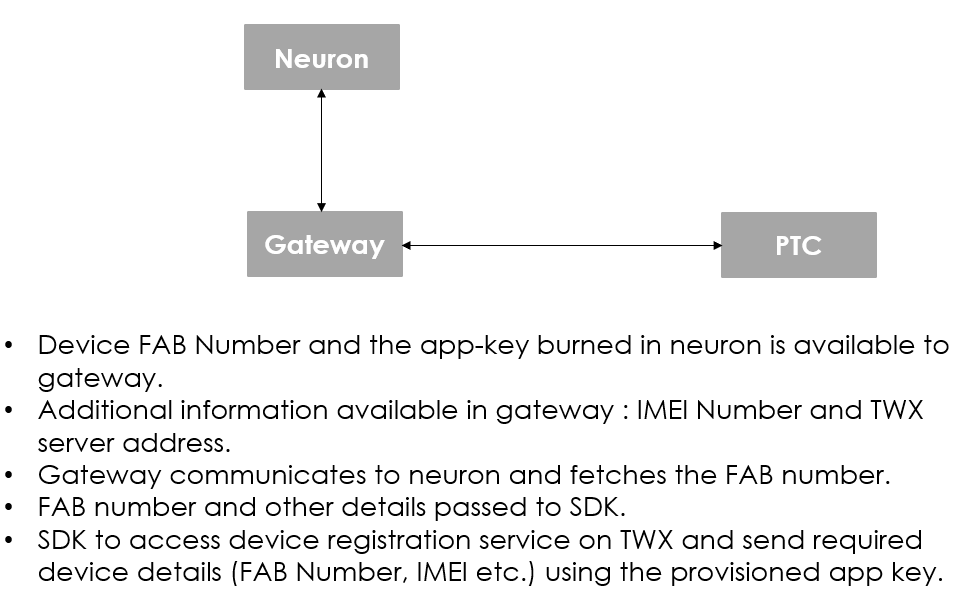
1. Device Registration
2. Device Configuration
3. Model Creation

The connectivity software, C SDK will be installed on the devices in the factory. ELGi team will be responsible for the installation and maintenance of devices, gateways and softwares installed on it, including the Edge SDK (C SDK in this case).

#### Device Registration

The first step in process of onboarding a device is when the gateway and device are switched on and it tried to communicate to the Thingworx server using the pre-burned app key.

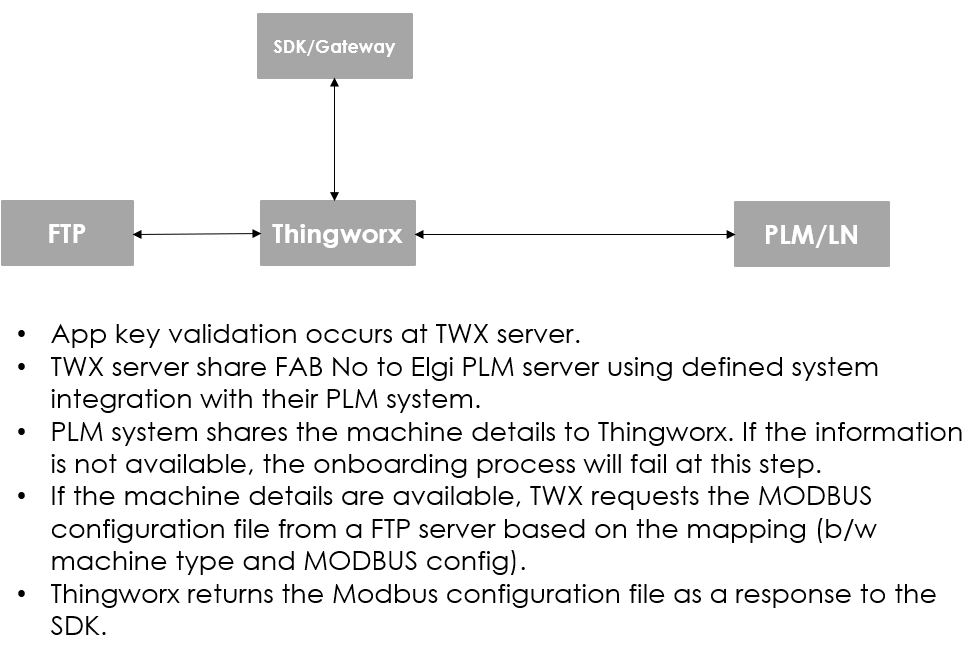
The following is the flow of device registration.



#### Device Configuration

Once the registration request is received at Thingworx, it performs the device configuration process using the wrapper services created at platform level.

The following flow is being executed.



Service/Model for this step is defined in the Model/Code Snippet in Section 4 of this document.

The following information is received from the PLM system when it is requested for machine details by passing the FAB number of the device.

