WISE-PaaS 2.0 Integration

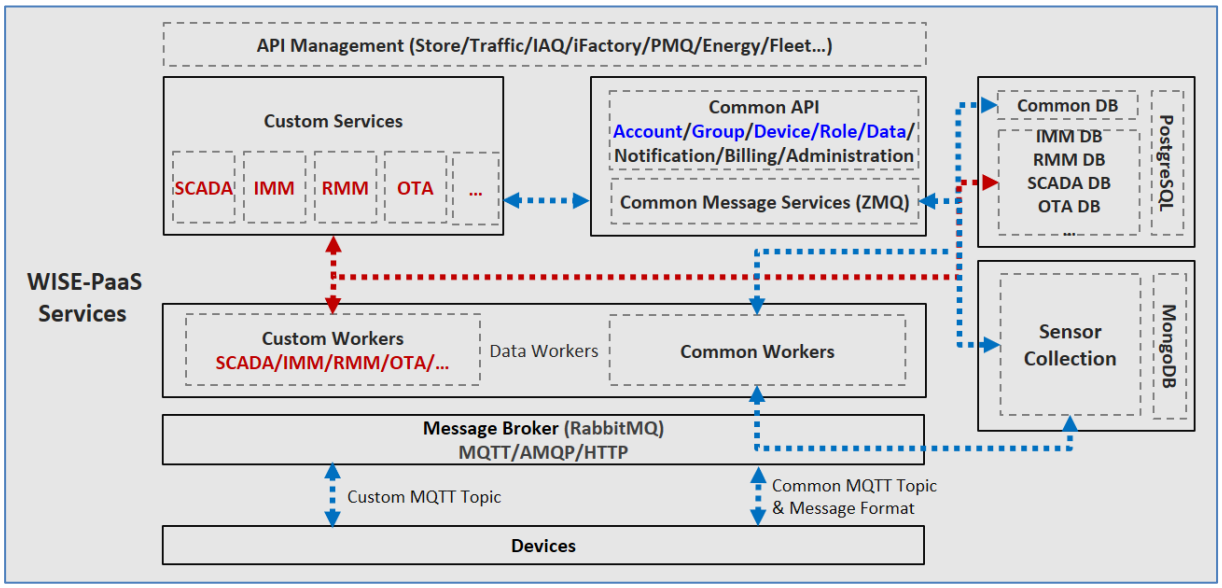
ADVANTECH

Enabling an Intelligent Planet

Table of Content

1. WISE-PaaS 2.0 Integration Components

1.1 Overview



1.2 Components Introduction

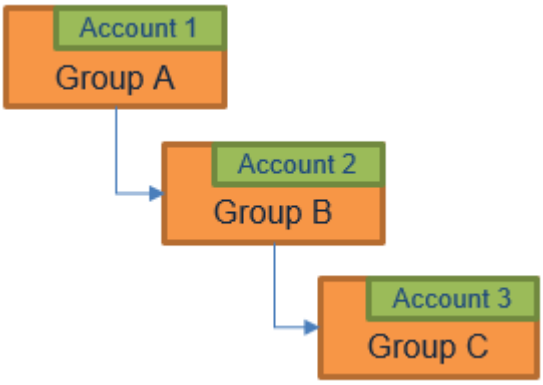
- API Management
  - ✧ Adopt Azure API Management/Open-source (WSO2) to integrate various SRP services API.
- Common Service APIs/Lib
  - ✧ Basic Account/Device/Group/Role operation APIs, including Add/Edit/Delete/Search/Advance Search
  - ✧ WISE-PaaS administration information (Product Support, version, capability…etc.)
- Custom/SRP Service
  - ✧ RMM/IMM/OTA/SCADA self-service APIs
- Common Data Workers
  - ✧ Including device basic information and sensor data from broker. (Common MQTT Topic)
  - ✧ For structure info (device Info) store into PostgreSQL
  - ✧ For non-structure (sensor Info) store into MongoDB/DocumentDB/...etc.
- Custom/SRP Data Workers
  - ✧ Process RMM/IMM/OTA/SCADA self-MQTT topic message and store into self-database (RMM/IMM/OTASCADA DB)

- Message Broker (RabbitMQ)
  - ✧ Adopt RabbitMQ as default cluster broker, and plugin MQTT as communication protocol.
- Device Agent
  - ✧ Define common MQTT topic handshake message and custom/SRP topic format.
- Database (SQL/NoSQL)
  - ✧ PostgreSQL as default relation database and including Common/RMM /SCADA/IMM/etc.
  - ✧ MongoDB/DoucmentDB for sensor information collect.

