Cisco Kinetic PoV

Deployment Guide

Date June 19, 2019

Authors Robin Lindelius (rlindeli), Moa Nissar (mnissar), Leandros Talman (ltalman)

**Disclaimer**

The software made available as part of this guide is only intended for Proof-of-Value purposes. The software is thereby not suitable for use in live production environments without going through the necessary scrutiny such an environment entail. Furthermore, the use of Raspberry Pi in a production environment is not recommended due to the lack of security and redundancy features such an environment entail. This includes, but are not limited to, software validation during the boot process as well as redundant power supplies.

# Introduction

This is a deployment guide for a Proof-of-Value of the Kinetic platform. The PoV aims to showcase the benefits that come with using the Kinetic platform in an IoT environment. The PoV utilizes the GMM, *Gateway Management Module*, and DCM, *Data Control Module*, modules of the Kinetic platform. Kinetic GMM is a secure, scalable module to provision, manage and monitor IoT gateways. DCM moves data from the edge to the right applications in public and private clouds, while allowing you to filter, throttle, and transform data through extensive rule sets.

The deployment consists of at least one Raspberry Pi with a Sense HAT module connected to an IR8x9 gateway. Through the Sense HAT module, the Raspberry Pi will be able to act as multiple sensors that sends data to the gateway. The gateway will retrieve that data, apply rule sets to that data and then publish the result to its internal broker in the Kinetic cloud. The result can then be retrieved from the internal broker using the Kinetic API.

**Minimum Requirements**

1 x Raspberry Pi with Sense HAT module (Will act as multiple sensors)

1 x IR8x9 Gateway

1 x Kinetic Dashboard

# Kinetic

The data that is produced by all the things around us is a high-value asset to businesses, but only if they can make full use of it. This can be a challenge when working with disparate things and a variety of applications that may live in edge or fog nodes, your data center, private and/or public clouds.

Cisco Kinetic is a platform that simplifies the use of data. With its different modules, it has three key functions.

It can help to:

* *Extract* *data* from disparate sources (“things”) regardless of protocol
* *Compute* *data* anywhere from the edge to a destination to provide processing where it’s needed, which enables fast decision at the point of action, dramatically reduces latency and make the most efficient use of network resources.
* *Move data* from connected things to various applications, at the right time. It distributes data in multi-cloud, multi-party, and multi-location simulations. The platform also helps enforcing policies, to give control over which data goes where, and when. All to gain maximum business benefit.

This PoV utilizes two of the Kinetic platform’s modules – Gateway Management Module (GMM) and Data Control Module (DCM).



Figure : DCM and GMM modules

## GMM

The Gateway Management Module is a cloud-based management tool for Cisco Gateways, that delivers a secure and scalable tool to provision, manage and monitor IoT gateways. It provides ongoing, remote visibility and control, of Cisco industrial IoT gateways (IR809 and IR829) from a desktop browser. Moreover, it can bring new gateways online and operational within minutes, as the gateway automatically connects to Kinetic and gets configured. All network configurations can be pushed from a single point in the cloud across all gateways based on user defined templates.

## DCM

The Data Control Module can move data from the edge to the right applications in any cloud, according to pre-set policies. For example, DCM can be used to deploy IOx fog applications, which enables the IR8x9 gateway to be used as a compute platform. The applications installed using DCM can communicate with, and extract data from, connected sensors, to be able to share observations for analysis.

# Architecture

The PoV consist of three different parts: The application that makes the sensor data consumable on the Raspberry Pi, the IOx application that retrieves the sensor data from the Pi, and the application that extracts the processed data from the Kinetic cloud.



