Asset Manager User Guide

Kinetic - Edge & Fog Processing Module (EFM) 1.6.0

Revised: October 24, 2018

Introduction

The Asset Manager is a component of the of Cisco Kinetic EFM that can detect and manages devices throughout the EFM message system.

The discovery and onboarding are accomplished through a set of definition files that specific how the devices are discovered in the EFM node system, which fields and values will be used, as well what the resulting output will be desired. Device fields are input from the original device node or through the Asset Manager UI, enabling the user to compliment detected values with user input.

The Asset Manager is extensible and each device type has its own unique set of definitions. These definitions files can be upgraded to support new fields or renaming as requirements change.

After discovery and a device is approved, the Asset Manager maintains a list of devices in the Asset Manager graphical interface, but also creates a node structure in the EFM data path that can be used as input for other applications. Each accepted device will be found under the kinetic-efm-asset-dslink/Assets, where the name is taken from user defined Label. Labels need not be unique.

Asset Manager Components and functions

The Asset manager encompasses the following components:

- Application Server Backend, an Asset DSLink
- Asset Manager UI Web component
- A set of (example) device onboarding definition files and device simulation DSLink.

The Application server backend is responsible for discovering the devices to be onboarded using the DQL[1] based search pattern from the onboarding definition files. It also persists the device onboarding decision of the end-user. This package also contains the kinetic-efm-dslink-bridge module which acts as the connection point between the application server and the DSA network.

The Asset DSLink (kinetic-efm-asset-link) is implemented in Java and its purpose is to transform Low Level Device Driver DSLink data into normalized device/device structure. It is also exposing the describing dimensions that an end-user can enter inside the Asset Manager UI.

The Asset Manager UI web component is used in managing the devices (Accept, Reject, Delete and edit details of devices) for the end-user.

The device onboarding definition, defines the device to be onboarded. The asset class definition, defines the normalized device data model. Example asset definitions are part of the installation and are used in automatically discovering devices from Device simulation DSLink in Asset Manager UI web component. More details on the onboarding can be found in the Onboarding Configuration section.

The Device simulation DSLink is used to generate simulated device readings for Vibration and Temperature devices. This can be used for demonstration purposes. By default, the DSLink generates 5 Temperature Devices and 5 Vibration

[1] DQL is a distributed query language for EFM. DQL allows for queries across a single broker or optionally across a multi-broker system. For more details on using DQL and examples see https://github.com/IOT-DSA/dslink-dart-dql/blob/master/README.md .

Devices. It is an optional component can be removed by the system administrator. More details on the device-simulator can be found in the Device Simulator section.

Logging in to the Asset Manager

In order to connect to the Asset Manager, enter default URL is the following `https://[Server IP Address].` If not already logged into the EFM Manager, it is necessary the user insert their username and password and click “Log In” to continue. After, select the “Asset Manager” to enter the application.

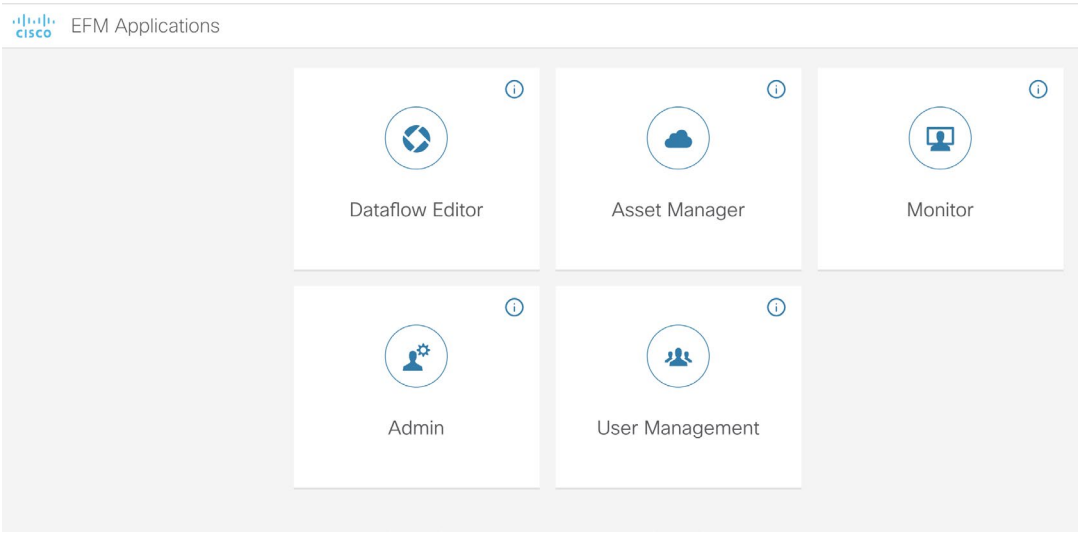


Figure 1. EFM Main applications page

User Interface Overview

The user interface of the Kinetic – EFM Asset Manager contains two main pages. In the leftmost portion of the UI, a selector provides the ability to switch between two different views. The options are:

1. Devices – All the approved devices appear in the Devices page. This is the initial page when logging in.
2. Discovery – The onboarding page shows all the discovered or rejected devices separately. The list selection is determined by the selection on the page.

Devices Page

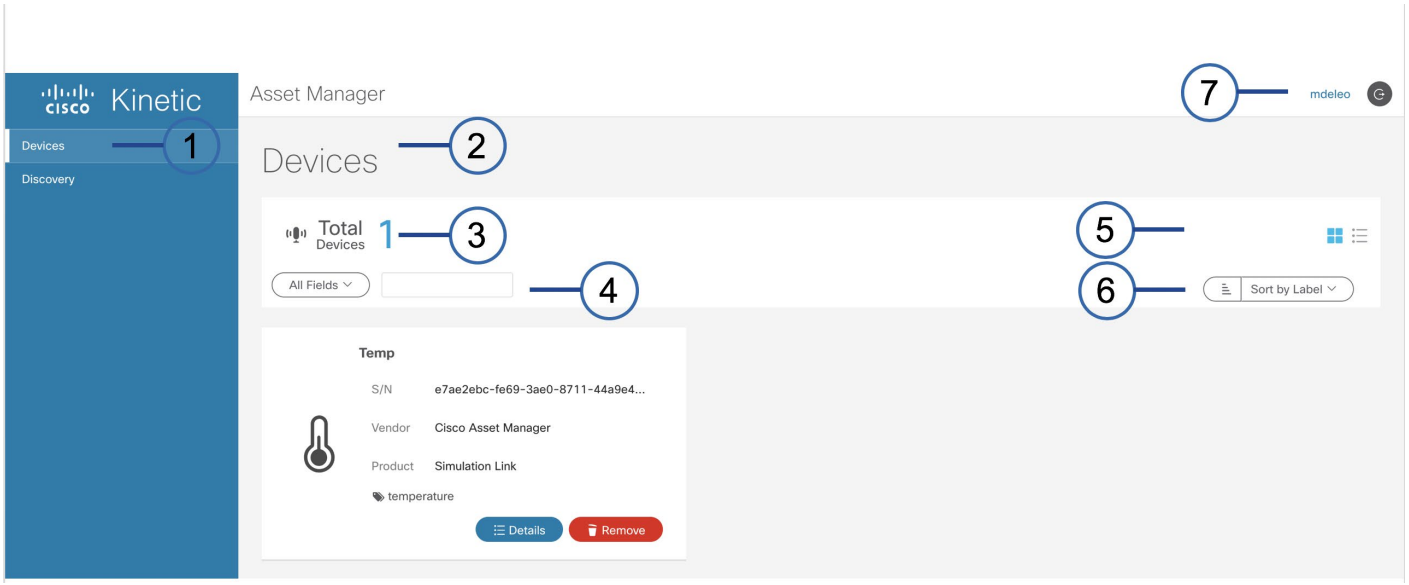


Figure 2. Asset Manager and View Selection (Block Mode)

1	View Selection pane	**2**	Devices List Title
3	Summary counts of devices	**4**	Search - The drop-down menu allows for selecting the attribute to search based upon the onboarding definition files, for example: • All Fields • Asset Type Definition Id • Discovery Definition Id • Tags • Label • Product • S/N • Vendor The text box allows for the user to input the search criteria.
5	Block (left) or detailed (right) selector for devices list	**6**	Ascending/Descending and Sort criteria: • Asset Type Definition Id • Discovery Definition Id • Label • Product • Vendor
7	Username (left) and logoff (right)	**8**	Devices List

Selecting one of the options in the View Selection pane causes the graphical view to change.