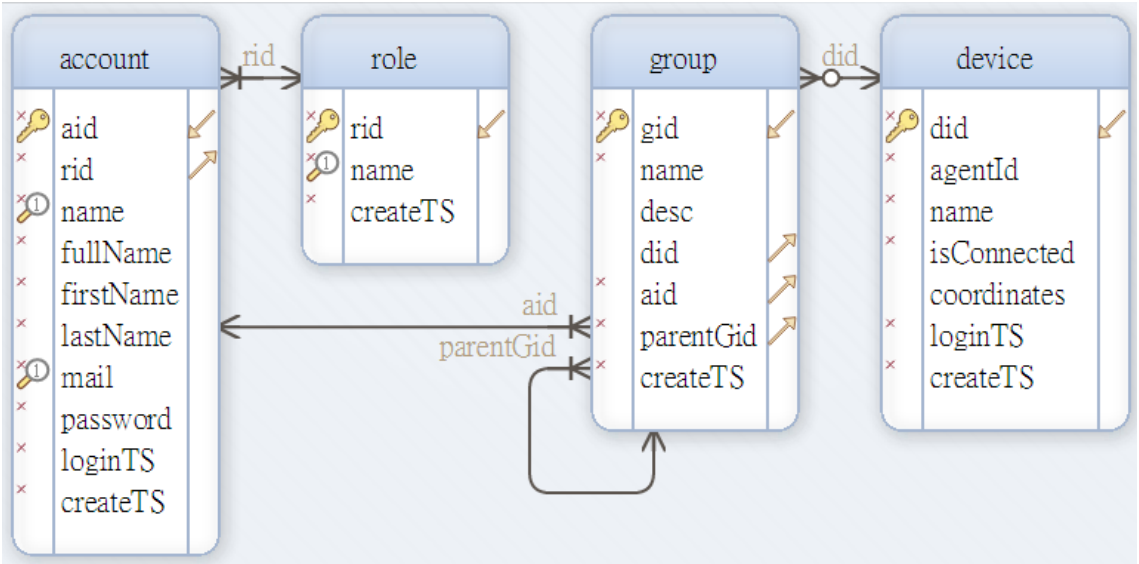
2. WISE-PaaS 2.0 Common Database Scheme

2.1 Concept & Design Rules

- ✓ Device belong to multi-groups or not to any group.
- ✓ Each account have non or multi-groups.
- ✓ Each group must belong to someone account.
- ✓ Group of group (1-3 Levels)
- ✓ Each account must belong to someone role.



2.2 Common Database Scheme Overview



2.3 Detail Common Tables Information

- Table Account

Columns			
*	aid	bigserial	Primary Key, Account Id
*	rid	bigint	Reference Key, Role Id
*	name	text	Unique Key, Account Name
*	fullName	text	Account Full Name
*	firstName	text	Account First Name
*	lastName	text	Account Last Name
*	mail	text	Unique Key, Account E-mail
*	password	text	Account Password
*	loginTS	timestamp	Account Last Login Timestamp
*	createTS	timestamp	Account Create Timestamp
Indexes			
Pk	PK_ACCOUNT_AID	ON aid	
U	UNI_ACCOUNT_MAIL	ON mail	
U	UNI_ACCOUNT_NAME	ON name	
Foreign Keys			
	FK_ACCOUNT_RID	(rid) ref role (rid)	

- Table Role

Columns			
*	rid	bigserial	Primary Key, Role Id
*	name	text	Unique Key, Role Name
*	createTS	timestamp	Role Create Timestamp
Indexes			
Pk	PK_ROLE_RID	ON rid	
U	UNI_ROLE_NAME	ON name	

- Table Group

	Columns		
*	gid	bigserial	Primary Key, Group Id
*	name	text	Group Name
	desc	text DEFAULT ''::text	Group Description
	did	bigint	Reference Key, For Device Id
*	aid	bigint	Reference Key, For Account Id
*	parentGid	bigint	Parent Group Id (Reserved)
*	createTS	timestamp	Group Create Timestamp
Indexes			
Pk	PK_GROUP_GID	ON gid	
Foreign Keys			
	FK_GROUP_AID	(aid) ref account (aid)	
	FK_GROUP_DID	(did) ref device (did)	
	FK_GROUP_PARENTGID	(parentGid) ref group (gid)	

- Table Device

Columns			
*	did	bigserial	Primary Key, Device Id
*	agentId	text	AgentId, UUID Format
*	name	text	Device Name
*	isConnected	bool DEFAULT false	Device Connect State
	coordinates	text DEFAULT '25.069439,121.582485'::text	GPS Coordinates, latitude andlongitude
*	loginTS	timestamp	Device Last Login Timestamp
*	createTS	timestamp	Devie Create Timestamp
Indexes			
Pk	PK_DEVICE_DID	ON did	