Figure : Architectural overview

# Deployment

<https://devnetsupport.cisco.com/hc/en-us/articles/360000135988-How-To-Build-Cisco-Kinetic-Data-Control-Module-Demo>

## Setting up the Raspberry Pi

The Raspberry Pi is a cheap and user-friendly way to simulate a sensor as it is a full computer in portable format. It can thus be used to emulate any sensor with relative ease, requiring only a change in information transport protocol, be it MQTT, PROFINET, or any other industry standard. Exactly because of this diversity, we opted to keep the concept simple and instead used an implementation that makes use of HTTP. Whatever the protocol used, the Pi needs to be able to:

1. Listen for sensor information requests
2. Poll the sensor hardware
3. Return the relevant sensor data as response to the request

The demonstration described in this document makes use of a Sense HAT expansion board developed by the Raspberry Pi Foundation which contains a host of sensors, a multi-directional switch and an 8x8 grid of individually addressable RGB LEDs. It simply connects to the Raspberry Pi by clicking onto the 40-pin connector on the Pi board. To poll its sensors, an official Python package is available that adds functionality to grab sensor information and return it in floating-point values, normalized to common SI units (e.g. pressure is given in hPa, temperature in C, etc.). Once this library is added, all that is required is to host a web server that will interpret properly addressed GET-requests as sensor polling commands.

The Python package Flask can be used to this end, as it allows function calls based on URL in GET-requests. Thus a browser can simply be pointed to e.g. “http://[pi]/sensor/temperature” and a JSON-formatted return value would be the temperature in Celsius.**Error! Hyperlink reference not valid.**

### Initializing the Raspberry Pi

To simplify deployment of the Pi and remove the need for manual interaction after it has been powered on, we can use the application PiBakery to create a custom bootable SD-card. This will execute of series of commands on first boot that will:

1. Install the required Python packages
2. Copy the scripts that run the web server and do sensor polling
3. Install the scripts as a service that will keep running

To write this custom bootable SD-card, follow these instructions

1. Go to the [PiBakery](https://www.pibakery.org/) site and download the application. At the time of writing, version 2 is in beta. This version allows you to add your own image of Raspbian to rewrite rather than be dependent on the version that is enclosed with PiBakery, so its use is recommended.
2. Download [Raspbian](https://www.raspberrypi.org/downloads/raspbian/). It is recommended you use the desktop version in case debugging is required. Unzip the archive.
3. Insert the MicroSD card into a card reader connected to the computer
4. Open PiBakery. In case of OSX, be sure to two-click the file and click “Open” as otherwise an unknown developer warning will not allow you to launch the application.
5. Import the file “setup.xml”
6. Click on “Write” in the top right corner of the application
7. Select the unzipped Raspbian image and name of the MicroSD card
8. Click “Write”
9. When finished, copy the files “app.py”, “sensor.py”, “synchTime.sh” and “sensehathttpd.service” onto the SD card labeled “boot”
10. All done! The Pi should be ready to act as a sensor now.

**Care must be taken to boot the Pi only when connected to a network that will allow it to reach the internet to install the required packages. If it boots without connectivity, the script will fail and steps 6-9 will have to be repeated**. If the sequence is performed successfully, it can be done simply once and need not be repeated per demonstration, though it is always good practice that the Pi be entirely wiped clean.

For those interested in the script that alters the image of the Raspberry Pi, below is an explanation of each step:

1. **Enable VNC at boot**: allows remote desktop connections to the Pi in case of debugging work.
2. **Install sense-hat**: installs scripts necessary for polling the sensor hardware
3. **Install python-gevent**: installs a Python package necessary for running a HTTP server, required for Flask
4. **Install python-flask**: Installs the Python package Flask that can respond to HTTP GET requests and execute functions per URL
5. **Install htpdate**: This package can be used to set the system time by using HTTP requests. The Raspberry Pi has no system clock, so all timing is done by software clocks running on the CPU. This is a poor timekeeping device. By installing the package and running it every minute against a response from google.com, the time can be kept more accurate. This is useful for both the time formatting in the dashboard and for general usage of the Pi, as connectivity can be hampered when the device time is not correct (HTTPS connections can be refused, for instance)
6. **sudo \mv /boot/syncTime.sh /home/pi**: moves the syncTime.sh script that runs to synchronise the time with google.com from the boot directory to the home directory
7. **chmod +x /home/pi/syncTime.sh**: sets execution permission to the script
8. **sudo timedatectl set-ntp false**: turns off NTP to allow the time to be set by the htpdate package
9. **sudo htpdate -s google.com**: fires a one-shot synchronization of the system clock
10. **sudo mv /boot/app.py /home/pi/**: moves the web interface script to the home directory
11. **sudo mv /boot/sensor.py /home/pi/**: moves the sensor polling script to the home directory
12. **sudo mv /boot/sensehathttpd.service /lib/systemd/system/**: moves the service specification file to the services directory
13. **sudo systemctl enable sensehathttpd**: enables the service
14. Create new cron job: Installs a “cron job” to execute the synchronization of the system clock every minute

### Claiming the Raspberry Pi in Kinetic

First connect the Raspberry Pi to the Gateway using an ethernet cable, then go to the Gateway page in Kinetic and click on the name of the gateway you’ve connected the Pi to, see Figure 3.