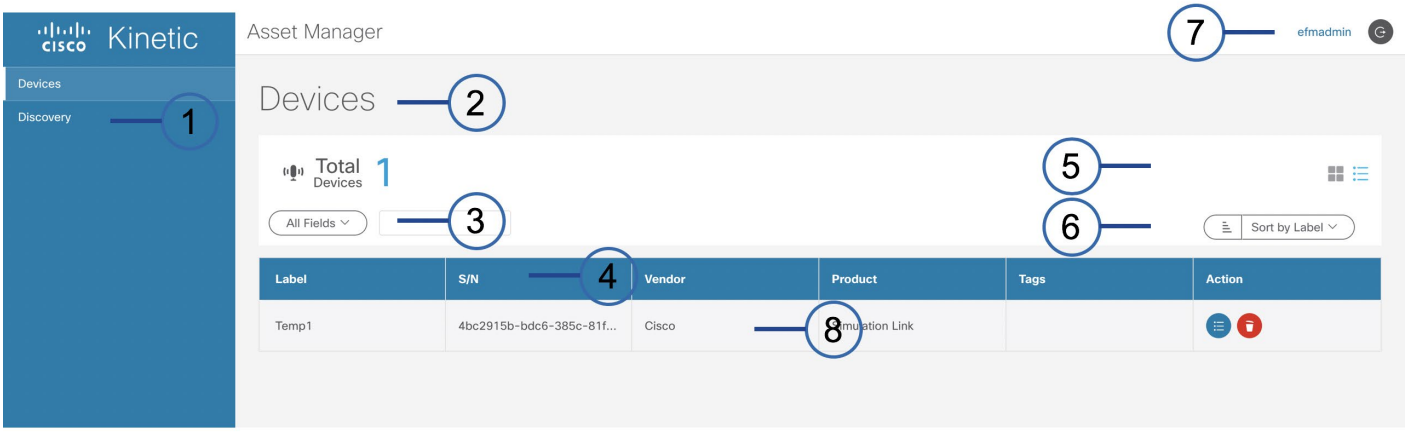


Figure 3. Asset Manager and View Selection (List Mode) - see description table above

Viewing and updating a Device

Select the “Details” or Details symbol for the specific device in the list to see the details. The page will be updated with the details defined in the onboarding definition files. For the simulated temperature device, it is as follows.

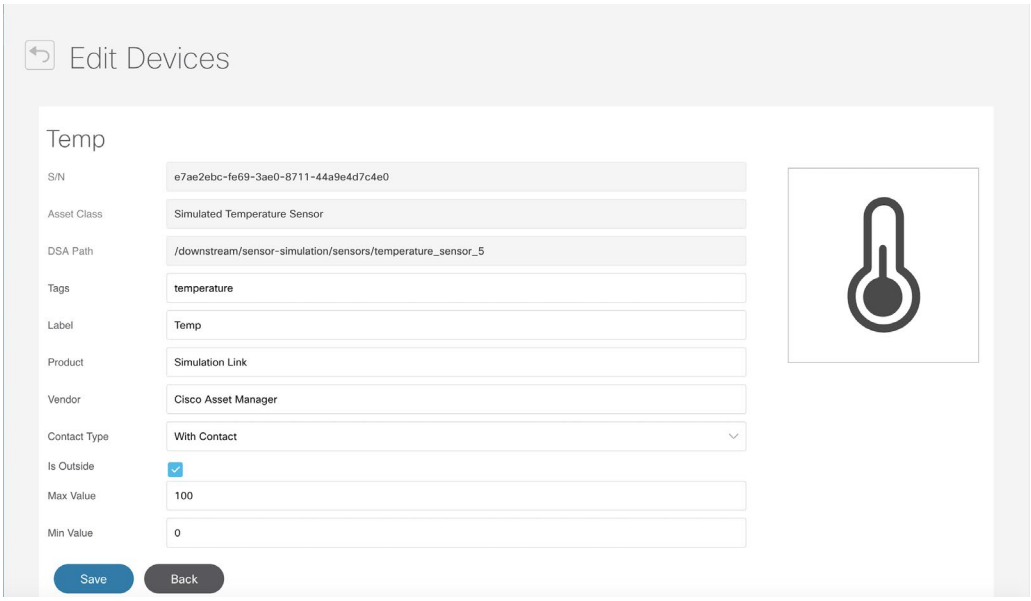


Figure 4. Device Detail

The device detail allows for viewing and editing the device attributes that permit. Select Save to store the updated information.

Removing a Device

Select the “Remove” or trash can symbol for the specific device in the list to remove. It will be removed from the Devices list.

If a device is removed by any other method than the Asset Manager UI, such as the use of dataflow editor, the device will be recreated.

Discovery Page and Device onboarding

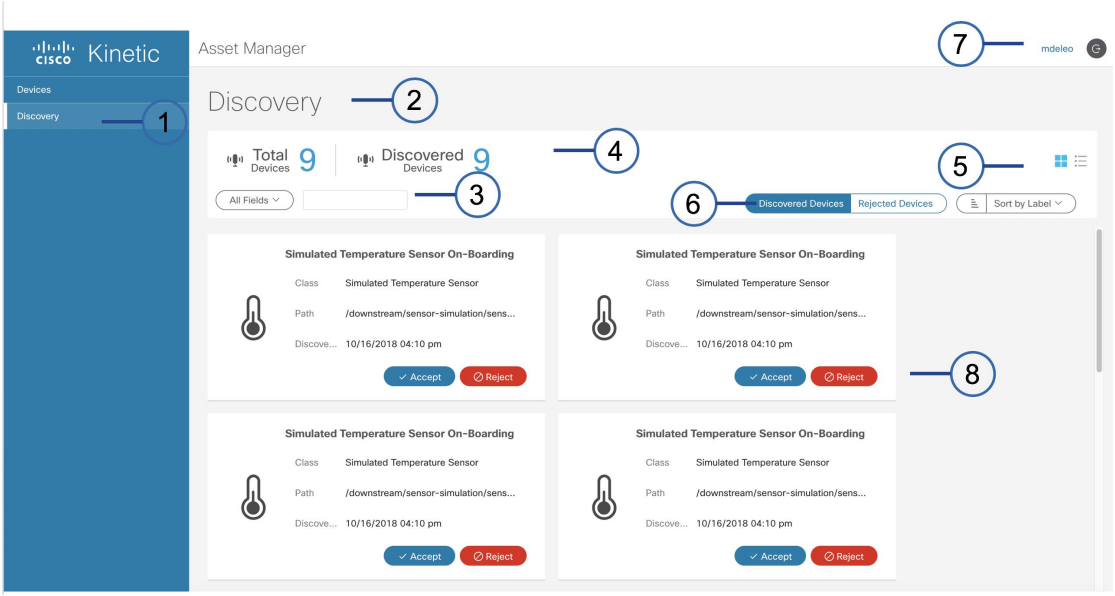


Figure 4. Onboarding Devices - Discovered Devices page

1	View Selection pane	**2**	Devices List Title
3	Summary counts of devices	**4**	Search - The drop-down menu allows for selecting the attribute to search based upon the onboarding definition files, for example: • All Fields • Label • Class • S/N The text box allows for the user to input the search criteria.
5	Block or detailed List display for devices (right)	**6**	Discovered Devices or Rejected Devices and Sort criteria: • Label • Class
7	Username (left) and logoff (right)	**8**	Discovered Devices List

The Asset Manager discovery is a continuous process. The Asset Manager queries the EFM message system for devices, devices or devices that have been defined using the Cisco EFM Device Object Model Standard structure. When

a new device is found and it is not already in the “Approved Device” list nor the Rejected Devices, it is placed in the Discovered list in the “Onboarding” section.

Devices that have been deleted from the “Rejected Devices” list can be re-discovered again but may take up to 10 minutes to be processed.

Accepting a Device

When accepting a device, the following attribute fields are shown and any extra fields defined in onboarding definition files:

- S/N (Serial Number) - (Device Provided)
- Asset Class - (Device Provided)
- DSA Path - (Device Provided)
- Tags
- Label
- Product
- Vendor

The Serial Number, Asset Class, DSA Path are read-only fields. The fields, Tag, Label, Product and Vendor is shown by default for all devices. Tags is a custom text-based value useful for grouping and searching the devices, for example “[Vibration]” can be used for vibration devices. The user must define a “Label” string that will identify the device by name. The Label string does not have to be unique. The Label Name will be the same node name that is created under the Assets path for the kinetic-efm-asset-dslink. The Device's Product and Vendor details are also mandatory fields which needs to be filled.

The other extra attributes are read and shown from the device definition if they exist and the input is based on how the field is defined in the onboarding definition files.

Once the device is saved, it is placed in the Devices list and can be viewed by selecting the “Devices” tab on the left pane.

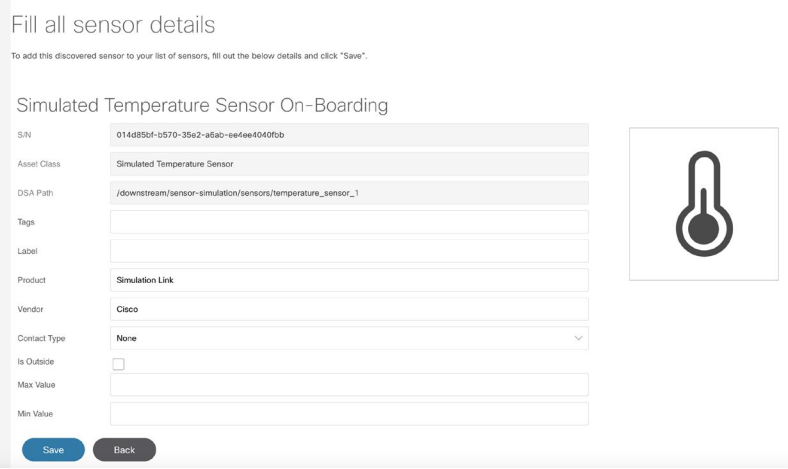


Figure 5. Onboarding a Device – Accepting a Device

Rejecting a Device in the Discovered Device list

If a device is rejected, it will be put into the Rejected Devices list in the Asset Manager Onboarding tab. Once a device is rejected it can be viewed by selecting the “Reject Devices” option on the screen. A list of devices will appear below. Even after rejecting a device, the user can decide to accept a device or delete. When accepting the device, it will be the same process as mentioned in Accepting an Asset above.