2.4 Non-Structure Data (MongoDB/DocumentDB)

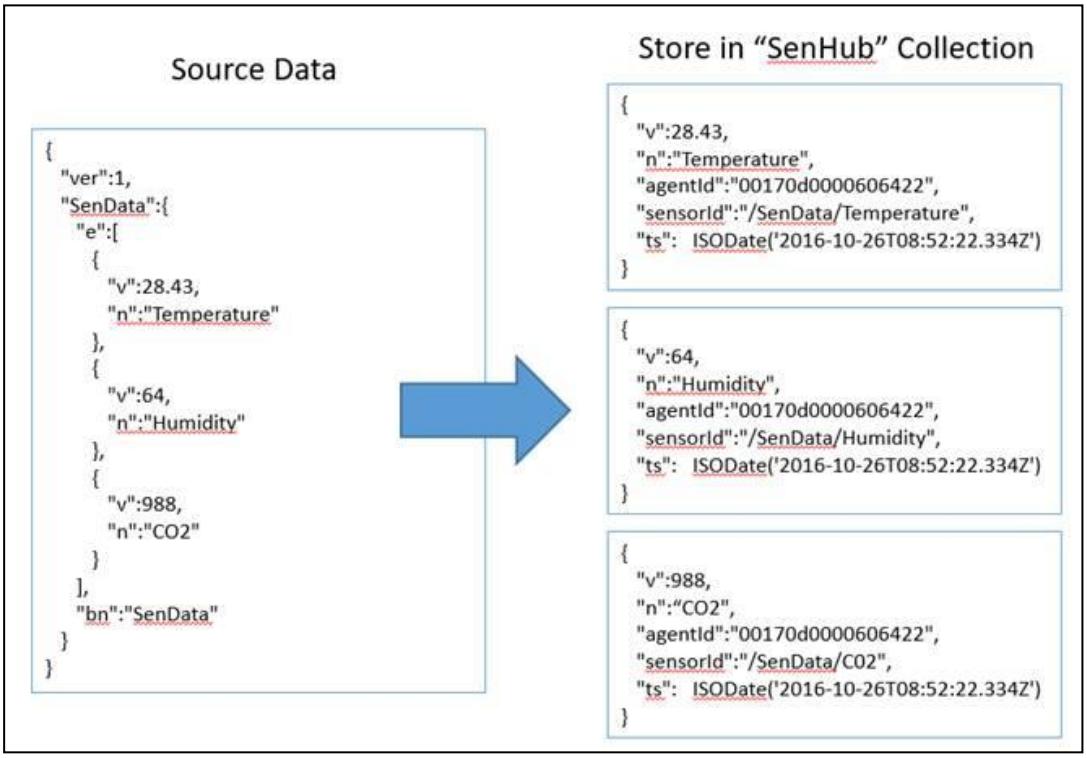
- Default Collections

- Event
  - WISE-PaaS platform event message
  - Format

```
{
    "product": "RMM",
    "type": "DEVICE",
    "subtype": "DEVICE_DISCONNECTED",
    "severity": "ERROR",
    "message": "DEVICE_DISCONNECTED",
    "extMsg": <JSONObject>,
    "device_id": 2,
    "device_name": "AC09",
    "account_id": 2,
    "account_name": "admin",
    "ts": ISODate('2016-11-29T05:13:46.759Z')
}
```

- Handler (Plugin)
  - Handler (Plugin) report data (source data)
- Capability
  - Handler capability structure
- Heartbeat
  - WISE-PaaS device's keepalive mechanism, ensure device connection state.

- Sensor Collections (Dynamic generate and transfer to key/value)
  - SUSIControl
  - HDDMonitor
  - Modbus
  - SenHub
  - IoTGW
  - …


Source Data
{
"ver":1,
"SenData":{
"e":[
{
"v":28.43,
"n":"Temperature"
},
{
"v":64,
"n":"Humidity"
},
{
"v":988,
"n":"CO2"
}
],
"bn":"SenData"
}
}
Store in "SenHub" Collection
{
"v":28.43,
"n":"Temperature",
"agentId":"00170d0000606422",
"sensorId":"/SenData/Temperature",
"ts": ISODate('2016-10-26T08:52:22.334Z')
}
{
"v":64,
"n":"Humidity",
"agentId":"00170d0000606422",
"sensorId":"/SenData/Humidity",
"ts": ISODate('2016-10-26T08:52:22.334Z')
}
{
"v":988,
"n":"CO2",
"agentId":"00170d0000606422",
"sensorId":"/SenData/CO2",
"ts": ISODate('2016-10-26T08:52:22.334Z')
}

3. API Management (Restful APIs)

3.1 Common REST APIs

3.1.1 Device APIs

- /wise-paas/**common**/device
  - <POST> – Add Device
- /wise-paas/**common**/device/<id>
  - <PUT> – Edit Device
  - <DELETE> – Delete Device
  - <GET> – Get device, account, group information.
- /wise-paas/**common**/device/advSearch
  - <POST> – Advanced search by each field.

3.1.2 Group APIs

- /wise-paas/**common**/group
  - <POST> – Add Group
- /wise-paas/**common**/group/<id>
  - <PUT> – Edit Group
  - <DELETE> – Delete Group
  - <GET> – Get group, account information
- /wise-paas/**common**/group/advSearch
  - <POST> – Advanced search by each field.

3.1.3 Account APIs

- /wise-paas/**common**/account
  - <POST> – Add Account
- /wise-paas/**common**/account/<id>
  - <PUT> – Edit Account
  - <DELETE> – Delete Account
  - <GET> – Get account and group information
- /wise-paas/**common**/account/advSearch
  - <POST> – Advanced search by each field.
- /wise-paas/**common**/account/login
  - <POST> – Account login, verify the password, and return access token.

3.1.4 Role APIs

- /wise-paas/**common**/role

- ✧ <POST> – Add Role
- /wise-paas/**common**/role/<id>
  - ✧ <PUT> – Edit Role
  - ✧ <DELETE> – Delete Role
  - ✧ <GET> – Get role information
- /wise-paas/**common**/role/advSearch
  - ✧ <POST> – Advanced search by each field.

3.1.5 Data Management APIs

- /wise-paas/**common**/data/history/<id>/<beginTS>/<endTS>/<handler>/<sensorId>
  - ✧ <GET> – Get row data, based on device Id, time range, handler and sensor Id.
- /wise-paas/**common**/data/realtime/<id>/<beginTS>/<endTS>/<handler>/<sensorId>
  - ✧ <GET> – Get real-time device data, based on device Id, time range, handler and sensor Id.
- /wise-paas/**common**/data/set/<id>
  - ✧ <POST> – Set device data to target device.
- /wise-paas/**common**/data/sensorId/<id>/<handler>
  - ✧ <GET> – Get sensor Id, based on device Id and handler.
- /wise-paas/**common**/data/startReport
  - ✧ <POST> – Set device start to report data.
- /wise-paas/**common**/data/stopReport
  - ✧ <POST> – Set device stop to report data.