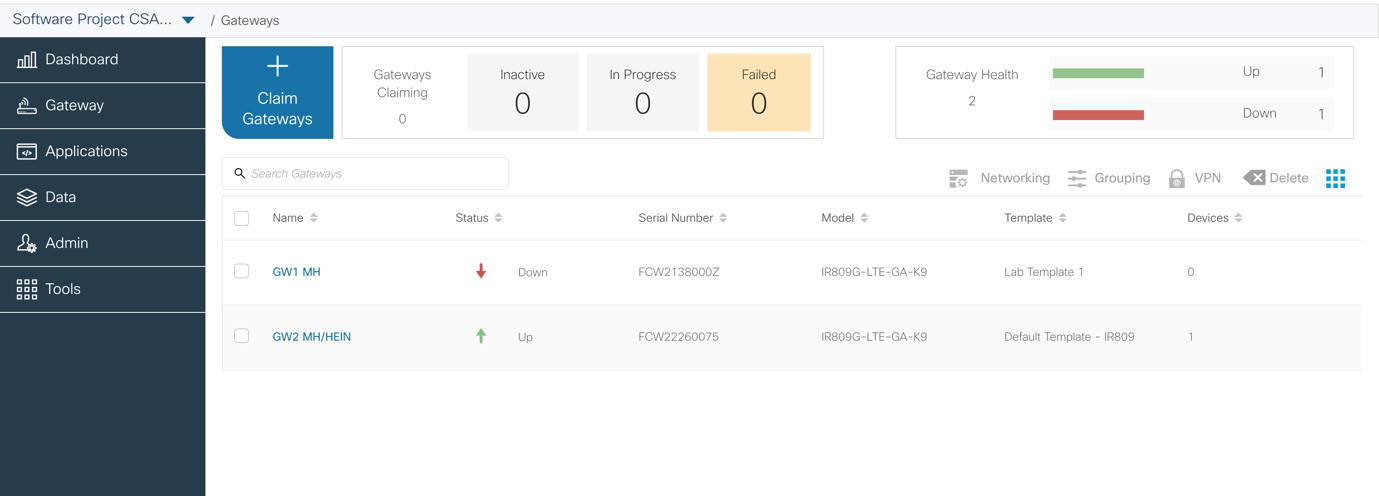


Figure : Gateway page

Go to the Devices tab and press Discover Devices, see Figure 4

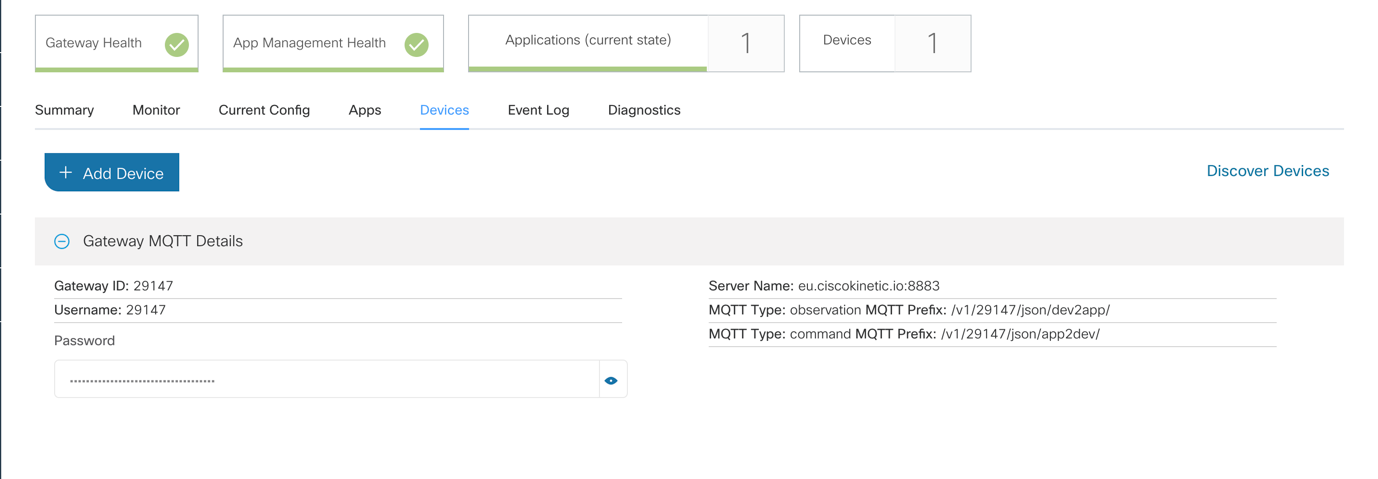


Figure : Discover devices

Name the Pi assetpi and press Add, see Figure 5.