If a device is rejected, the following message will appear to confirm:

Confirm Rejection

Do you really want to reject this discovery entry? NOTE: This action will not delete the entry, but only mark it as rejected. It can be recovered later on.

Once the Reject button is selected, the device will move to the Rejected Devices List.

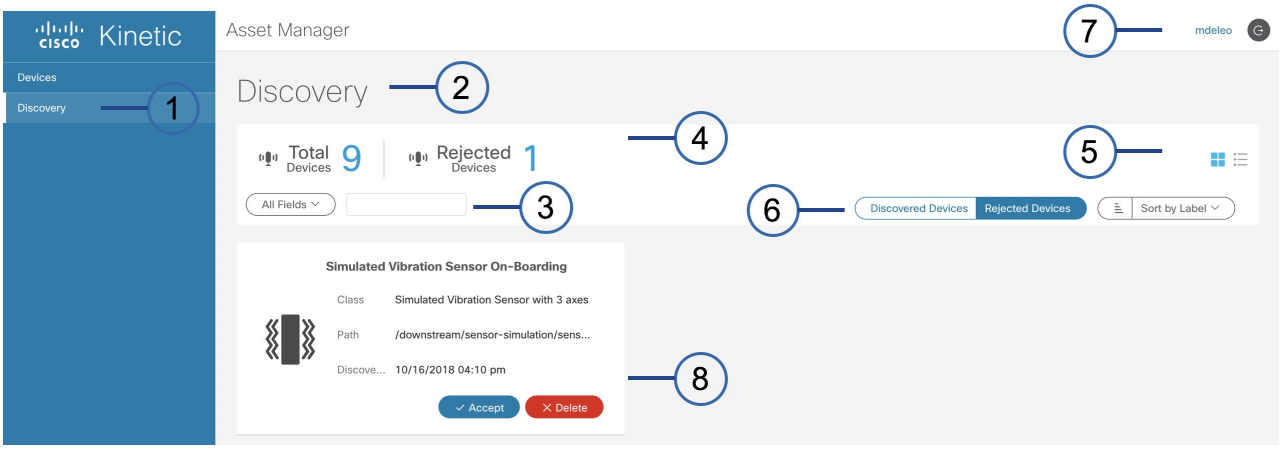


Figure 6. Onboarding DEVICES – Rejected Devices page

1	View Selection pane	**2**	Devices List Title
3	Summary counts of devices	**4**	Search (when search is toggled ON). The arrow allows for selecting the attribute to search based upon the onboarding definition files, for example: • All Fields • Label • Class • S/N
5	Block or detailed List display for devices (right)	**6**	Discovered devices or Rejected devices and Sort criteria: • Label • S/N • Vendor • Product
7	Username (left) and logoff (right)	**8**	Devices List

A device can be accepted in the Rejected Devices list or deleted from the Asset Manager. If it is accepted, then the same page as when Onboarding appears to fill in all the device details. The fields S/N, Asset Class and DSA Path are modifiable by the user.

If a device is deleted, the following message appears to confirm:

Confirm Delete

Do you really want to delete this discovery entry? NOTE: This action will delete the entry permanently. It cannot be recovered.

By selecting Delete the action is taken. Only if the device is discovered again will it be placed into the “Discovered Devices” list in the Asset Manager Discovery tab. This process may take up to 10 minutes from the deletion.

Deleting a Device in the Rejected Device list

When deleting the device, the Asset Manager loses all information.

Only if the device is discovered again will be placed into the “Discovered Devices” list in the EFM Manager Onboarding tab. This process may take up to 10 minutes from the deletion. The discovery period is configurable in the application-conf.json file (restartIntervalMinutes).

Normalized Asset Synchronization

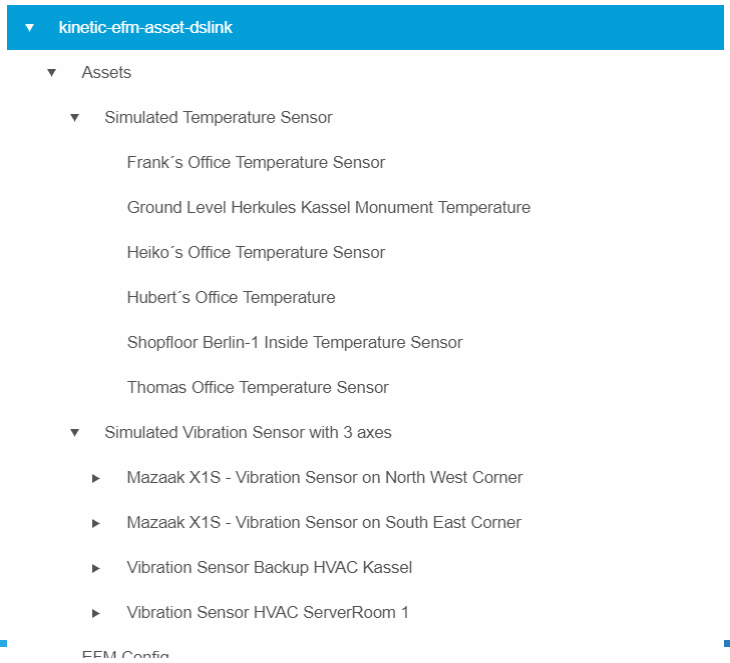
The Asset Manager maintains synchronization between the Asset Registry and the node structure under the Asset DSLink (kinetic-efm-asset-dslink/Assets). The synchronization process provides the following:

- On removing device from Asset manager UI, the entry is removed from Asset DSLink.
- On removing a device from Asset DSLink, the entry gets recreated.
- The Changes done on the Asset Manager UI are immediately reflected in Asset DSLink.

Normalized Asset Grouping

The kinetic-efm-asset-dslink 1.6 link supports device grouping under the node structure downstream/kinetic-efm-asset-dslink/Assets. The grouping is is performed per the label indicated in the configuration schema file(graphql file).

Asset grouping example



Backup and restore of EFM Asset Manager

The vital information from EFM Asset Manager can be backed up and restored.

Implications:

- The Asset Manager efm-manager needs to be stopped for backup and restore
- Files are copied into the same directories they have been backed up from
- During restore Ignite data is deleted

For Backup:

1. Stop the Asset Manager efm-manager:
   `service efm-manager stop`
2. Execute
   `java -cp lib/efm-servicelayer*-fat.jar com.cisco.efm.launcher.ApplicationLauncher backup --outputDir=/path/to/backup`
3. Default parameters such as `--configDir`, `--workingDir` and `--conf` are supported
   a. `configDir` - the directory of the application where configuration is stored
   b. `workingDir` - the working directory of the application, used for storing ignite data and DSLink data
   c. `conf` - default config parameter defining json configuration file
4. A zip files will be created if backup is successful, during the backup a directory named `_build` will be created in the `outputDir`
5. An Example is given below.
   This command can be run from `/opt/cisco/kinetic/efm_manager/lib`
   ```
   java -cp efm-servicelayer* com.cisco.efm.launcher.ApplicationLauncher backup --outputDir=/home/efm/ --configDir=/opt/cisco/kinetic/efm_manager/config/ --workingDir=/opt/cisco/kinetic/efm_manager/app-data/ --conf=/opt/cisco/kinetic/efm_manager/config/application-conf.json
   ```
   This command can be run from `/opt/cisco/kinetic/efm_manager/`
   ```
   java -cp lib/efm-servicelayer* com.cisco.efm.launcher.ApplicationLauncher backup --outputDir=/home/efm/
   ```
6. On executing the above command, the logs of execution are written in to log/server.log relative to the path from where the script is executed.

For Restore:

1. A successful installation of the efm-manager is required before restoring
2. Stop efm-manager
   `service efm-manager stop`
3. The folder `config` and `app-data` in `/opt/cisco/kinetic/efm_manager/` will be overwritten. It is necessary to delete the folder before restore.
4. Start restore
   `java -cp efm-servicelayer*-fat.jar com.cisco.efm.launcher.ApplicationLauncher restore --backupPath=/path/to/backup.zip`
5. No other parameters are required to restore.
6. An example is given below:
   This command can be run from `/opt/cisco/kinetic/efm_manager/lib`
   `java -cp efm-servicelayer*-fat.jar com.cisco.efm.launcher.ApplicationLauncher restore --backupPath=/home/efm/backup-1540303333.zip`

   This command can be run from `/opt/cisco/kinetic/efm_manager/`
   `java -cp lib/efm-servicelayer*-fat.jar com.cisco.efm.launcher.ApplicationLauncher restore --backupPath=/home/efm/backup-1540303333.zip`
7. On executing the above command, logging would be shown. Look for message that says " Restored all service data" .
8. Check whether the `config` folder and the `app-data` folder are restored and run the following command in case `efm` is not the owner of the folder.
   `chown -R efm:efm config/`
   `chown -R efm:efm app-data/`
9. Start efm-manager
   `service efm-manager start`
10. Restart `efm-asset-dslink` from EFM System Administrator or dataflow editor

Configuring the EFM Asset Manager via the application-conf.json file

The EFM Asset Manager is configured in the `/opt/cisco/kinetic/efm_manager/config/application-conf.json` file.