3.1.6 NoSQL APIs

- /wise-paas/**common**/noSQL/collection
  - ✧ <POST> – Add new collection into noSQL DB, index options…etc.
  - ✧ <Delete> – Delete collection on noSQL DB.
- /wise-paas/**common**/noSQL/collection/data
  - ✧ <POST> – Add data into collection
  - ✧ <PUT> – Replace data on collection, via conditions.
- /wise-paas/**common**/noSQL/advSearch
  - ✧ <POST> – Advance search on collection and conditions.

3.2 Custom REST API Categories

3.2.1 RMM

- /wise-paas/**rmm**/apiInfo

- /wise-paas/**rmm**/dashboard
- /wise-paas/**rmm**/kvm
- /wise-paas/**rmm**/msgNotify
- …

3.2.2 IMM

- /wise-paas/**imm**/apiInfo
- …

3.2.3 SCADA

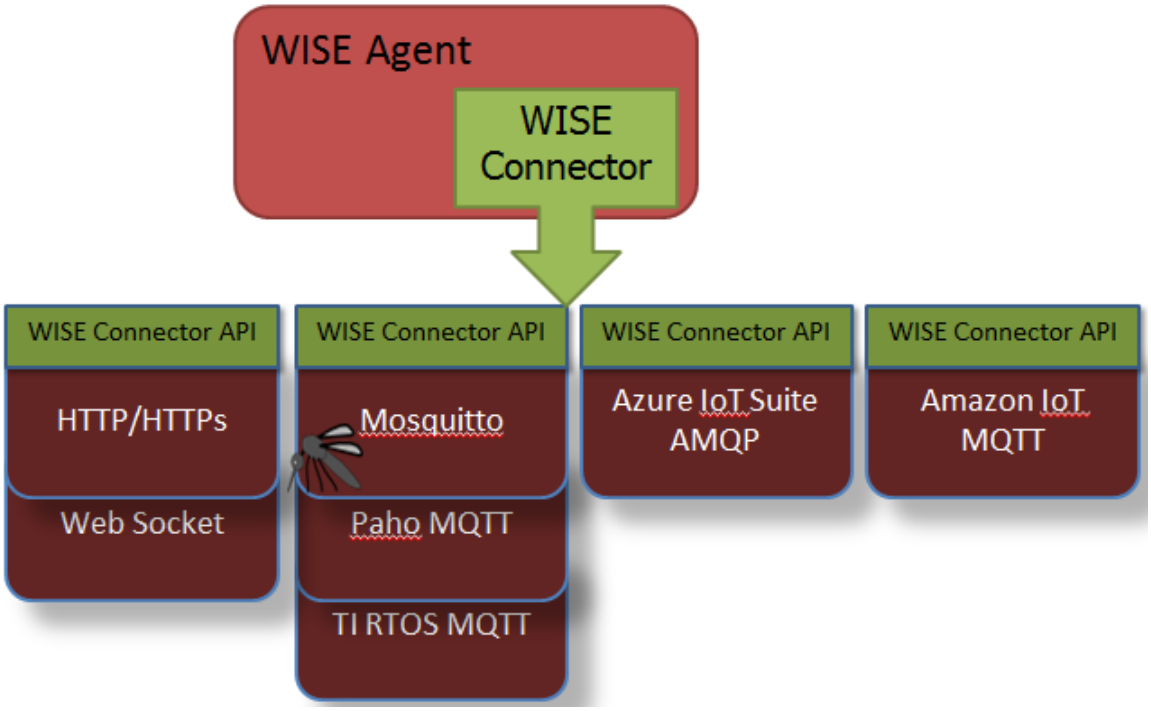
- /wise-paas/**scada**/apiInfo
- …

3.2.4 OTA

- /wise-paas/**ota**/apiInfo
- …

4. Device Connectivity

- Adopt open source RabbitMQ on server side to support both AMQP and MQTT protocol.
- Transportation supports **MQTT**, AMQP and HTTP (Websocket)
  - ✧ Define the common communication APIs for our WISE Agent, we called WISE Connector APIs
  - ✧ Implement the WISE Connector APIs with different communication protocol.
  - ✧ WISE Agent can link different communication protocol at compile time.

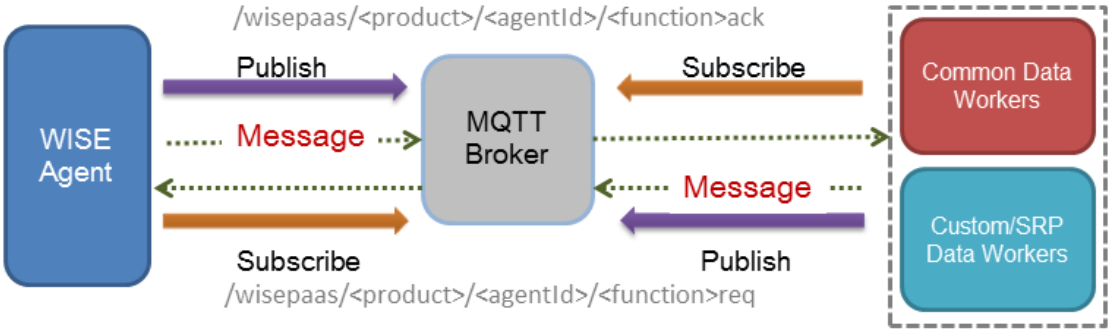


- Supports secured connection (**TLS/SSL**) with anonymous and authenticated account