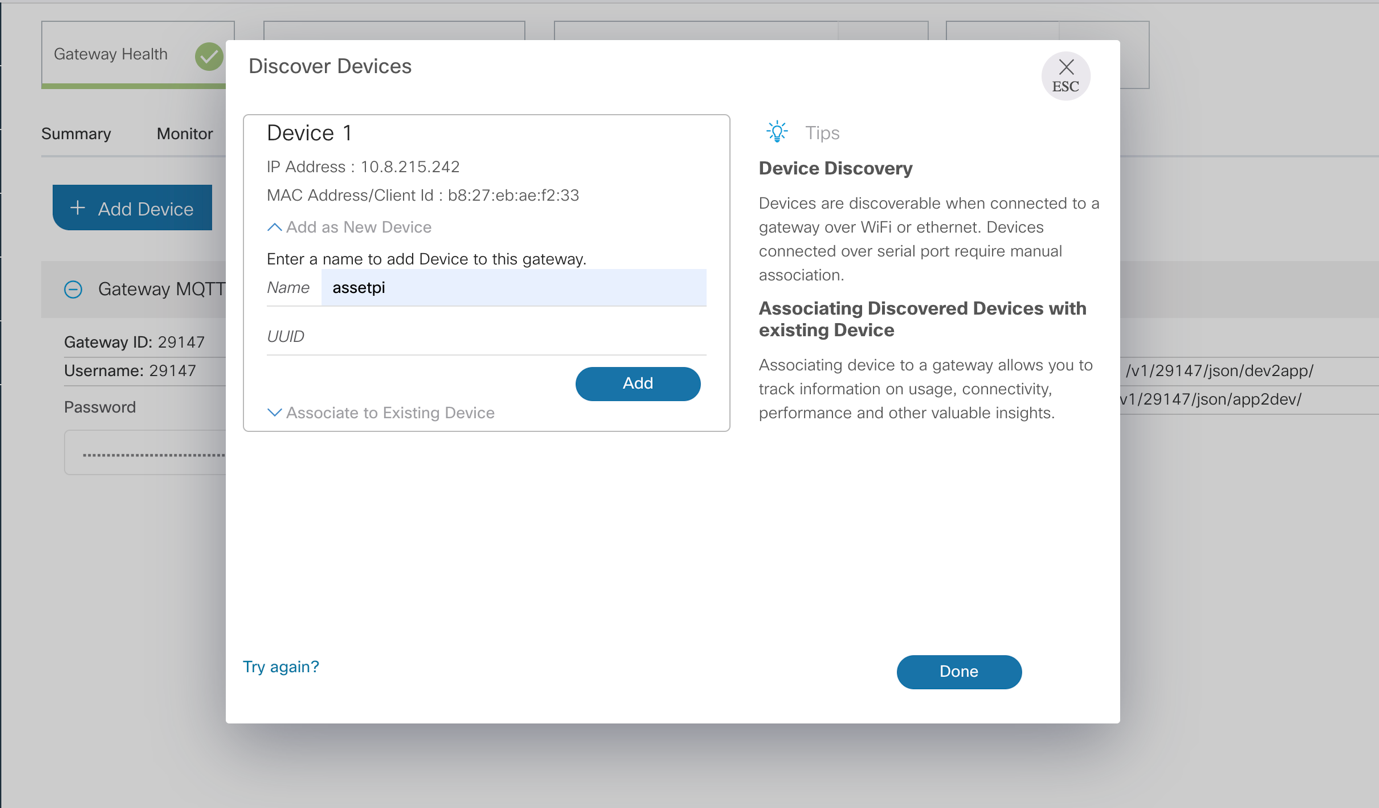


Figure : Name the Raspberry Pi

When the Pi has been added, press Done, see Figure 6.