Configuration Property Name / Path	Description	Mandatory	Default Value
asset-registry/normalization/assetSynchronizationInitialDelaySeconds	Initial delay for asset DSLink synchronization in seconds. When the value is negative or zero initial synchronization is disabled.	N	30 s
asset-registry/normalization/assetSynchronizationTimeoutMinutes	Total timeout minutes for asset DSLink synchronization.	N	5 min
asset-registry/normalization/assetSynchronizationListIdleTimeoutSeconds	Idle timeout seconds for asset DSLink synchronization list request. After the given amount of seconds without message from the broker the list request is closed.	N	15 s
asset-registry/normalization/actionTimeoutSeconds	Timeout seconds for asset DSLink action invocations	N	30 s
asset-registry/normalization/enabled	Boolean that indicates if the asset DSLink integration is enabled.	N	true
asset-registry/normalization/assetSynchronizationIntervalMinutes	Interval minutes for asset DSLink synchronization. Default value is 10. When the value is negative or zero synchronization is disabled.	N	10 min
asset-registry/discovery/enabled	Boolean that indicates if the device discovery is enabled.	N	true
asset-registry/discovery/fetchDataTimeoutSeconds	Timeout seconds for fetching a single dimension value from a source DSLink node.	N	60 s
asset-registry/discovery/restartIntervalMinutes	Interval minutes between scheduled restarts of the device discovery workflow.	N	10 min
asset-registry/discovery/startDelaySeconds	Delay before initial first start of device discovery.	N	10 s
asset-registry/discovery/queryIdleTimeoutSeconds	Idle DQL / List timeout seconds.	N	15 s
asset-registry/listenerInvocationTimeoutSeconds	Amount of minutes a before asset registry change listener has to respond until a timeout occurs.	N	2 min
asset-registry/listenerMaxRetryWaitSeconds	Max. number of seconds between two retry attempts.	N	30 s
asset-registry/listenerInvocationRetryCount	Max. number of retry attempts	N	5
web-app/jwt/usernameClaimKey	JWT token claim name for user name	N	username
web-app/jwt/permissionsClaimKey	JWT token claim name for user permissions array	N	permissions
web-app/jwt/pubSecKeys	Array of public key information for JWT authentication. The array has to contain objects with keys publicKey and algorithm.	Y	
web-app/noAuthX	Disables authentication requirement	N	false

dslink-bridge/broker	Broker connection url. This has to be the url for the root broker.	Y	
dslink-bridge/token	Optional credential token	N	
dslink-bridge/listTimerIntervalMillis	Interval milliseconds in between checks if a list request is considered to be finished	N	500 ms
dslink-bridge/listIdleTimeoutSeconds	Timeout seconds for a list request. If no response is being received for the specified amount of time the list request is considered to be finished.	N	5
dslink-bridge/requestTimeoutSeconds	Total timeout seconds for a DSA request.	N	30 s
dslink-bridge/customDQLQuery	Custom DQL query for resolving DSLinks by sysId	N	option traverseBrokers =true | list * | filter $sysId=" %s" | subscribe :name $base $sysType $sysVersion
dslink-bridge/dqlPath	Optional DQL path used for global queries	N	
config-installer/discoveryConfigFolder	Folder from where asset class definitions and discovery definitions are loaded and installed. The files can be placed in sub-folders. If an absolute path is configured the absolute path is used. Otherwise the path is interpreted relative to the configuration folder.	N	discovery
runtime/systemdNotifierEnabled	Flag if the systemd notifier workflow should be enabled	N	true

Onboarding Configuration

Devices Onboarding Definition

For DSA devices to be added to the EFM Device Manager, there needs to be a definition on how the node structure should be transformed to a normalized EFM Asset Manager device. Device onboarding definitions enable exactly this. There can be multiple onboarding definitions for the same target device schema definition.

The EFM Asset Manager is an extensible component, support many types of device definitions. It ships with the simulator link and the corresponding device onboarding configuration file definitions for vibration and temperature sensors. In addition, there is a set of template files to make the process of onboarding definition easier for the user. These will be explained in the next section.

All EFM Asset Manager configuration files are placed in the `$EFM_ROOT/efm_manager/config/discovery` directory. They can be grouped under folder if desired, such as Panduit-link, simulation-link, template, etc. Each device requires two file definitions:

- Asset Onboarding Definition for the device (source DSLink mapping). This is a json formatted file. This file defines the inputs for the device.
- Asset Class that corresponds to the specific device Definition (describes the normalized device). This is a GraphQL formatted file. This file defines what the outputs will be for the device.

Creating user Onboarding Definition files using the Onboarding Definition Templates

The Asset Manager installation includes a generic set of onboarding definition files that we recommend using as templates. These can be found under the efm_manager/config/discovery/template directory. These files contain several sections, but we describe what fields are user configurable as well as the mandatory input fields.

Note that the files suffix terminate with .template. This termination is not a valid .json or .graphqls suffix and is ignored by the Asset Manager at startup or restart. The user must copy and rename the template files for detection by the EFM Asset Manager.

The EFM Asset Manager template configuration files are placed in the `$EFM_ROOT/efm_manager/config/discovery/template` directory. Copy the files to a new directory in the `$EFM_ROOT/efm_manager/config/discovery` path. For example, power-link.

Each device requires two file definitions:

- Asset Onboarding Definition for the device (source DSLink mapping). This is a json formatted file. This file defines the inputs for the device. The template file is device_discovery_definitions_((assetType))_v((assetTypeNumber)).json.template. It is suggested that the user replace ((assetType)) with the assetType name and v((assetTypeNumber)) with the version number.[2]
- Asset Class that corresponds to the specific device definition (describe the normalized device). This is a GraphQL formatted file. This file defines what the outputs will be for the device. The template file is

[2] The JSON file format is a standard JSON structure. All comments must be removed from the template for proper parsing to occur.

device_class_definitions_((assetType))_v((assetTypeNumber)).graphqls.template. It is suggested that the user replace ((assetType)) with the assetType name and v((assetTypeNumber)) with the version number.

For example, we will rename the json file as `asset_discovery_definitions_temperatureSensor_v2.json` and graphqls file as `asset_class_temperatureSensor_v2.graphqls`.

In the following sections, we will describe the template files and the sections that can be modified. After these descriptions, example to discover message brokers and onboard them as devices are explained..

Device Discovery Definition JSON Template explained

This file is a JSON formatted file that defines how a device class is discovered and where it is placed into the broker data path. The template file name is "`device_discovery_definitions_((assetType))_v((assetTypeNumber)).json.template`". An example file name is `device_discovery_definitions_BrokerDevice_v1.json`.

The values surrounded by <> require valid values to be configured. Remove all lines that start with ##, since commenting is not supported in the json file definition.

Note that if a device has been discovered with an existing device onboarding definition, modification of the device definition file is not recommended. Rather it is recommended a new instance be created increasing the version number inside the file, for example from version 1 to 2, 2 to 3, and so on.

This file has the following functional sections:

- General definitions (label, discoveryClassId, assetType, assetTypeVersion and productImageRef)
- Dimension Mapping, an array that defines a **static** field to be read from the discovered node, which will contain the serial field mapping and any other static fields
- Discoveries, the method of finding devices is an array of Discovery Configs or queries
- Dataflow, an optional array that defines a **streaming** field to be read from the discovered node. A Dataflow field can only be output as streaming value in the node path under the "label", the Asset Manager UI does not show this field.

```
[
  {
    "label": "<Label for Discovery>",
    "discoveryClassId": "<Discovery Class ID>",
    "assetType": "<Name for Asset Type>",
    "assetTypeVersion": <Version Number for Asset Type>,
    "productImageRef": "<static/images/cisco.svg>",
```

Section 1. - General definitions:

Label - Discovery name, for example "`Simulated Temperature Sensor On-Boarding''`

discoveryClassId - unique name definition, for example "`SimulationDevice-TemperatureSensor-1.0`"

assetType - unique asset type, for example "`SimulatedTemperatureSensor`"

assetTypeVersion - asset type configuration version

	productImageRef - icon image that appears in the Asset Manager UI. Path relative to $EFM_ROOT/efm_manager /web/efm_manager/static/images/cisco.svg

"dimensionMapping": [
{
"type": "DSA",
"targetPath": "<Source DSA Path to read dimension>",
"fieldName": "<uniqueFieldName>"
}
	Section 2. - Dimension Mapping
	
	Type - always DSA
	
	tartgetPath - Source DSA Path to read dimension
	
	fieldname - unique field name for the field read dimension
	
	If more than one dimension needs to be read, repeat the red highlighted lines for those many times and separate them by a comma

"discoveries": [
{
"type": "DQL",
"query": "<DQL Query which fetches the device lists>"
}
	Section 3. - Discoveries
	
	Type - DQL
	query - DQL query statement
	
	If more than one DQL needs to be run, repeat the above red highlighted lines for those many times and separate each of them by a comma