4.1 Message Topic

- MQTT Topic Define
  - ✧ From Server to Agent:
    /wisepaas/<product>/<agentId>/<function>ack

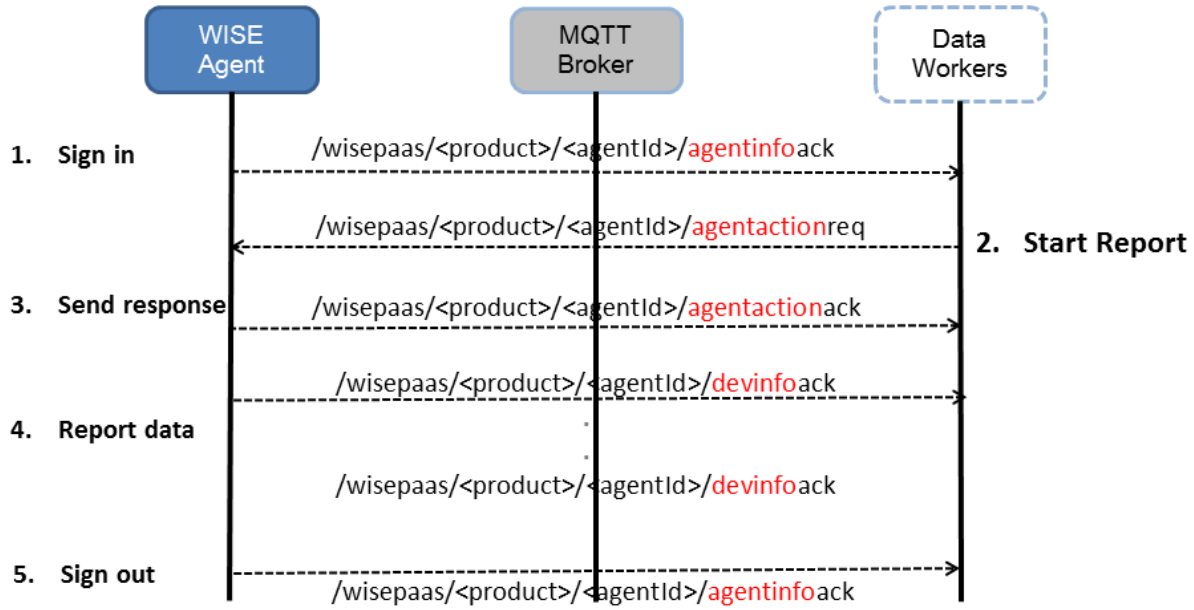
  - ✧ From Agent to Server:
    /wisepaas/<product>/<agentId>/<function>req



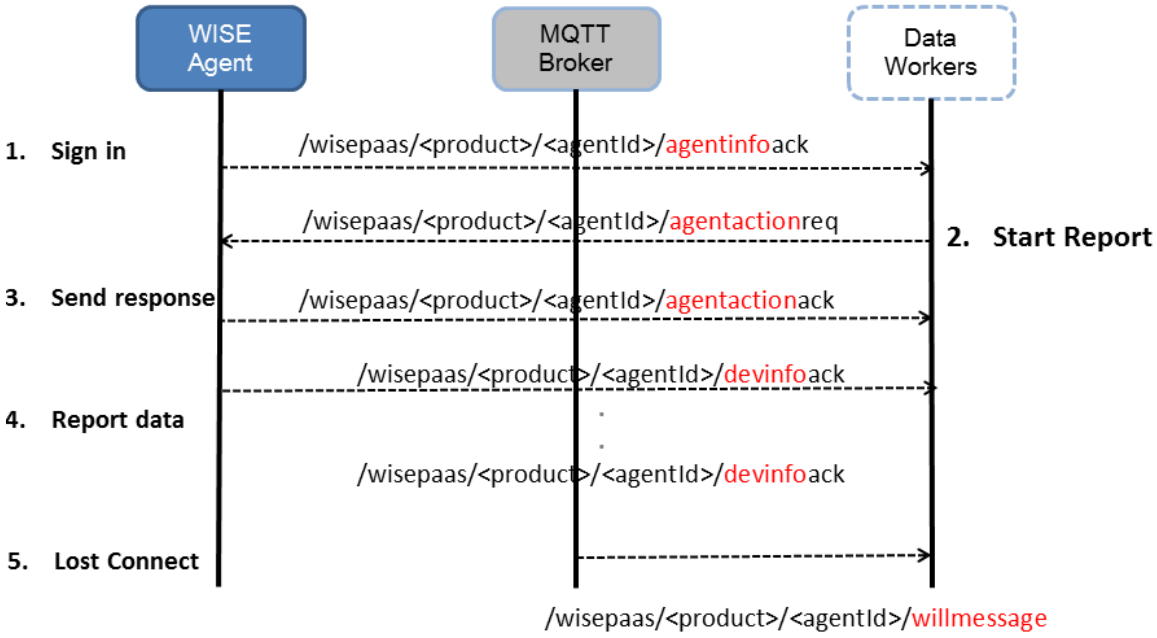
- Preserved Topic for Common use
  - ✧ Agent Registration Topic:
    /wisepaas/device/<agentId>/agentinfoack
  - ✧ Will Message Topic:
    /wisepaas/device/<agentId>/willmessage
    Or
    /wisepaas/device/<serverId>/willmessage
  - ✧ Command Topic:
    /wisepaas/device/<agentId>/agentactionreq
  - ✧ Response Topic:
    /wisepaas/device/<agentId>/agentactionack
  - ✧ Report Data Topic:
    /wisepaas/device/<agentId>/devinfoack
- Custom Topic Sample
  - ✧ Custom Topic for power ON/OFF:
    - ✧ Server send command with topic:
      /wisepaas/custom/<agentId>/powerctrlreq
    - ✧ Agent send response with topic:
      /wisepaas/custom/<agentId>/powerctrlack