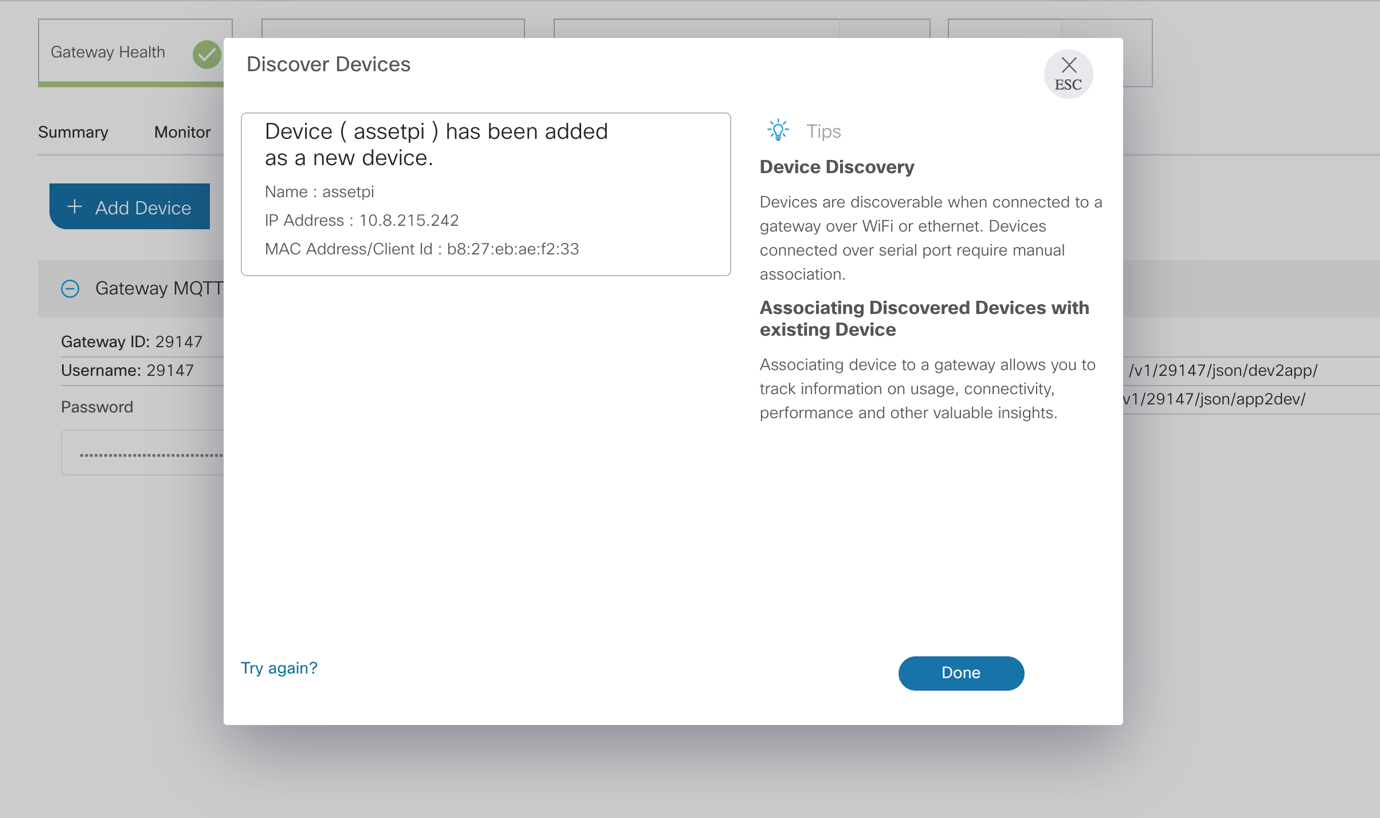


Figure : New device added

Locate the internal IP address of the newly added assetpi, see Figure 7, and copy it for later.