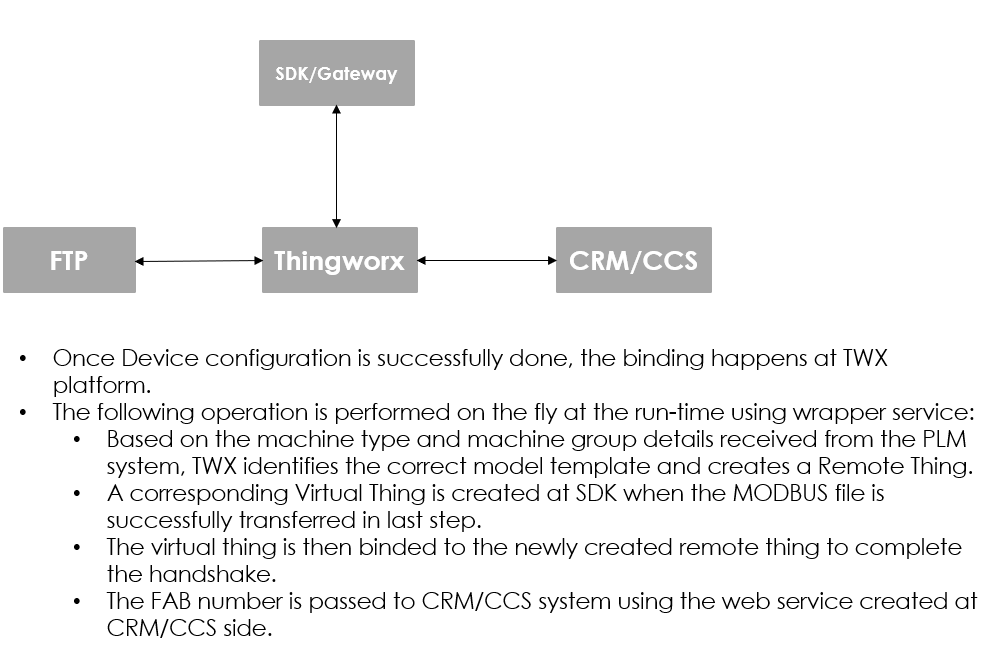
|  |  |
| --- | --- |
| Input to PLM/LN | Output information from PLM/LN |
| FAB Number | Machine FAB No |
| Machine group details (EPSAC /LEPSAC / DPSAC / Oil free etc.) |
| Machine Type |
| Mfg. Date |
| Name plate details |
| Rated CFM |
| Rated Pressure |
| Rated power |
| VFD or Non – VFD |
| AMC or Non AMC customer |
| Next AMC visit date |

For details on the parameters, please refer the “Parameter mapping worksheet”.

#### Model Creation

Once the device registration and device configuration is successfully done, the next step of the process is to successfully create the data model corresponding to the remote thing in action.

Following is the flow of model creation when device configuration is completed.



For details on the wrapper service and Service Thing on model creation, please refer to Section 4 of the document.

Once the handshaking is done, and the device connection is established, the virtual thing on SDK has to be updated through a firmware update to accommodate additional parameters derived from machine type selection.

### Device Connectivity

Once your connection is alive and active, any requests made to the server for registered properties and services will automatically be forwarded to your application, and the appropriate callback function will be called. To push properties to the server, execute a service on another entity in the system, or trigger an event on the server. Helper functions are available for these actions.

Assuming all the proper visibility, permissions, and other security aspects are correct, an entity built using the C SDK can read or write properties, create a list of subscribed properties, set values of subscribed properties, invoke services, and trigger events on itself or other entities in the system. The following sections describe the helper functions.

Reading a property

This helper function retrieves the current value of a property of a specific entity on ThingWorx platform.

Service: twApi\_ReadProperty ()

Please refer to section 4 for the snippet and code support.

Writing a property

This helper function writes a new value for a property of a specific entity on ThingWorx platform.

Service: twApi\_WriteProperty ()

Please refer to section 4 for the snippet and code support.

Push Properties

This function will be used to update one or more properties with a single message to ThingWorx platform. You can also use it to send multiple values of the same property to ThingWorx platform in a single message.

Service: twApi\_PushProperties ()

Execute a service

This helper function executes a service on a named entity on ThingWorx platform.

Service: twApi\_InvokeService ()

Triggering an event

This helper function triggers a specific event on a named entity on ThingWorx platform.

Service: twApi\_FireEvent ()

The data format coming from the gateway <TBD>

The data from C SDK will be transmitted on change. The SDK Subscribed Properties Manager (SPM) will be configured to only maintain changed values.

There are certain parameters which will not be changing frequently, or are not needed at a frequent interval, cacheType of “1 hour” and “24 hour” will be applied to these parameters to ensure the cached content in Platform is referred to until the cache timeout is achieved.

Additionally, store and forward will be implemented for it,

### Asset Management

Provides the capability of viewing the asset details, perform monitoring and remote operations on the selected asset/device to stream and view the live device data and calculated operations/information on them.