4.2 Handshake Sequence

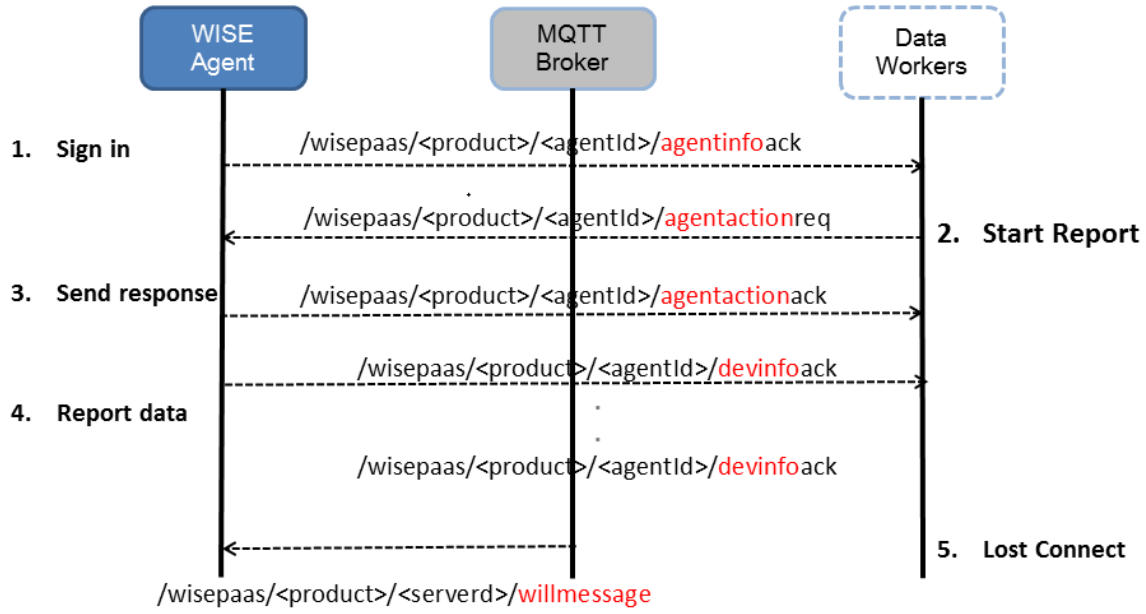
- Normal sign in and sign out sequence:



- Agent lost connection with will message sequence:



- Server lost connection with will message sequence:



4.3 Data Format

4.3.1 Basic JSON Format:

- JSON (JavaScript Object Notation) is a lightweight data-interchange format.
- JSON is built on two structures:
  - ✧ A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
  - ✧ An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.
- JSON Forms:

  - *Object:* **{** *string* **:** *value***}**
  - *Array:* **[** *value* **]**
  - *Value: string*
    number
    object
    array
    true
    false
    null

- WISE-PaaS predefined string:
  - ✧ agentID: target device ID, value type is string.
  - ✧ handlerName: target device supported function block (Plugin or Handler) name, value type is string.
  - ✧ commCmd: the sub command ID in function block (Plugin or Handler), value type is

unsigned integer.

- Device Registration Format:

```
{
   "agentID":"0123456789012",
   "handlerName":"general",        //defined for device registration
   "commCmd":1,                    //defined for device registration
   "hostname":"PC001104",          //target device name
   "sn":"14DAE996BE04",            //target device serial number
   "mac":"14DAE996BE04",           //target device mac address
   "version":"1.0.0.0",            //client application version
   "type":"IPC",                   //target device type
   "product":"",
   "manufacture":"",
   "account":"anonymous",          //bind target device to a specific account
   "password":"",                  //encrypt with DES and Base64
   "status":1                      //1: registration, 0: un-registration
}
```

- Report Data Format:

```
{
   “agentID”:”0123456789012”,
   “handlerName”:”custom”,
   “commCmd”:123,
   “custom_object”:{               //User defined report data object.
      “custom_data1”:”string_value”,
      “custom_data2”:123,
      “custom_data3”:false
   }
}
```

4.3.2 IPSO Application Framework

The IPSO Application Framework makes use of IETF standards as building blocks for a simple and efficient RESTful design model for IP smart objects. The framework may be used over either HTTP or CoAP web transfer protocols. HTTP, REST, XML, JSON, COAP and other key components of web technology are powerful mechanisms in an Internet of Things application.

Note: The objective of the Alliance is not to define technologies, but to document the use of IP-based technologies defined at the standard organizations such as IETF with focus on support by the Alliance of various use cases.

Function Sets: The framework is organized into groups of resource types called Function Sets. A Function Set has a recommended root path, under which its sub-resources are organized. Each Function Set is assigned a Resource Type parameter, therefore making it possible to discover it.