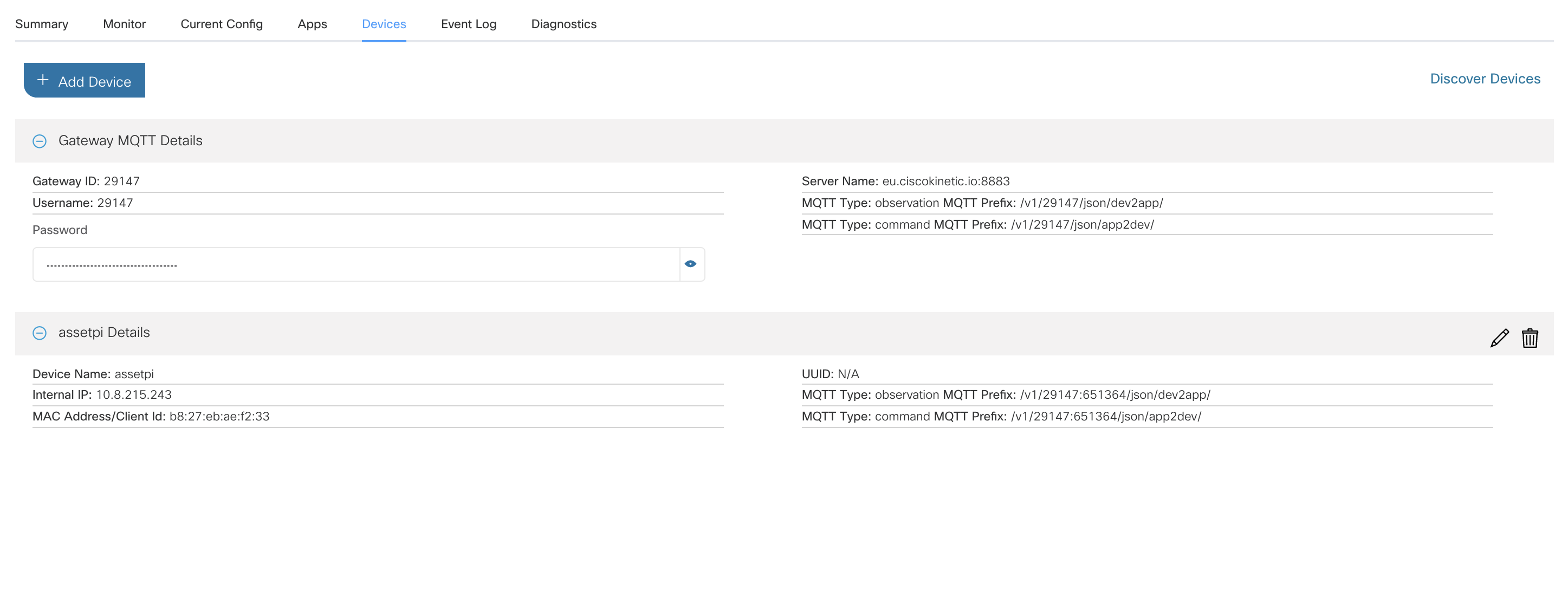


Figure : Device information on Kinetic

## The IOx Application

The IOx application being used is the *iotsp-edge-rule-engine*[[1]](#footnote-2) provided by Cisco DevNet.

### Making changes to the IOx application

To make changes to the IOx application code:

Make the necessary changes to the java code located under the “src” folder. Build the jar package through the terminal by:

cd iotsp-edge-rule-engine

./gradlew clean build

Copy the jar package located under “build/libs/” to the “IOxPackage” folder.

Build the IOx application through the terminal by entering the “iox-client” folder and then run:

**IMPORTANT NOTE:**

*The contents of the IOxPackage directory to be built using the ioxclient in the iox-client directory MUST be built in a LINUX environment. Building it in a local macbook environment will not work. Ubuntu 16 is recommended.*

cd iox-client

./ioxclient package ../IOxPackage

### Deploying the IOx application

Deploy the application to Kinetic, see Figure 8, by uploading the *package.tar.gz* archive from the IOxPackage folder.