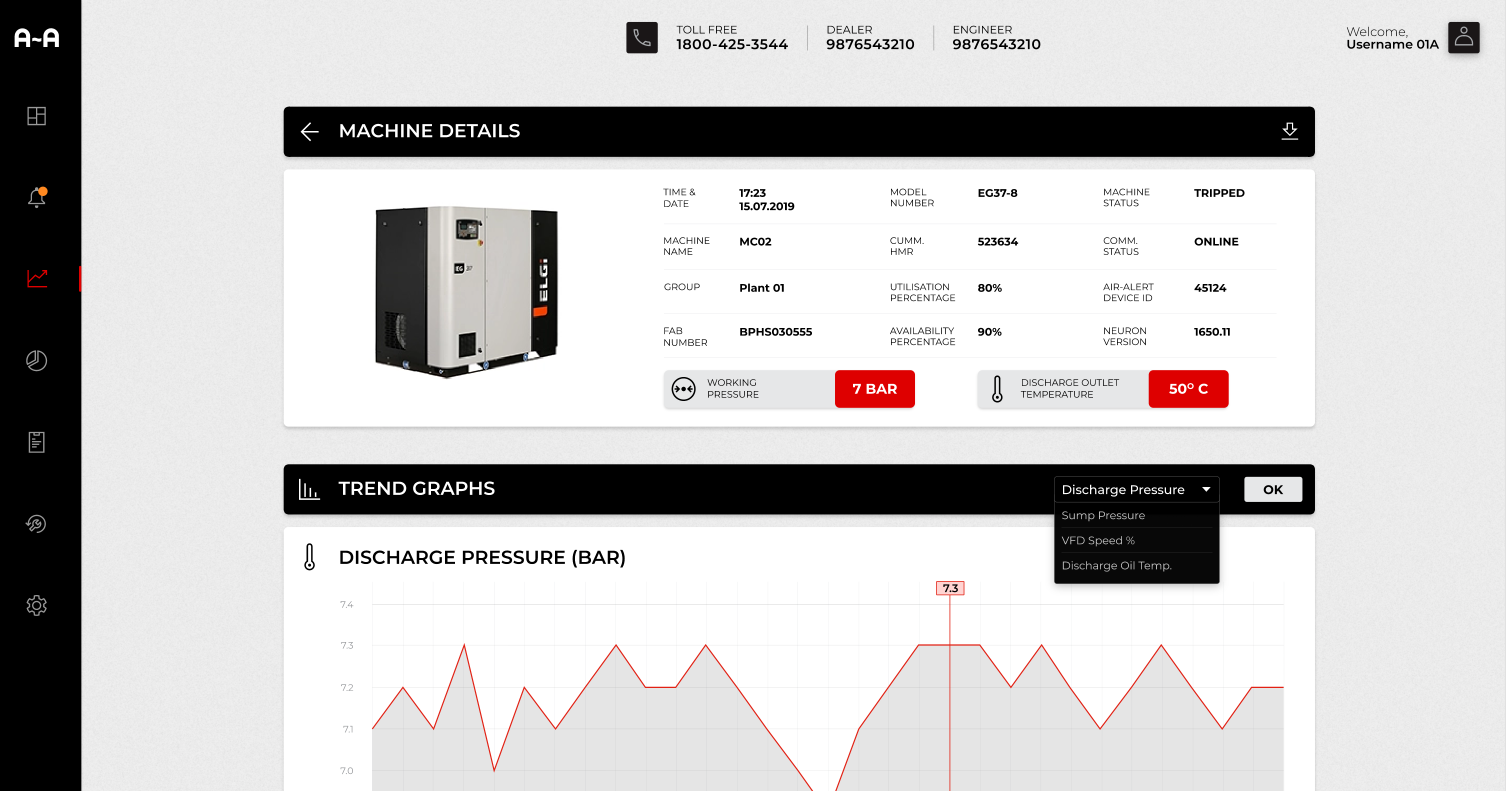
The asset management capability will allow users to view all the information related to a device on user screen, execute services and events associated to them, create and manage alerts on it and interact with the devices using remote operations.

Each asset represents a RemoteThing corresponding to the live compressor type that is commissioned at a site (internal or external) connected to the Thingworx system.

Each asset will have a visual representation and set of services associated to it within the RemoteThing itself, as well as through additional wrapper things interacting with the device and the system.

For more details on the snippet the model and services, please refer to Section 4.

The below screen represents Asset details page:



### Change Management

<TBD> Standard Software content management utility to be deployed.

No customization on the OOTB utility.

### Device Decommissioning

Device decommissioning capability is associated to each Remote Thing.

This capability will be provided in one or more mashups defined in “Asset Management” section. When the user initiates device decommissioning, the following flow takes place:

* A wrapper service is called that:
  + Removes the access/permissions that the entity has on any of the system wrappers or other entities.
  + Deletes the data associated to the Remote Thing in Thingworx System.
  + Executes a remote service to SDK to invoke service for deletion of the data and information at the device side.
  + Removes the links and dependencies that the Remote Thing has in platform side.
  + Unbinding and deletion of the virtual Thing/Remote Thing.

Please refer to Section 4 for the service snippets associated to it.

## Alert Management

Rules will be either authored in JavaScript and or a combination of Analytical services. This will be a capability blending the Property transformation capability of TWX analytics and subscription-based capabilities in the TWX platform.

Each rule is a combination of filters, statistical calculations followed by mathematic or comparisons. Statistical calculations will be handled by Analytical services. Rest will be handled through JavaScript/ThingWorx Extension.

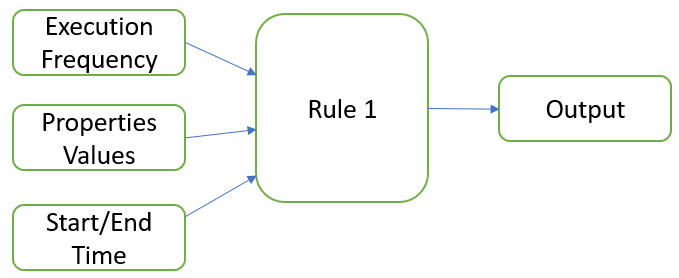
Below are design considerations drawn after initial discussion,

* A predefined structure for the Rules

e.g. *Median Discharge temperature > (median ambient temperature + Design limit) for 48 Hr where General Status is "Onload".*

In this scenario, two calculated parameters with a Threshold value is input and 48Hr is a timer.

* + - * Execution\_Frequency : 24h
      * General\_Status: Onload
      * Med\_Props: Disch\_Temp; Ambient\_Temp
      * Constant\_Properties *(From Things)*: Design\_Limit
      * AlertType: Email/SMS



* A DataTable will be maintained for each onboarded Thing which will store the Rule in below sample format,

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Name** | **Rule** | **Frequency** | **Data - *RuleResult* (*NO DS*)** | **DataShapeName** | **Assigned\_To** | **Status** |
| 1 | My\_Rule1 | Rule1 | Every\_1H | [object Object] | Rule1.DS | Thing 1 | ON |
| 2 | My\_Rule2 | Rule2 | Every\_24H | [object Object] | Rule2.DS | Template 1 | OFF |

* Predefined Timers will be created.
* Timer will have its predefined subscription either at ThingTemplate(TT) level or HelperThing(HT) level.
* A subscription will refer to a DataTable for the rule to execute. Respective timer rule will be executed.