Function Set	Root Path	Resource Type
General Purpose IO	/gpio	gpio
Power	/pwr	pwr
Load Control	/load	load
Sensors	/sen	sen
Light Control	/lt	lt
Message	/msg	msg
Location	/loc	loc
Configuration	/cfg	cfg

- IPSO Smart Object

IPSO Smart Object [2] Guidelines provide a common design pattern, an object model, which can effectively use the IETF CoAP protocol to provide high level interoperability between Smart Object devices and connected software applications on other devices and services

The common object model is based on the Lightweight M2M (LWM2M 1.0) specification from the Open Mobile Alliance. OMA LWM2M is a device management and service architecture specification based on IETF CoAP, and provides a simple and flexible object template (object model) for constrained device management.

The object model from OMA LWM2M is reused to define application level IPSO Smart Objects. This enables the OMA Name Authority (OMNA) to be used to register new objects, and enables existing LWM2M compliant device libraries and server software to be used as an infrastructure for IPSO Smart Objects

- Media Types for Sensor Markup Language (SENML)
  - ✧ Semantics

SenML	JSON	Type	Description
Base Name	bn	String	This is a string that is prepended to the names found in the entries
Base Time	bt	Integer	A base time that is added to the time found in an entry
Base Units	bu	String	A base unit that is assumed for all entries, unless otherwise indicated
Version	ver	Number	Version number of media type format

Measurement or Parameters	e	Array	Array of values for sensor measurements or other generic parameters
Name	n	String	Name of the sensor or parameter
Units	u	String	Units for a measurement value
Value	v	Float	Value of the entry
String Value	sv	String	
Boolean Value	bv	Boolean	
Value Sum	s	Float	Integrated sum of the values over time
Time	t	Integer	Time when value was recorded
Update Time	ut	Integer	Update time. A time in seconds that represents the maximum time before this sensor will provide an updated reading for a measurement.

✧ The Data Type of Sensor Data Type

Data Type (type)	SenML Field
b (boolean)	bv
s (string)	s
e (enum)	e
i (integer)	v
d (decimal)	v
h(hexadecimal)	s
o(octet-stream)	s

✧ Advantech Sensor Semantics

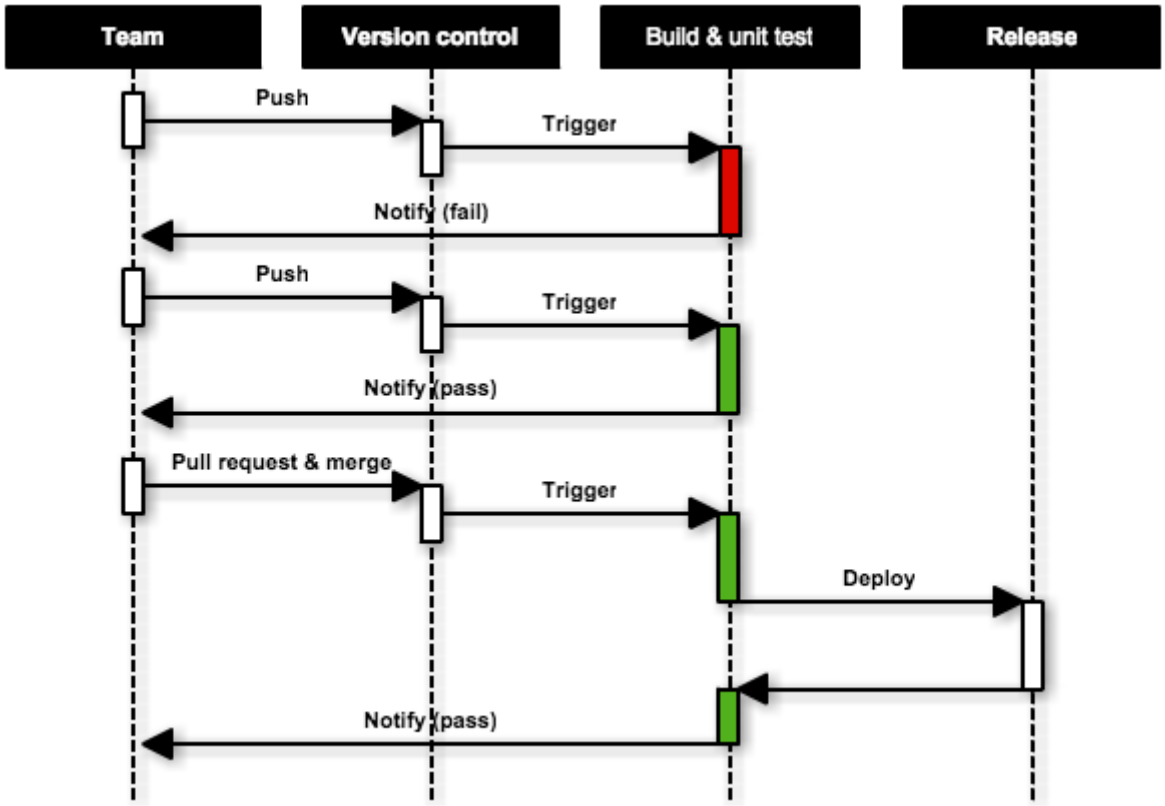
SenML	JSON	Type	Description
Min Range Value	min	Float	The minimum value that can be measured by the sensor
Max Range Value	max	Float	The maximum value that can be measured by the sensor
Access Mode	asm	String	The access mode of the resource. Ex: read (r), write (w), read/write (rw)
Standard Format	st	String	The sensor format is which standard format
Health Status	Health	Integer	The health status of network or device. Range: -1 ~ 100 Good: > 80, Average: 60 ~ 80, Below Average: 40~60, Bad:0~40, -1: Off line or Fault

- IPSO Report Data Format:

```
{
   “agentID”:“0123456789012”,
   “handlerName”:“custom”,
   “commCmd”:123,
   “custom_object”:{                    //User defined report data object in IPSO format.
      “bn”:“custom_object”,
      “e”:[
         {“n”:“ custom_data1”, “sv” ”:”string_value”},
         {“n”:“ custom_data2”, “v” ”:123},
         {“n”:“ custom_data3”, “bv” ”:false}
      ]
   }
}
```

5. DevOps Management

5.1 Continuous Integration & Continuous Delivery



5.2 Version Control

GitHub offers all of the distributed version control and source code management (SCM) functionality of Git. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

GitHub Flow is a lightweight, branch-based workflow that supports teams and projects where deployments are made regularly.