"dataflow": {
"<uniqueFieldName>": {
"type": "write-through-dataflow",
"config": {
"path": "<Source DSA Path to read metric value>"
}
}
	Section 4. Dataflow
	
	uniqueFieldName - unique field name for the field read metric
	type - must be “write-through-dataflow”
	path - source of DSA path to read metric value (Relative path from broker where efm manager is connected to)
	
	If more than one metric needs to be read - repeat the above red highlighted lines for those many times and separate each of them by a comma

Device Discovery Definition JSON Template reference

This section details the device_discovery_definitions_((assetType))_v((assetTypeNumber)).json.template reference file. This first section is the template file followed by a detailed description of the sections of the file.

```
## <> Rule - Values surrounded by <> are the ones that are expected to be changed to valid values.
Consider | rule explained above as well. Remove characters < and > as well ##
## Remove lines that start with ## characters as well ##
[
  {
    "label": "<Label for Discovery>",
    "discoveryClassId": "<Discovery Class ID>",
    "assetType": "<Name for Asset Type>",
    "assetTypeVersion": <Version Number for Asset Type>,
    "productImageRef": "<static/images/cisco.svg>",
    "dimensionMapping": [
```

```
      {
        "type": "DSA",
        "targetPath": "<Source DSA Path to read dimension>",
        "fieldName": "<uniqueFieldName>"
      }
      ## If more than one dimension needs to be read - repeat the above 5 lines for those many times and
separate them by comma##
    ],
    "defaultValues": {
      "<uniqueFieldName>": "<Default Value to be entered into this dimension field>"
      ## If more than one dimension needs to be defaulted - repeat the above 1 line for those many times
and separate them by comma##
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "<DQL Query which fetches the device lists>"
      }
      ## If more than one DQL needs to be run - repeat the above 4 lines for those many times and
separate each of them by comma##
    ],
    "dataflow": {
      "<uniqueFieldName>": {
        "type": "write-through-dataflow",
        "config": {
          "path": "<Source DSA Path to read metric value>"
        }
      }
      ## If more than one metric needs to be read - repeat the above 6 lines for those many times and
separate each of them by comma##
    }
  }
]
```

Device Class Definition graphqls Template explained

Normalized devices are described by a schema. The schema defines the *dimension* fields (that appear in the UI and also under the asset manager data node structure) and *metric* (that are streamed to the asset manager data node structure with the label header) fields and their value scopes. There are some fields such as **`id`** or **`label`**, that all devices share, can be generalized to an abstract device base type. The device schema definition needs a serialized format because adding of new schemas are expected to work at start or restart.

Note that if a device has been discovered with an existing device onboarding definition, modification of the device class definition file is not recommended. Rather it is recommended a new instance be created increasing the version number inside the file, for example from version 1 to 2, 2 to 3, and so on.

This file has the following functional sections:

- General definitions
- Defining Dimensions, apart from ones defined in Asset or Device class definitions

- Defining Metrics
- Renaming dimensions
- Renaming metrics
- Defining Default Value, needs to be set when changing a nullable to non-nullable dimension
- Defining enum dimensions (a list dimension) required in the asset. The next section defines the enum items that will use a UI drop down list. They must be unique definitions and used once.
- Defining a mandatory set of different enum dimensions required in the asset. The next section defines the enum items which will appear in the UI drop down list. They must correspond to the enum definition in the previous section.

This is the example file "`device_class_definitions_((assetType))_v((assetTypeNumber)).graphqls.template`".

The definition file has rules that need to be followed:

- Comments are allowed and will be ignored. It is recommended the use of comments for documentation purposes.
- | Rule - Values which have a | in the middle indicate there are choices that user has, by which only one choice can be chosen. Remove all other choices including the | symbol(s)
- <> Rule - Values surrounded by <> are the ones that are expected to be changed to valid values. Consider | rule explained above as well. Remove characters < and > as well
- [] Rule - Values surrounded by [] are optional definitions which either needs to be completely removed or to be used exactly as defined. Consider <> rule explained above as well. Remove characters [and] as well
- The optional directive @dsaMapping will be defaulted to $<uniqueFieldName> if the whole @dsaMapping directive is omitted
- The optional directive @rename must be used when intending to rename a field name
- The optional directive @defaultValue for dimension must be used when making a nullable field as a non nullable field
- The optional argument *unique* will be defaulted to false if whole argument is omitted
- The optional argument *searchable* will be defaulted to false if whole argument is omitted
- The optional argument *access* will be defaulted to RW if whole argument is omitted
- The optional argument *unitName* will be defaulted to " " , if whole argument is omitted
- The optional argument *unitSymbol* will be defaulted to " " , if whole argument is omitted
- The type name defined in enum <assetType>_<uniqueEnumFieldDefinition> must be unique across all available asset definitions. Hence <assetType> is also added in the type name
- The type name must match between the field definition and the enum definition, for the enum values to be used

[#<Comments for type of asset>] type <AssetType> implements Asset & Device @asset(version: <AssetTypeVersion> label: "<Label for Asset Class>") {	**Section 1. - General definitions**: assetType - unique asset type, for example "SimulatedTemperatureSensor" assetTypeVersion - asset type configuration version(numeric value) Label -Asset Class name, for example "Simulated Temperature Sensor'' This section is define once.
[#<Comment for Dimension Field>] <uniqueFieldName>: <String | Int | Float | Long | Boolean>[!] [@rename(oldName: " <uniqueFieldNameOld>")] [@defaultValue(value: " <String representation of default value>")] [@dsaMapping(path: "<DSA path expression>")] @dimension(label: "<Display Label of Field>" [unique: <true | false>] [searchable: <true | false>] [access: "<R | RW>"])	**Section 2. - Defining Dimensions, apart from ones defined in Asset or Device class definitions:** Dimension - This field appears in the UI at acceptance. Field will be copied once from source link and as per access type. It can be modified or this can be a completely new field which will be made available in the normalized asset DSLink uniqueFieldName - unique name. Optional use of "!" to define a non-null field. label - The Label to be shown in the UI unique - indicates if field values is unique(example a serial value is unique) searchable - To indicate whether the field is searchable in UI access - read and write capability Repeat this block for each dimension to be defined, apart from ones defined in Asset or Device class definitions. As part of installation the following dimensions are already available. They are Serial, Tag, Label, Product and Vendor. In case of renaming the field after initial version is installed, then the @rename directive must be used. In case of making a nullable field as non nullable after initial version is installed, @defaultValue directive must be used

`[#<Comment for Metric Field>]`
`<uniqueFieldName>: <String | Int | Float | Long | Boolean> [@rename(oldName: "<uniqueFieldNameOld>")] [@dsaMapping(path: "<DSA path expression>")] @metric(`
`label: "<Display Label of Field>"`
`[unitName: "<Metric unit name>"]`
`[unitSymbol: "<Metric unit symbol>"]`
`)`
	Section 3. - Defining metrics:
	
	Metric - which is updated with the latest streamed value from source node
	
	uniqueFieldName - unique name.
	
	label - The Label for the metric
	
	unitName – indicates the Unit Name for Metric
	
	unitSymbol – indicates the Unit Symbol for Metric
	
	Repeat this block for each metric to be defined
	
	In case of renaming the field after initial version is installed, then the @rename directive must be used.

`[#<Comment for Dimension Field>]`
`<uniqueFieldName>: <assetType>_<uniqueEnumFieldTypeName> [@dsaMapping(path: "<DSA path expression>")] @dimension(`
`label: "<Display Label of Field>"`
`[searchable: <true | false>]`
`[access: "<R | RW>"]`
`)`
	Section 4. - Defining a set of enum dimensions required in an asset.
	
	The Type <assetType>_<uniqueEnumFieldTypeName> must match between the enum and this field definition.(Section 5)
	
	Repeat the 6 lines for numbers of enum dimensions required in the asset.

`[enum <assetType>_<uniqueEnumFieldTypeName> {`
`<uniqueEnumFieldItemName> @enumValue(label: "<Display Value for Dropdown Item>")`
`## Repeat above 1 line for number of options which needs to be shown in the Dropdownlist ##`
`}]`
	Section 5. - Defining (mandatory set of different enum dimensions required in the asset
	
	Note: This is inserted into the global section of the template.
	
	`uniqueEnumFieldTypeName` – the unique enum definition that corresponds to a specific enum dimension field name in section 4.
	
	`uniqueEnumFieldItemName` – field item name for the dropdown list
	
	Repeat the Display Value Dropdown Item line for the number of options which need to be shown in the Drop-down list
	
	Repeat the block for each different enum dimensions definition required in the asset.