## Organization and User Setup Strategy

### Organizations and Business Units

The organization will be created by the ELGi administrator to provision the devices under corresponding region. The visibility context of each of the assets will vary as per the business units in picture.

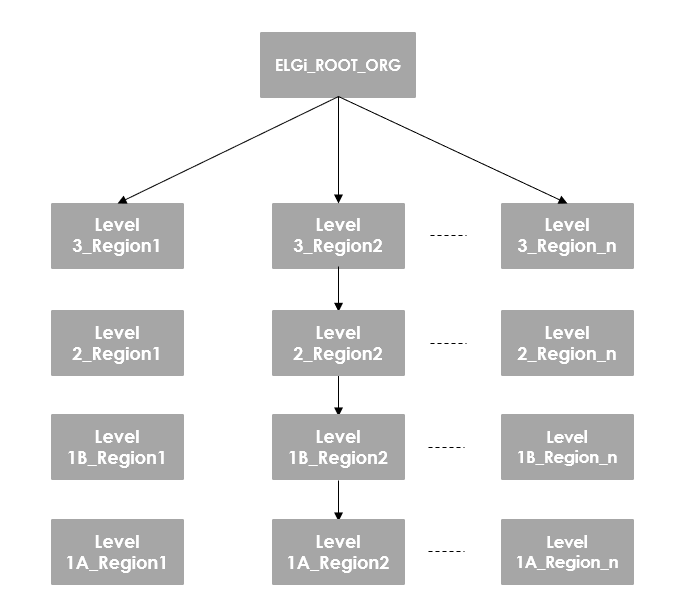
Level 1A – Lowest authority groups falling within this organization. Usually the end customer with minimum visibility privileges are a part of this sub-organization.

Level 1B – Applicable to Dealers, service engineers etc.

Level 2 – Applicable to Regional Sales manager, Area sales manager.

Level 3 – Highest authority groups falls within this organization. Country head for each country/region is mainly a user group contributing to this sub-organization.

The following will be the organizations applicable to the system.

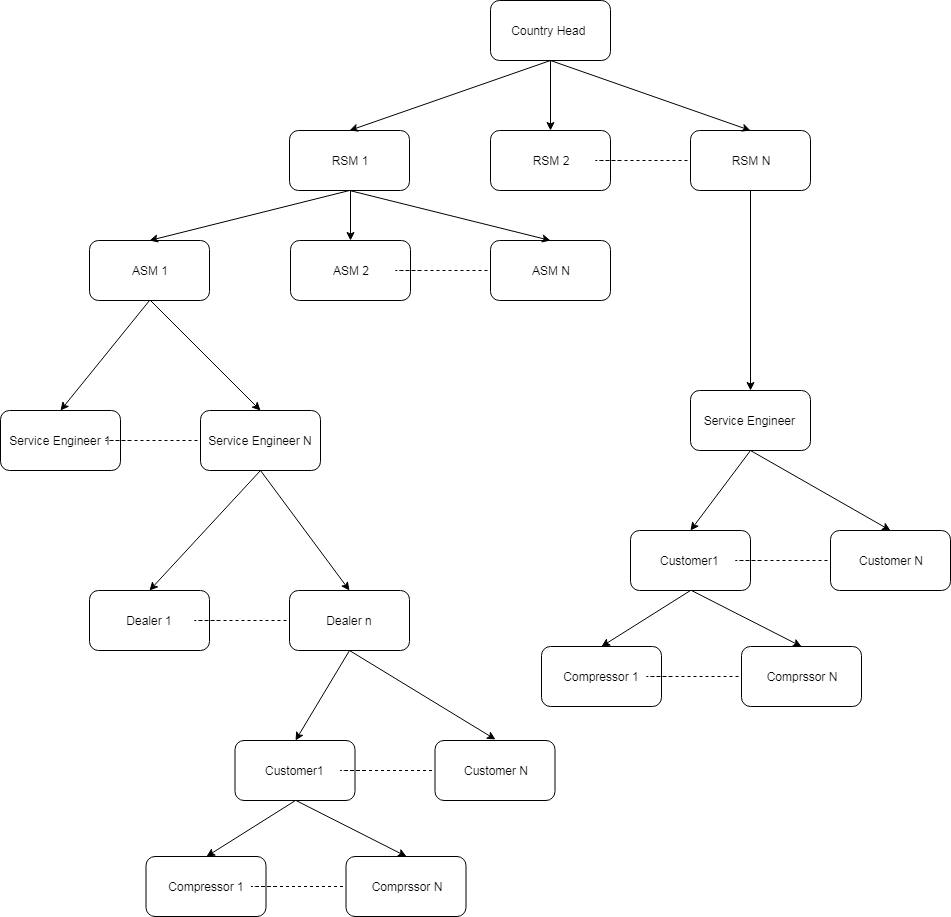


### Visibility restrictions

The visibility of entities will follow a bottom up approach, that means, a member of higher level sub-organization will have visibility to the entities defined at the subsequent lower level sub-org.

### User roles

The following user groups will be a part of the system.



These user groups will be specific to each organization and country. The ELGi administrator will have access to a screen that allows him to create and configure new/existing organization and user groups, and to associate users to each of these corresponding groups.

A compressor, or an asset will be linked to a customer/dealer/service engineer etc. and as per the user groups defined, the access to that asset will go up the chain.

For e.g. a Dealer will have access and visibility to all the assets that are available to all its customers. Similarly, a Country head should be able to see all the assets that are available in the country/region it falls in.

The customer information is received from CRM/CCS system and creates the user(s) that will be provided by the CRM. This user creation and association to relevant user group will be dynamic and the service for creation and configuration of the user will be triggered once the CRM/CCS system updates the TWX asset with relevant user details.