5.3 Build and Test

Add Jenkins script to check out and compile the source code. Then execute on a Docker container to verify the application. The Jenkins will send the report mail to notify pass or not.

Jenkins helps to automate the non-human part of the whole software development process, with now common things like continuous integration, but by further empowering teams to implement the technical part of a Continuous Delivery.

Docker is an open-source project that automates the deployment of applications inside software containers.

5.4 Release & Deploy

Add Jenkins script to release and deploy to target VM or upload the install packet to release folder.

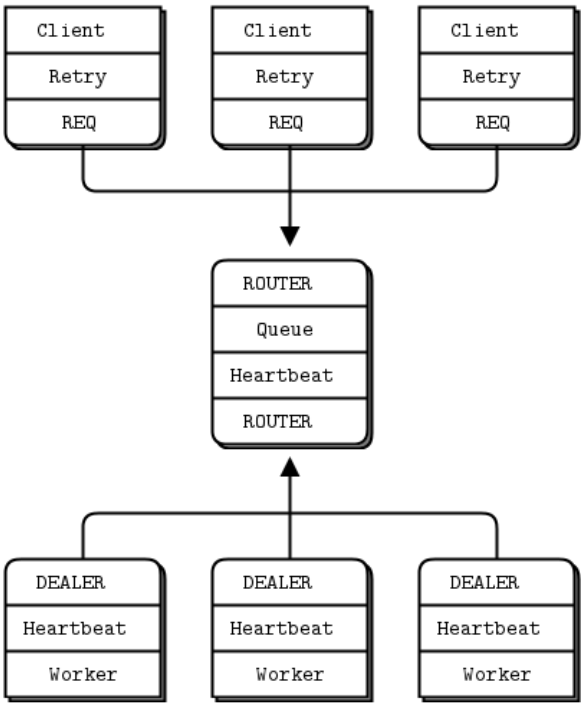
6. Custom Service Integration

6.1 Access Methods

In the common services, there are 3 mechanisms for access these information, ZMQ is the most convenient to communication information between different services. Second, the Java library have a lot's of interface to access common resources, also you would via Restful APIs to integrate.

6.2 ZMQ

In the ZMQ mechanism, we adopt the most robust with load balancing model which called "Paranoid Pirate Pattern". Based on the ZMQ to handshake message, provide high performance, stability, and cross-languages (Java/C#/C++/C/GO/Python/PHP/…etc.)



The message is adopt JSON format and similar to Restful API, reuse the translator. Format define guide as below.

Example: Get Device Information

- Request

```
{
    "request":{
```

```
        "ProductId":"<xxxx-xxx-yyyy-yyy>",
        "resource":"/wise-paas/common/device/<id>",
        "method":"GET"
    }
}
```

- Reply

```
{
    "reply":{
        "did":1,
        "name":"ARK-1155",
        "agentId":"0000BAB374520",
        "isConnected":true,
        "coordinates":"25.069439,121.582485",
        "loginTS":"2017-1-10 12:30",
        "createTS":"2017-1-5 11:00"
    }
}
```

Example: Add Account Information

- Request

```
{
    "request":{
        "ProductId":"<xxxx-xxx-yyyy-yyy>",
        "resource":"/wise-paas/common/account",
        "method":"POST",
        "content":{
        }
    }
}
```

- Reply

```
{
    "reply":{
        "result":true,
        "aid":2
    }
}
```

6.3 Java Library

6.4 Common Restful APIs

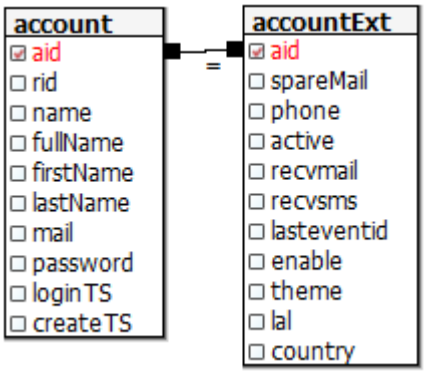
6.5 RMM/OTA/SCADA/IMM Integration

- Through ZMQ/library/Common Restful API get account/group/device information.
- Join/Merge custom DB information for advanced usage.

6.5.1 Scenario 1) RMM show more information about account

Due to common account's fields not enough for more information on RMM, the RMM would have "AccountExt" table to extend and manage.

Step 1) Get common account information via ZMQ then use primary-key to link these tables.



Step 2) Merge the result common/custom account information, then return.

6.5.2 Scenario 2) OTA deploy a package for region devices

The OTA may deploy an update package to devices locate on America, and record the upgrade status.

Step 1) Adopt common device information, use advSearch to get these devices on America.

Step 2) For these agentId/deviceId to send the request and record the status on OTA database.