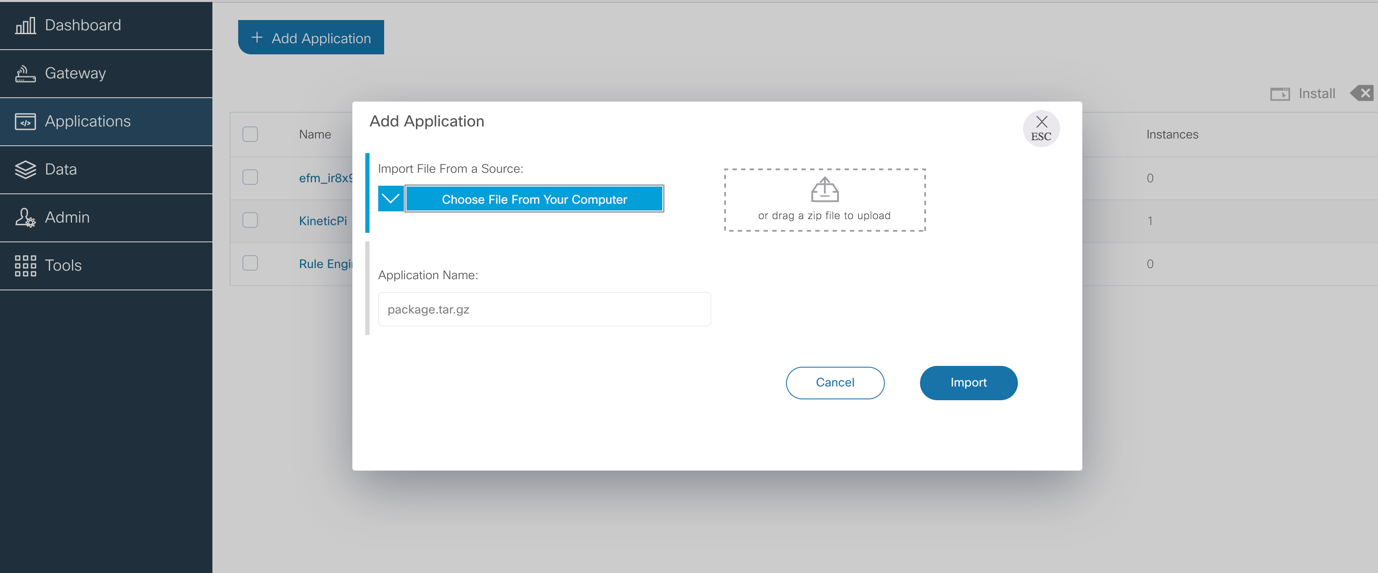


Figure : Add new application on Kinetic

Select the KineticPi application and press Install, see Figure 9.