The CRM/CCS will share the following user information with TWX

|  |
| --- |
| Information from CRM/CCS |
| Customer Name |
| Customer mobile no |
| Customer email ID |
| Customer alternate mobile no |
| Customer complete address with country, region, area |
| Service Engineer/dealer mapped to machine |
| Service Engineer/Dealer mobile no |
| Service Engineer/Dealer email ID |
| ASM name/mobile no/email ID |
| RSM name/mobile no/email ID |
| Country Head name/mobile no/email ID |

The following mashups and Things will be created to perform tasks for user management.

Mashup = ELGi.UserManagement.Admin

Thing = ELGi.UserManagementWrapperThing

For more details on User Management snippets, please refer to Section 4.

## System Integration

There are three defined integration points for the platform. A separate connector extension is required to be created for each one of these integration operations. The three systems which platform will be connecting to are:

1. LN/PLM system
2. CRM system
3. CCS system

The following are the integrations that will be addressed in the development.

### LN/PLM Integration

Integrating into the machine data coming from the LN/PLM system to enable the streamline flow of information from LN/PLM to PTC.

The first condition that this integration addresses is providing the machine data based on the FAB number sent by Thingworx.

Use Case Detail 1:

Provide the machine data based on FAB number details.

**Business event:** Need to get the configuration details based the machine data associated to the FAB number.

**Preconditions**: FAB Number Exists

**Scenario:**

When PTC receives the FAB number from gateway, to pick the configuration details based on the machine data (An asset associated to the specific FAB number) that is available in LN/PLM system

**Sample Request/Response:**

**Request:** FAB number

**Response:**

{

"machine\_fab\_number":"",

"machine\_group":"",

"machine\_type":"",

"tpl\_code":"",

"manufacturing\_date":"",

"rated\_cfm":"",

"rated\_pressure":"",

"rated\_power":"",

"vfd\_or\_nonvfd":""

}

### CRM Integration

The CRM integration will facilitate the connector extension that allows the Thingworx system to communicate to the CRM system for fetching customer related information, and vice versa.

Use Case 1:

Send the FAB number to CRM once PTC receives machine data from LN/PLM system.

**Business event:** To intimate the CRM system about the asset enabled with air alert

**Preconditions**: Machine data available in LN/PLM

**Scenario:**

PTC is having the information about the assets which are air alert enabled, this information has to be send to CRM to maintain the air alert enabled system.

**Sample Request/Response:**

**Request:** FAB number

**Response:** status code

Use Case 2:

Send the customer details to PTC when machine is commissioned / during data change event.

**Business event:** Customer details will be filled in CRM system when the machine/asset is commissioned, which needs to be send to PTC.

**Preconditions**: Commissioned machine/asset FAB should be received from CRM

**Scenario:**

CRM system will not have customer details during testing phase and it will be filled in CRM when it is commissioned/assigned to specific customer/dealer.

**Sample Request/Response:**

**Request:**

{

“customerdata”:{

"customer\_name":"",

"customer\_mobile\_no":"",

"customer\_email\_id":"",

"customer\_alternate Mobile No":"",

"customer\_address":"",

"amc\_or\_non\_amc\_customer":"",

"next\_amc\_visit\_date":""

},

“fab\_number”:””

}

**Response:** Status code

Use Case 3:

Send the alert data to CRM if there is any abnormal detection.

**Business event:** Service request/ call need to be created based on the alert received from PTC

**Preconditions**: FAB number should be available in CRM

**Scenario:**

CRM system will create service request/call based on the alert that is send from PTC when there is any abnormal detection in specified parameters.

**Sample Request/Response:**

**Request:**

{

<Alert details> TBD

}

**Response:** Status code

Use Case 4:

Send the PTC customer login credentials to CRM when PTC receives customer information from CRM.

**Business event:** When PTC login is created for specific customer that is needs to be shared through email from CRM based on service engineer’s approval.

**Preconditions**: FAB number should be available in CRM with customer details

**Scenario:**

CRM should be the single point of contact to reach out the customer. Once PTC login is created for a specific customer that needs to send to CRM and it will be shared though email from CRM based on service engineer’s approval.

**Sample Request/Response:**

**Request:**

{

“fab\_number”:””

“username”:””

“password:””

"customer\_name":"",

"customer\_mobile\_no":"",

"customer\_email\_id":"",

"customer\_alternate Mobile No":"",

"customer\_address":"",

}

**Response:** Status code

### CCS Integration

The CCS integration will facilitate the connector extension that allows the Thingworx system to communicate to the CCS system for fetching customer related information, and vice versa.

Use Case 1:

Send the FAB number to CCS once PTC receives machine data from LN/PLM system.

**Business event:** To intimate the CCS system about the asset enabled with air alert

**Preconditions**: Machine data available in LN/PLM

**Scenario:**

PTC is having the information about the assets which are air alert enabled, this information has to be send to CCS to maintain the air alert enabled system.

**Sample Request/Response:**

**Request:** FAB number

**Response:** status code

Use Case 2:

Send the customer details to PTC when machine is commissioned / during data change event.

**Business event:** Customer details will be filled in CCS system when the machine/asset is commissioned, which needs to be send to PTC.

**Preconditions**: Commissioned machine/asset FAB should be received from CCS

**Scenario:**

CCS system will not have customer details during testing phase and it will be filled in CCS when it is commissioned/assigned to specific customer/dealer.

**Sample Request/Response:**

**Request:**

{

“customerdata”:{

"customer\_name":"",

"customer\_mobile\_no":"",

"customer\_email\_id":"",

"customer\_alternate Mobile No":"",

"customer\_address":"",

"amc\_or\_non\_amc\_customer":"",

"next\_amc\_visit\_date":""

},

“fab\_number”:””

}

**Response:** Status code

Use Case 3:

Send the alert data to CCS if there is any abnormal detection.