Device Class Definition graphqls Template reference

This section details the device_class_definitions_((assetType))_v((assetTypeNumber)).graphqls.template reference file. This first section is the template file followed by a detailed description of the sections of the file.

```
## | Rule  - Values which has | in the middle are the choices that user has by which only one choice can be chosen. Remove all other choices including the | symbol(s)##
## <> Rule - Values surrounded by <> are the ones that are expected to be changed to valid values. Consider | rule explained above as well. Remove characters < and > as well##
## [] Rule - Values surrounded by [] are optional definitions which either needs to be completely removed or to be used exactly as defined. Consider <> rule explained above as well. Remove characters [ and ] as well ##


[#<Comments for type of asset>]
type <AssetType> implements Asset & Device @asset(
    version: <AssetTypeVersion>
    label: "<Label for Asset Class>"
) {

    ## Repeat below 7 lines for numbers of dimensions to be defined apart from ones defined in Asset or Device class definitions ##
    [#<Comment for Dimension Field>]
    <uniqueFieldName>: <String | Int | Float | Long | Boolean>[!] [@rename(oldName: "<uniqueFieldNameOld>")] [@defaultValue(value: "<String representation of default value>")] [@dsaMapping(path: "<DSA path expression>")] @dimension(
        label: "<Display Label of Field>"
        [unique: <true | false>]
        [searchable: <true | false>]
        [access: "<R | RW>"]
    )

    ## Repeat below 6 lines for numbers of metrics to be defined apart from ones defined in Asset and Device class definitions ##
    [#<Comment for Metric Field>]
    <uniqueFieldName>: <String | Int | Float | Long | Boolean> [@dsaMapping(path: "<DSA path expression>")] @metric(
        label: "<Display Label of Field>"
        [unitName: "<Metric unit name>"]
        [unitSymbol: "<Metric unit symbol>"]
    )



    ## Repeat below 6 lines for numbers of enum dimensions required in the asset. The Type <assetType>_<uniqueEnumFieldTypeName> must match between the enum and this field definition.##
    [#<Comment for Dimension Field>]
    <uniqueFieldName>: <assetType>_<uniqueEnumFieldTypeName> [@dsaMapping(path: "<DSA path expression>")] @dimension(
        label: "<Display Label of Field>"
        [searchable: <true | false>]
        [access: "<R | RW>"]
    )
}

## Repeat below 4 lines for number of different enum dimensions required in the asset##
[enum <assetType>_<uniqueEnumFieldTypeName> {
    <uniqueEnumFieldItemName> @enumValue(label: "<Display Value for Dropdown Item>")
    ## Repeat above 1 line for number of options which needs to be shown in the Dropdownlist ##
}]

## The optional directive @dsaMapping will be defaulted to $<uniqueFieldName> if whole @dsaMapping directive is omitted ##
## The optional directive @rename must be used when intending to rename a field name ##
```

```
## The optional directive @defaultValue for dimension must be used when making a nullable field as a non
nullable field ##
## The optional argument unique will be defaulted to false if whole argument is omitted ##
## The optional argument searchable will be defaulted to false if whole argument is omitted ##
## The optional argument access will be defaulted to RW if whole argument is omitted ##
## The optional argument unitName will be defaulted to "", if whole argument is omitted ##
## The optional argument unitSymbol will be defaulted to "", if whole argument is omitted ##
## The type name defined in enum <assetType>_<uniqueEnumFieldDefinition> must be unique across all
available asset definitions. Hence <assetType> is also added in the type name ##
## The type name must match between the field definition and the enum definition, for the enum values to
be used ##
```

Note: The following definitions must match the corresponding values from the Device Definition file to the Asset Class.

device_discovery_definitions_((assetType))_v((assetTypeNumber)).json.template	**device_class_definitions_((assetType))_v((assetTypeNumber)).graphqls.template**
"assetType": "**<Name for Asset Type>**", "assetTypeVersion": **<Version Number for Asset Type>**,	type **<Name for Asset Type>** implements Asset & Device @asset(version: **<Version Number for Asset Type>**
"dataflow": { **"<uniqueFieldName>":**	**<uniqueFieldName>**: <String | Int | Float | Long | Boolean> [@dsaMapping(path: "<DSA path expression>")] @metric(label: "<Display Label of Field>" [unitName: "<Metric unit name>"] [unitSymbol: "<Metric unit symbol>"])

"dimensionMapping": [
{
"type": "DSA",
"targetPath": "<Source DSA Path to read dimension>",
"fieldName": "<**uniqueFieldName**>"
}
	<uniqueFieldName>: <String | Int | Float | Long | Boolean>[!] [@rename(oldName: "<uniqueFieldNameOld>")] [@defaultValue(value: "<String representation of default value>")] [@dsaMapping(path: "<DSA path expression>")] @dimension(
	label: "<Display Label of Field>"
	[unique: <true | false>]
	[searchable: <true | false>]
	[access: "<R | RW>"]
	)

Onboarding a Message Broker as a device example

This example describes the use of a set of defined files that discover message brokers and onboard them as devices. Throughout the example we will make modifications in incremental steps to the files and highlight them we go for clarity. The following steps will be shown:

- Modifying a field
- Adding a field
- Renaming a field
- Removing a field
- Making a field non-nullable

Let us create two files under a new directory broker-link in /opt/cisco/kinetic/efm_manager/config/discovery/ and place the files device_discovery_definitions_BrokerDevice_v1.json and device_class_definitions_BrokerDevice_v1.graphqls respectively as follows. See the tables below.

And we will use the following file content as version 1. Note that even if one of the files is not updated, the version numbers must be in sync on both definition files and therefore incremented.

device_discovery_definitions_BrokerDevice_v1.json

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 1,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "list brokers | filter $is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Usage/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v1.graphqls

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @asset(
    version: 1
    label: "Broker Device"
) {

    distribution: String @dsaMapping(path: "/Distribution/:value") @dimension(
        label: "Distribution"
        access: "R"
    )

    cpu: Float @dsaMapping(path: "/CPU_Usage/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path: "/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

The files will be show for before and after with the changes highlighted in red.

- **Modifying a field**
    - Changing the display name of a dimension distribution from " Distribution" to " Distribution Name"
    - Changing the unit Symbol of metric data_in from " bytes" to " B"

device_discovery_definitions_BrokerDevice_v1.json	device_discovery_definitions_BrokerDevice_v2.json

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 1,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "list brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v1.graphqls

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 2,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "list brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v2.graphqls

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 1
    label: "Broker Device"
) {

    distribution: String @dsaMapping(path: "/Dis-
tribution/:value") @dimension(
        label: "Distribution"
        access: "R"
    )

    cpu: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path:
"/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 2
    label: "Broker Device"
) {

    distribution: String @dsaMapping(path: "/Dis-
tribution/:value") @dimension(
        label: "Distribution Name"
        access: "R"
    )

    cpu: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path:
"/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "B"
    )
}
```

Example log output:

```
[root@centos7 efm_manager]# sudo service efm-manager stop

[root@centos7 efm_manager]# sudo /opt/cisco/kinetic/efm_manager/bin/load-configurations.sh
********************************************************************************************
Schema migration command started
********************************************************************************************
Configuration Directory: /opt/cisco/kinetic/efm_manager/config
Working Directory:       /opt/cisco/kinetic/efm_manager/app-data
Schema Directory:        /opt/cisco/kinetic/efm_manager/config/discovery
--------------------------------------------------------------------------------------------
Starting Ignite
Acquiring asset manager startup lock
Reading configuration
Loaded 14 configuration file(s)
Installed 1 asset schema version(s)
Installed 1 asset discovery definition(s)
[Start] 'Asset type migration (1 type(s) to be migrated)' job
 + Start task 'Migrate asset type 'BrokerDevice' from  v1 to v2'
[SUCCESS - processed 1 object(s) in 316 ms]
Done

[root@centos7 efm_manager]# sudo service efm-manager start
```

- **Adding a field**
    - Adding a new dimension field hostname which will be read from the source link and must be a searchable field.
    - Adding a new dimension field contact_person, which will be given as input by the user and must be searchable as well.
    - Adding a new Metric field data_out which will be read from the source link.

device_discovery_definitions_BrokerDevice_v2.json	device_discovery_definitions_BrokerDevice_v3.json

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 2,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "list brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v2.graphqls

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 3,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution"
      },
      {
        "type": "DSA",
        "targetPath": "/downstream/System/Host-
name/:value",
        "fieldName": "hostname"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "list brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      },
      "data_out": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataOutPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v3.graphqls

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 2
    label: "Broker Device"
) {

    distribution: String @dsaMapping(path: "/Dis-
tribution/:value") @dimension(
        label: "Distribution Name"
        access: "R"
    )









    cpu: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path:
"/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "b"
    )




}
```

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 3
    label: "Broker Device"
) {

    distribution: String @dsaMapping(path: "/Dis-
tribution/:value") @dimension(
        label: "Distribution Name"
        access: "R"
    )

    hostname: String @dsaMapping(path: "/Host-
name/:value") @dimension(
        label: "Hostname"
        searchable: true
        access: "R"
    )

    contact_person: String @dsaMapping(path:
"$contactPerson") @dimension(
        label: "Contact Person"
        searchable: true
        access: "RW"
    )

    cpu: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path:
"/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "b"
    )

    data_out: Float @dsaMapping(path:
"/Data_Out/:value") @metric(
        label: "Data Out"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

Example log output:

```
[root@centos7 efm_manager]# sudo service efm-manager stop

[root@centos7 efm_manager]# sudo /opt/cisco/kinetic/efm_manager/bin/load-configurations.sh
************************************************************************************************
Schema migration command started
************************************************************************************************
Configuration Directory: /opt/cisco/kinetic/efm_manager/config
Working Directory:       /opt/cisco/kinetic/efm_manager/app-data
Schema Directory:        /opt/cisco/kinetic/efm_manager/config/discovery
------------------------------------------------------------------------------------------------
Starting Ignite
```

```
Acquiring asset manager startup lock
Reading configuration
Loaded 16 configuration file(s)
Installed 1 asset schema version(s)
Installed 1 asset discovery definition(s)
[Start] 'Asset type migration (1 type(s) to be migrated)' job
 + Start task 'Migrate asset type 'BrokerDevice' from  v2 to v3'
[SUCCESS - processed 1 object(s) in 493 ms]
Done

[root@centos7 efm_manager]# sudo service efm-manager start
```