**Business event:** Service request/ call need to be created based on the alert received from PTC

**Preconditions**: FAB number should be available in CCS

**Scenario:**

CCS system will create service request/call based on the alert that is send from PTC when there is any abnormal detection in specified parameters.

**Sample Request/Response:**

**Request:**

{

"alert\_name":"",

"alert\_id":"",

"alert\_type":"",

"alert\_message":"",

"current\_value":"",

"previous\_value":"",

"alert\_condition":"",

"alert\_duration":"",

"timestamp":"",

"thing\_name":"",

"alert\_property":"",

“fab\_number”:””

}

**Response:** Status code

Use Case 4:

Send the PTC customer login credentials to CCS when PTC receives customer information from CCS.

**Business event:** When PTC login is created for specific customer that is needs to be shared through email from CCS based on service engineer’s approval.

**Preconditions**: FAB number should be available in CCS with customer details

**Scenario:**

CCS should be the single point of contact to reach out the customer. Once PTC login is created for a specific customer that needs to send to CCS and it will be shared though email from CCS based on service engineer’s approval.

**Sample Request/Response:**

**Request:**

{

“fab\_number”:””

“username”:””

“password:””

"customer\_name":"",

"customer\_mobile\_no":"",

"customer\_email\_id":"",

"customer\_alternate Mobile No":"",

"customer\_address":"",

}

**Response:** Status code

## User Interface

<ELGi> to provide with the user interface requirement document.

## Security Model

### Platform Security

#### Authentication and Authorization

The ThingWorx Platform supports HTTP authentication, which requires a user to establish a web session using a user name and password.

#### Encrypt Storage of all Sensitive Data

The ThingWorx Platform uses encrypted storage for sensitive data. Passwords are stored encrypted at all components within the ThingWorx solution. This includes passwords stored at the platform for user accounts, and passwords at the edge components for use in connecting back to the server.

### Server Security

#### Securing ThingWorx Platform

## High Level Architecture

### Functional Architecture

ThingWorx is a Java Web Application that runs inside Apache Tomcat and utilizes a modern, service based architecture. ThingWorx deployment involves the tiers mentioned below.

A screenshot of a cell phone

Description automatically generated

* Things/Devices: This layer contains the things, devices, and/or assets that will connect with, send data to, and receive content from your ThingWorx platform. In ELGi the Neuron controller shall communicate with the Edge SDK on the Gateway. The Edge SDK enables devices to securely communicate to the ThingWorx server. The Edge SDK can also be used for pre-processing of data to be moved to the edge.
* Users/Clients: This layer contains the products (primarily web browsers on Laptops and Desktops) that people use to access the ThingWorx platform.

|  |  |  |
| --- | --- | --- |
| OS | Browser | Version(s) |
| Windows | MS Edge | 44 and later |
| Mac/Windows | Firefox | Quantum – 59 and later |
|  |  | ESR – 60 and later |
| Mac | Safari | 11 and later |
| Mac/Windows | Chrome | 64 and later |

* Platform: The platform layer (or application tier) is where ThingWorx Core resides, which serves as the hub of your ThingWorx environment. The content from the connection layer is ingested, user requests from the client layer are answered, and content is analyzed to generate alerts, etc.

ThingWorx Analytics is a also part of the platform layer that is integrated with the ThingWorx core to serve the features required.

* Database: The database layer maintains two forms of data;
  + ThingWorx runtime model definitions and its persisted properties
  + Tabular type data that is persisted by the runtime model as rows of content in Blogs, Wikis, Streams, ValueStreams, and DataTables.

The core functional architecture for solution is as below.

The initial first year infrastructure is provided through PTC’s Cloud SAAS offerings for initial 500 assets and beyond

### Cloud Deployment

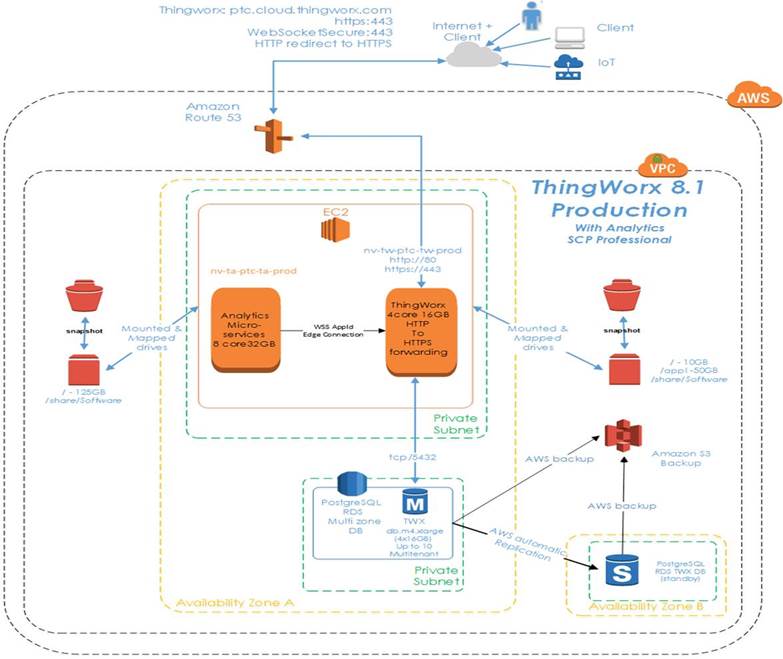
Following instances of the system will be deployed:

The current Infrastructure is provided by PTC over the cloud instance. ELGi plans to use own Cloud infrastructure in the future with an addition of Influx DB as a part of scaling requirement.

* Development (sandbox);(Captured in the email sent by PTCCloudServices Team to ELGi)
* Production.