- **Renaming a field**
    - Renaming the dimension field distribution to distribution_name
    - Renaming the metric filed cpu to be called as cpu_usage

device_discovery_definitions_BrokerDevice_v3.json	device_discovery_definitions_BrokerDevice_v4.json

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 3,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution"
      },
      {
        "type": "DSA",
        "targetPath": "/downstream/System/Host-
name/:value",
        "fieldName": "hostname"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "list brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      },
      "data_out": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataOutPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v3.graphqls

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 4,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution_name"
      },
      {
        "type": "DSA",
        "targetPath": "/downstream/System/Host-
name/:value",
        "fieldName": "hostname"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "list brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu_usage": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      },
      "data_out": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataOutPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v4.graphqls

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 3
    label: "Broker Device"
) {

    distribution: String @dsaMapping(path: "/Dis-
tribution/:value") @dimension(
        label: "Distribution Name"
        access: "R"
    )

    hostname: String @dsaMapping(path: "/Host-
name/:value") @dimension(
        label: "Hostname"
        searchable: true
        access: "R"
    )

    contact_person: String @dsaMapping(path:
"$contactPerson") @dimension(
        label: "Contact Person"
        searchable: true
        access: "RW"
    )

    cpu: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path:
"/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "b"
    )

    data_out: Float @dsaMapping(path:
"/Data_Out/:value") @metric(
        label: "Data Out"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 4
    label: "Broker Device"
) {

    distribution_name: String @rename(oldName:
"distribution") @dsaMapping(path: "/Distribu-
tion/:value") @dimension(
        label: "Distribution Name"
        access: "R"
    )

    hostname: String @dsaMapping(path: "/Host-
name/:value") @dimension(
        label: "Hostname"
        searchable: true
        access: "R"
    )

    contact_person: String @dsaMapping(path:
"$contactPerson") @dimension(
        label: "Contact Person"
        searchable: true
        access: "RW"
    )

    cpu_usage: Float @rename(oldName: "cpu")
@dsaMapping(path: "/CPU_Usage/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path:
"/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "b"
    )

    data_out: Float @dsaMapping(path:
"/Data_Out/:value") @metric(
        label: "Data Out"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

Example log output:

```
[root@centos7 efm_manager]# sudo service efm-manager stop

[root@centos7 efm_manager]# sudo /opt/cisco/kinetic/efm_manager/bin/load-configurations.sh
********************************************************************************************
Schema migration command started
********************************************************************************************
Configuration Directory: /opt/cisco/kinetic/efm_manager/config
Working Directory:       /opt/cisco/kinetic/efm_manager/app-data
Schema Directory:        /opt/cisco/kinetic/efm_manager/config/discovery
--------------------------------------------------------------------------------------------
```

```
Starting Ignite
Acquiring asset manager startup lock
Reading configuration
Loaded 18 configuration file(s)
Installed 1 asset schema version(s)
Installed 1 asset discovery definition(s)
[Start] 'Asset type migration (1 type(s) to be migrated)' job
 + Start task 'Migrate asset type 'BrokerDevice' from  v3 to v4'
[SUCCESS - processed 1 object(s) in 558 ms]
Done

[root@centos7 efm_manager]# sudo service efm-manager start
```

- **Removing a field**
    - Remove the dimension field distribution_name
    - Remove the metric field data_in

device_discovery_definitions_BrokerDevice_v4.json	device_discovery_definitions_BrokerDevice_v5.json

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 4,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/sys/dist/:value",
        "fieldName": "distribution_name"
      },
      {
        "type": "DSA",
        "targetPath": "/downstream/System/Host-
name/:value",
        "fieldName": "hostname"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "list brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu_usage": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_in": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataInPerSecond/:value"
        }
      },
      "data_out": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataOutPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v4.graphqls

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 5,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/downstream/System/Host-
name/:value",
        "fieldName": "hostname"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "list brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu_usage": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_out": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataOutPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v5.graphqls

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 4
    label: "Broker Device"
) {

    distribution_name: String @rename(oldName:
"distribution") @dsaMapping(path: "/Distribu-
tion/:value") @dimension(
        label: "Distribution Name"
        access: "R"
    )

    hostname: String @dsaMapping(path: "/Host-
name/:value") @dimension(
        label: "Hostname"
        searchable: true
        access: "R"
    )

    contact_person: String @dsaMapping(path:
"$contactPerson") @dimension(
        label: "Contact Person"
        searchable: true
        access: "RW"
    )

    cpu_usage: Float @rename(oldName: "cpu")
@dsaMapping(path: "/CPU_Usage/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_in: Float @dsaMapping(path:
"/Data_In/:value") @metric(
        label: "Data In"
        unitName: "bytes"
        unitSymbol: "b"
    )

    data_out: Float @dsaMapping(path:
"/Data_Out/:value") @metric(
        label: "Data Out"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 5
    label: "Broker Device"
) {

    hostname: String @dsaMapping(path: "/Host-
name/:value") @dimension(
        label: "Hostname"
        searchable: true
        access: "R"
    )

    contact_person: String @dsaMapping(path:
"$contactPerson") @dimension(
        label: "Contact Person"
        searchable: true
        access: "RW"
    )

    cpu_usage: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_out: Float @dsaMapping(path:
"/Data_Out/:value") @metric(
        label: "Data Out"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

Note: The @rename directive must be removed on the next version of changes on the graphqls file.

Example log output:

```
[root@centos7 efm_manager]# sudo service efm-manager stop

[root@centos7 efm_manager]# sudo /opt/cisco/kinetic/efm_manager/bin/load-configurations.sh
********************************************************************************************
Schema migration command started
********************************************************************************************
Configuration Directory: /opt/cisco/kinetic/efm_manager/config
Working Directory:       /opt/cisco/kinetic/efm_manager/app-data
Schema Directory:        /opt/cisco/kinetic/efm_manager/config/discovery
--------------------------------------------------------------------------------------------
```

```
Starting Ignite
Acquiring asset manager startup lock
Reading configuration
Loaded 20 configuration file(s)
Installed 1 asset schema version(s)
Installed 1 asset discovery definition(s)
[Start] 'Asset type migration (1 type(s) to be migrated)' job
 + Start task 'Migrate asset type 'BrokerDevice' from  v4 to v5'
[SUCCESS - processed 1 object(s) in 532 ms]
Done

[root@centos7 efm_manager]# sudo service efm-manager start
```

- **Making a field non-nullable**

device_discovery_definitions_BrokerDevice_v5.json	device_discovery_definitions_BrokerDevice_v6.json

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 5,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/downstream/System/Host-
name/:value",
        "fieldName": "hostname"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "list brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu_usage": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_out": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataOutPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v5.graphqls

```
[
  {
    "label": "Broker Device Discovery",
    "discoveryClassId": "BrokerDeviceAsset-1.0",
    "assetType": "BrokerDevice",
    "assetTypeVersion": 6,
    "productImageRef": "static/images/cisco.svg",
    "dimensionMapping": [
      {
        "type": "DSA",
        "targetPath": "/$brokerUUID",
        "fieldName": "serial"
      },
      {
        "type": "DSA",
        "targetPath": "/downstream/System/Host-
name/:value",
        "fieldName": "hostname"
      }
    ],
    "defaultValues": {
      "vendor": "Cisco",
      "product": "Dart Broker"
    },
    "discoveries": [
      {
        "type": "DQL",
        "query": "list brokers | filter
$is=\"dsa/broker\" and $brokerUUID"
      }
    ],
    "dataflow": {
      "cpu_usage": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/downstream/System/CPU_Us-
age/:value"
        }
      },
      "data_out": {
        "type": "write-through-dataflow",
        "config": {
          "path": "/sys/dataOutPerSecond/:value"
        }
      }
    }
  }
]
```

device_class_definitions_BrokerDevice_v6.graphqls

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 5
    label: "Broker Device"
) {

    hostname: String @dsaMapping(path: "/Host-
name/:value") @dimension(
        label: "Hostname"
        searchable: true
        access: "R"
    )

    contact_person: String @dsaMapping(path:
"$contactPerson") @dimension(
        label: "Contact Person"
        searchable: true
        access: "RW"
    )









    cpu_usage: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_out: Float @dsaMapping(path:
"/Data_Out/:value") @metric(
        label: "Data Out"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

```
# BrokerDevice Details
type BrokerDevice implements Asset & Device @as-
set(
    version: 6
    label: "Broker Device"
) {

    hostname: String @dsaMapping(path: "/Host-
name/:value") @dimension(
        label: "Hostname"
        searchable: true
        access: "R"
    )

    contact_person: String! @defaultValue(value:
"Joe") @dsaMapping(path: "$contactPerson") @dimen-
sion(
        label: "Contact Person"
        searchable: true
        access: "RW"
    )


    contact_method: String! @defaultValue(value:
"Telephone") @dsaMapping(path: "$contactMethod")
@dimension(
        label: "Contact Method"
        searchable: true
        access: "RW"
    )

    cpu_usage: Float @dsaMapping(path: "/CPU_Us-
age/:value") @metric(
        label: "CPU Usage"
        unitName: "percentage"
        unitSymbol: "%"
    )

    data_out: Float @dsaMapping(path:
"/Data_Out/:value") @metric(
        label: "Data Out"
        unitName: "bytes"
        unitSymbol: "bytes"
    )
}
```

Example log output:

```
[root@centos7 efm_manager]# sudo service efm-manager stop

[root@centos7 efm_manager]# sudo /opt/cisco/kinetic/efm_manager/bin/load-configurations.sh
********************************************************************************************
Schema migration command started
********************************************************************************************
Configuration Directory: /opt/cisco/kinetic/efm_manager/config
Working Directory:       /opt/cisco/kinetic/efm_manager/app-data
Schema Directory:        /opt/cisco/kinetic/efm_manager/config/discovery
--------------------------------------------------------------------------------------------
Starting Ignite
Acquiring asset manager startup lock
Reading configuration
```

```
Loaded 22 configuration file(s)
Installed 1 asset schema version(s)
Installed 1 asset discovery definition(s)
[Start] 'Asset type migration (1 type(s) to be migrated)' job
 + Start task 'Migrate asset type 'BrokerDevice' from  v5 to v6'
[SUCCESS - processed 1 object(s) in 515 ms]
Done

[root@centos7 efm_manager]# sudo service efm-manager start
```

Restarting EFM Manager after adding or modifying Device Definition files

As a reminder, for all changes to onboarding configuration files require restarting the EFM Asset Manager for them to be read as well as running the load-cofngurations.sh script before restarting. This is accomplished with the following steps:

1. Stop asset manager:
   `sudo service efm-manager stop`
2. Run load configuration script.
   `sudo $EFM_ROOT/efm_manager/bin/load-configurations.sh`
   Look for the SUCCESS Message
3. Start asset manager:
   `sudo service efm-manager start`

Onboarding FAQ

Q: Should I stop the efm manager before I start to add new definitions files?

No. It is not required to stop the efm manager to add new configuration files to file system. But for the new files to be read by efm manager, the following three steps are required:

1. Stop efm-manager
2. Run load_configuration.sh script
3. Start efm-manager

Q: Do I need to do the above steps for every new version of the definition file?

Yes. the above mentioned three steps are must to followed for the new definitions to take effect.

Q. What if a dimension is defined in json and not defined in graphqls?

This field will not be shown in UI and when trying to save the device, an error will be generated that the field is missing.

Q. What if a dimension is defined in graphqls and not defined in json?

This field will be shown in UI as new field and this will be a new dimension for the device.

Q. What if a metric is defined in json and not defined in graphqls?

This metric will not be carried over to asset DSLink and there will not be any error specific to that.

Q. What if a metric is defined in graphqls and not defined in json?

This metric will be displayed in asset DSLink but will be empty.

Q: Should a field name be unique in the same asset?

Yes. The field name must be unique within a defined asset.

Q. Should a field name be unique across different version of same asset?

The field name must be unique considering all the versions of same asset.

Q. Should the field name be unique across different types of asset?

No. Each of the assets can have its own dimension and they could be used in whichever way required for the specific asset.

Q. Should the enum fieldtype name be unique across different asset?

YES. This is very important to have different fieldtype name for enum across different assets.

Q. Can a comment be added anywhere in the json file?

No. The json file cannot contain any comments.

Q. Can a comment be added to graphql file? Can it be added anywhere?

Yes. The comment is recommended to be added just before the start of a field definition or a type definition. But graphql file will allow comments to be added anywhere. The comment line starts with #.

Q. Can I read the same value from source as dimension and metric, with two different field names and use it in the graphqls file.

Yes. it is possible.

Q. Will it break if any dimension related argument is used in metric and vice versa?

No. It will not break anything. Those additional arguments are ignored.

Q. Where can I place the config files for on-boarding?

{EFM_INSTALLATION_PATH}/efm_manager/config/discovery/

Q. Will it break if I have older versions of the asset definition files in the file system?

No. It will not break anything and in fact its recommended to have the older versions of file as well, so its easily understandable for the user to see changes between versions.

Q. Why two files are required for on-boarding a new device type?

There are two files which drives the on-boarding of a device type.

- A Json file, which has details on the dimensions to read from source dslink, a field name for it, the source DQL Query to discover the device, default values for any fields and path from where the mtric needs to be read from the source.
- A corresponding graphqls, file which will have details about the way its stored in the asset DSLink and how it will be shown in EFM manager UI as well.

Q. What do I need the serial field for?

With EFM 1.6, the serial field is the primary key. It is used to uniquely identify the device.

Q. Can I leave the @rename directive in the next version of changes on graphqls file?

No. The @rename directive must be removed on the next version of changes on the graphqls file.

Q. Do I need to have @defaultvalue directive for every non nullable(mandatory) dimension?

Yes, if it is not the first version of the definition file. In case of changed version of file for migration, this directive is mandatory, as this will be used in migrating the already accepted devices from older version to newer version.

Q. What are the specific details of the each of the field, argument, type?

Refer to the documentation help for exact definition, syntax template etc.,

Q: What if the schema definition files are duplicated with same content?

Nothing will be affected, as the migration is triggered only based on the assettype and version number changes.

Q: What if in the config files, the contents were changed, but the version number is not increased?

Nothing will be affected, as the migration is triggered only based on the assettype and version number changes.

Q: What if in the json config file, the version is increased and not in graphqls file?

The load configuration script will error out expecting the similar version file of graphqls.

Q: What if in the graphqls config file, the version is increased and not in json file?

The load configuration script will error out expecting the similar version file of json.

Q. Do I have to stick to the file naming convention?

The file naming is not important as long as the extension are json and graphqls for the two files. The recommendation is to have name like below where assetType and version numbers are mentioned.

- device_discovery_definitions_((assetType))_v((assetTypeNumber)).json
- device_class_definitions_((assetType))_v((assetTypeNumber)).graphqls

Q: Can I change the data type of a field in a new version?

No. Changing the data type of a field is not supported.

Device Simulator Link

The sensor-simulation DSLink is an optional component of the EFM Asset Manager. It is used for testing and demonstration purposes and generates virtual devices that are published to the message broker. Since the sensor-simulator devices are defined using the Cisco EFM Device Object Model Standard structure, they will be detected properly by the Asset Manager.

The System Administrator can decide to disable or remove the device-simulator DSLink if desired.

By default, the device simulator performs the following:

- Generates 10 virtual devices, 5 temperature and 5 vibration devices
- The device update interval is every 60000 ms (milliseconds)
- A sinus finishing time of 15 minutes

The metrics for the device-simulator DSLink can be modified by the System Administrator tool or in the dataflow Editor under the device-simulator DSLink, but not in the Asset Manager. The following metrics are user definable and can be set to new values:

- Simulated Temperature Device Count
- Simulated Vibration Device Count
- Sinus Finishing Time (in minutes)

DSLink Input Definitions for generating devices

This DSLink uses the following as input in generating the device simulation. They can be modified using the Dataflow Editor or the System Administrator for the device-simulator DSLink.

- Device Update Interval – This defines the interval in which the device reading has to be updated. Default – 60000 msec.
- Simulated Temperature Device Count – This defines the number of temperature devices that needs to be simulated. Default – 5
- Simulated Vibration Device Count – This defines the number of Vibration devices that needs to be simulated. Default – 5
- Sinus Finishing Time: This defines the time required for the sinus curve to complete. Default – 15 minutes.

Device Simulation Details

Simulated Temperature Device:

According to the number specified in Simulated Temperature Device Count, the Devices are labelled. By default, the labels are Temperature Device 1, Temperature Device 2 thru Temperature Device 5.

On changing the “Simulated Temperature Device Count” value, those specific number of NEW devices are simulated. For example, if it set to 20, then Temperature Device 1 thru Temperature Device 20 will be generated.

Each Device will have Temperature value simulated from 18 to 22 in a sinus curve. The value is represented as °C.

Simulated Vibration Device

According to the number specified in Simulated Vibration Device Count, the Devices are labelled. By default, the labels are Vibration Device 1, Vibration Device 2 thru Vibration Device 5.

On changing the “Simulated Vibration Device Count” value, those specific number of NEW devices are simulated. For example, if it set to 20, then Vibration Device 1 thru Vibration Device 20 would be generated.

Each device will contain X Axis, Y Axis and Z Axis simulated values for Acceleration, Distance and Frequency.

Acceleration is defined as a value simulated from –1 to +1 in a sinus curve. The value is represented in $\frac{mm}{s^2}$.

Distance is defined as values between 0.7 and 3.7 (with stronger vibration) and between 0 and 0.2. The value is represented in mm.

Frequency – Upper values between 10 and 20 (with stronger vibration) and between 1 and 5. The value is represented in hertz.

Obtaining documentation and submitting a service request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

Subscribe to the *What's New in Cisco Product Documentation* as a Really Simple Syndication (RSS) feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service and Cisco currently supports RSS Version 2.0.