A screenshot of a social media post

Description automatically generated



### Data Storage

The solution will be use following storage setup:

* Reference data: PostgreSQL (For the first year min 500 devices)
* Time series data: Influx DB (clustered | not clustered) (Eventually Influx DB will be used as a part of scaling)
* Data Retention and archival policy to be defined with ELGi, provided a separate database is provided.

# Solution Artifacts

Use cases are used to capture user-system interaction during business scenarios. As use cases are developed to describe the application behavior of the system and general sequence of events, the process description is elaborated to reflect user-system interaction at task level detail. Use cases and use case scenarios can then be applied for the purpose of application design and validation.

This chapter provides a list of all the Use Cases which are implemented by this project. Secondly, paragraphs below, describe in detail changes to each of the impacted systems. The structure of this section is use case biased, which means that each use case is listed as heading section with sub-sections for details of the implementation.

Please refer to the Development Plan for details on the service snippets.

## Use Case 1: Generic Services

This use case talks about the generic services that will be a part of the system. These services will be existing as the wrapper services which will be utilized for performing several operations not specifically binded to a particular asset.

### Snippet Details

Implementation of this Use Case will be realized by the following artifacts and work items:

#### Device Registration

|  |  |  |  |
| --- | --- | --- | --- |
| Template Name | Entity Type | Extends | User Access Restrictions |
| ELGi.DeviceRegistrationThing | Thing | Generic Thing | ELGi Admin |

The following are the service details:

|  |  |  |  |
| --- | --- | --- | --- |
| Service Name | Input | Output | Business Logic |
| RegisterNewDevice | Device FAB Number | Machine type details  (Infotable/DataShape) | This service takes the FAB number as an input parameter and queries the LN system of ELGi (configured as a web service request) to get the details of the corresponding device from LN. |
| QueryFTPforModbus | MachineType | Modbus configuration filename/location  (String) | This service provides the relevant Modbus configuration file (located in an FTP server) based on its mapping to the value of MachineType for the device. |
| RestartDevice | Device Identifier |  | Remote service for restarting SDK to reflect firmware updated properties. |
| UpdateSDKFirmware | FAB Number, machineType |  | Initiates the SDK firmware update using SCM utility. |

# Appendix

### On Premise Architecture Considerations

The initial first year the application is deployed and provided through PTC’s Cloud SAAS offerings for initial 500 assets or for additional assets incremented as required. After one year ELGi proposes to move on own cloud. Following are the architecture considerations. As devices grow in number, the ThingWorx solution grows in capacity and complexity, so does the architecture needs within each tier.

High availability solutions are important considerations for business continuation. High availability components need to be applied at the Ingestion Layer, Application layer and Database layer to be complete.

When hosted on-premise, high availability (HA) is critical to decrease the duration of outages for critical Internet of Things (IoT) systems. The ThingWorx High Availability (HA) configuration requires more servers in the application and database tiers, including supporting Apache ZooKeeper and pgpool-II nodes.

The ThingWorx Platform can run load balanced or on virtual servers to support high availability in case one server goes down.

Additionally, the PostgreSQL allows for high availability capability at the database level. It can be set up with a master and multiple slave nodes in the same or different availability zones.

For ThingWorx HA, Apache ZooKeeper is an additional required component. For the Database layer, the need for additional components depends on the need of the data provider.

Zookeeper - Apache Zookeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. It is a coordination service for distributed application that enables synchronization across a cluster. Specific to ThingWorx, Zookeeper is used to Coordinate the shift from the active ThingWorx Platform to the passive platform. It monitors the active server availability and will elect a new Core leader during a system failure.

Apache Ignite - Ignite is a distributed cache. It is used by ThingWorx to share state. It may be embedded with each ThingWorx instance or can be run as a standalone cluster for larger scale.

Database HA features

* PostgreSQL: ThingWorx provides the option to use PostgreSQL High Availability (HA) as part of the data solution. High availability offers the option to set up separate servers to capture reads and writes for data in the case of a failure on the primary server.
* InfluxDB: A time-series database ideally suited for high-scale ingestion of ThingWorx streams and value streams in development and production systems, alongside a relational database (Microsoft SQL Server or PostgreSQL) managing the ThingWorx data model. For the ThingWorx enterprise tests conducted in this sizing guide, the largest number of series, where each property instance gets its own series in the InfluxDB database, was 1,000,000.
* For example, if there are 10,000 Things, and each has 100 properties, then the database will create 1,000,000 property instances (10,000 × 100), reflected as 1,000,000 series within InfluxDB.



### Scalability Factors and On Premise Architecture Considerations

The following slide depict the Scalability factors and the Single Server Mode to HA transitions conditions.

A close up of text on a black background

Description automatically generatedA screenshot of text

Description automatically generatedA close up of text on a white background

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA close up of text on a white background

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedFollowing

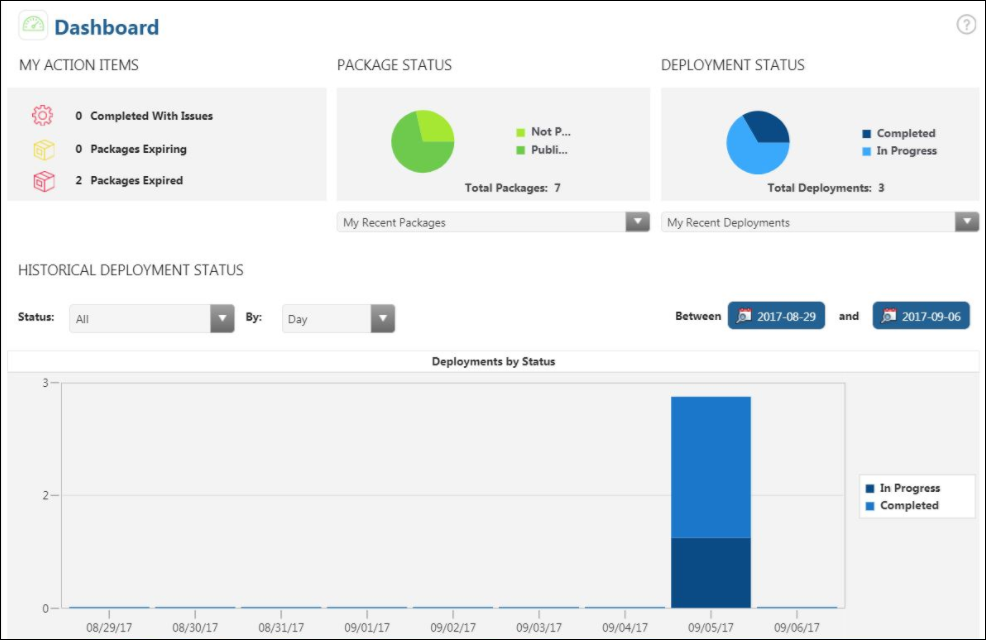
### Software Content Management:

Introduction:

ThingWorx Software Content Management provides tools that allow you to create, manage, test, and deploy packages. With this utility, you can create packages for immediate deployment or at a scheduled time in the future. You can also test packages and deployments before making them available in production. Search and auditing tools for packages and deployments are provided as well. For deployments, you can enable automatic retries when a deployment fails.

Features:

* Create and Search for existing software packages inside ThingWorx Platform
* Publish a package as a distinct version of a software
* Specify Access control permissions to a package
* Create, Schedule and test the deployment of a package to target assets
* Execute the deployment and transfer the software package to target assets
* Search for created, scheduled, completed and failed deployments
* Track a deployment by package or asset
* View Audit messages for package deployments
* Purge deployments from the system when they are no longer needed



**My Action Items:**

Displays the status of deployments. MY ACTION ITEMS lists the number of deployments that are in one of the following states:

Completed with Issues

Expiring

Expired

**Package Status:**

Displays the status for all packages created in the ThingWorx Software Content Management utility. It identifies the total number of packages created, and the number of packages that have and have not been published. To view more information about Published or Not Published packages, double-click the appropriate section on the pie chart.

To view details about a specific package, select it from the My Recent Packages list. The View Package Details page appears with more information about that package.

You can edit those packages that are not published. Unpublished packages cannot be used for a production deployment.

Published packages are ready for production deployment. You cannot edit or republish published packages. You can delete only those published packages that do not have active deployments.

**Deployment Status:**

Displays the status for all production deployments created in the ThingWorx Software Content Management utility. It displays the total number of deployments and displays a pie chart that illustrates the number of deployments in the Created, In Progress, and Completed states. Double-click an appropriate section of the pie chart to view more information for a represented status of deployment.

To view details about a specific deployment, select it from the My Recent Deployments list. The View Assets For Deployment page appears with more information about that deployment.

**Historical Deployment Status:**

Displays a graph that organizes historical deployments by status and date. You can filter the graph using the Status: or By: lists, or by specifying a date range using the Between calendar pickers above the graph. If you select All from the Status list, all statuses are displayed together in a stacked graph.

**Data History:**

Complete the following steps to configure automatic purge of deployments:

1. From the left pane in ThingWorx Software Content Management, under the Administration section, navigate to Configuration > Data History.
2. The Historical Data Configuration page appears.

From the Purge Deployments After list, choose one of the following options to select when deployments will be purged:

* Never
* 1 Day
* 1 Week
* 1 Month
* 3 Months
* 6 Months
* 1 Year

### On Premise Targeted Architecture [Maintained for Reference – 8.x Series Architecture]

Below is a Proposed Architecture for Every year ahead,

* Each ThingWorx is connected with 5 Connection Servers(CS) , each CS holds 10k web socket connections to compressors
* Each TWXI uses a Load Balancer to manage the communication with 5 CS and to allow edge apps to still send data if some CS die
* TWXI will use an internal DB (PostgreSQL) to hold the thing model and InfluxDB to store value streams
* Both TWXI and Asset Management Portal are ThingWorx installations in high availability mode
* To optimize database storage, the InfluxDB installation can be scaled to server multiple TWXI

|  |
| --- |
|  |
| A screenshot of a video game  Description automatically generated |
| A picture containing screenshot  Description automatically generated |
| A picture containing screenshot  Description automatically generated |
| A picture containing screenshot  Description automatically generated |
| A picture containing screenshot  Description automatically generated |
| Sizing notes for 10k to 200k scenario   * Split compressors by 50K chunks which means 4 TWX instances in HA for ingestion (4 alive + 4 sleeping) * 1 TWX HA (1 sleeping + 1 alive) instance for UI purposes to serve mashups to the users with stats about their account, traffic, cost, assets etc. This TWX will query the TWX ingestion server that manages the logged user * Each TWX ingestion will speak with 5 connection servers, each connection server will speak with 50K compressors things = 5 connection servers * 1 TWX admin for admin purposes to see system stats, connected users, problems and alerts and to allocate edges to CS load balancers as more edges on board and more CS are added until 200k assets * 1 Load balancer between each 50k edges and 5 Conn.Server = 4 Conn Server load balancers in total * 1 Load balancer between a TWX alive and TWX sleeping = 4 ThingWorx load balancers in total * 1 CAS (Ping Federate) with 1 IdP and 1 LDAP (for HA mode) * 5 Zoo keeper nodes to monitor all 10 TWX instances * 3 PostgreSQL server to host 5 thing model schemas each (HA) for 4 alive ingestion ThingWorx and 1 alive UI Thingworx * 3 PGPool hosted in TWX VMs sleeping or separately (small HW) * 1 file storage system to host repositories for all 5 alive TWX Instances * 1 archive DB to store value stream data older than 1 year |