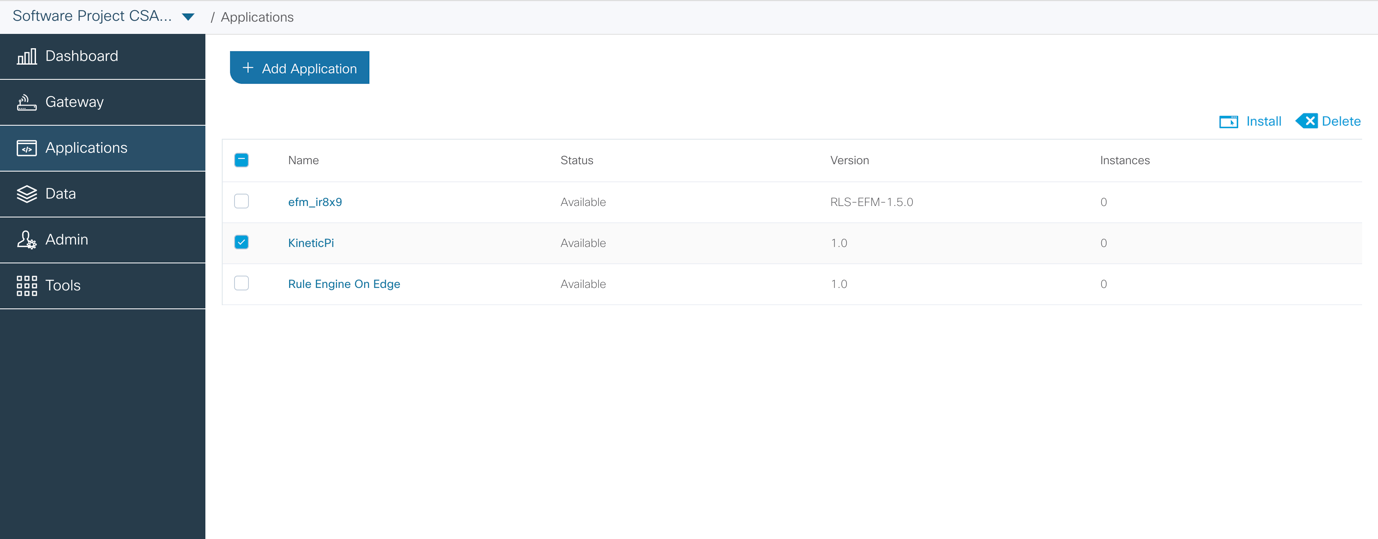


Figure : Install application

Choose the gateway you want to install the application on, see Figure 10.

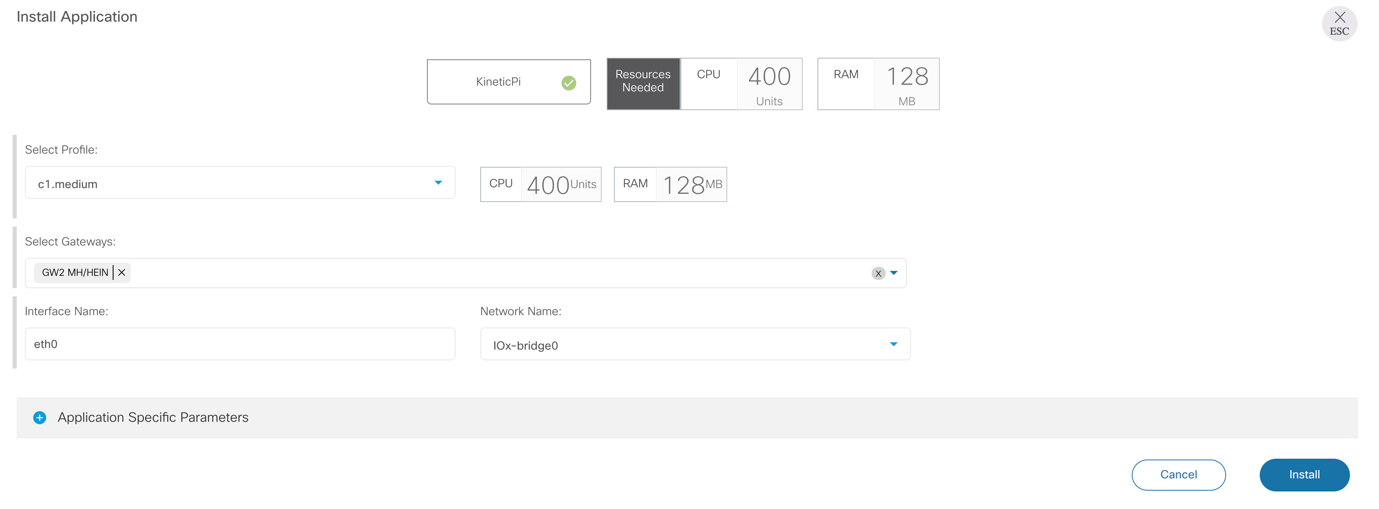


Figure : Installing application on gateway

Expand the Application Specific Parameters tab and enter the internal IP address of the assetpi you previously copied, see Figure 7 under 3.2.2 Claiming the Raspberry Pi in Kinetic, in the “device1\_ip (Optional)” field and enter the port number of the web server running on the Pi, which is by default 5000, in the device1\_port (Optional) field, see Figure 11, and then press Install.

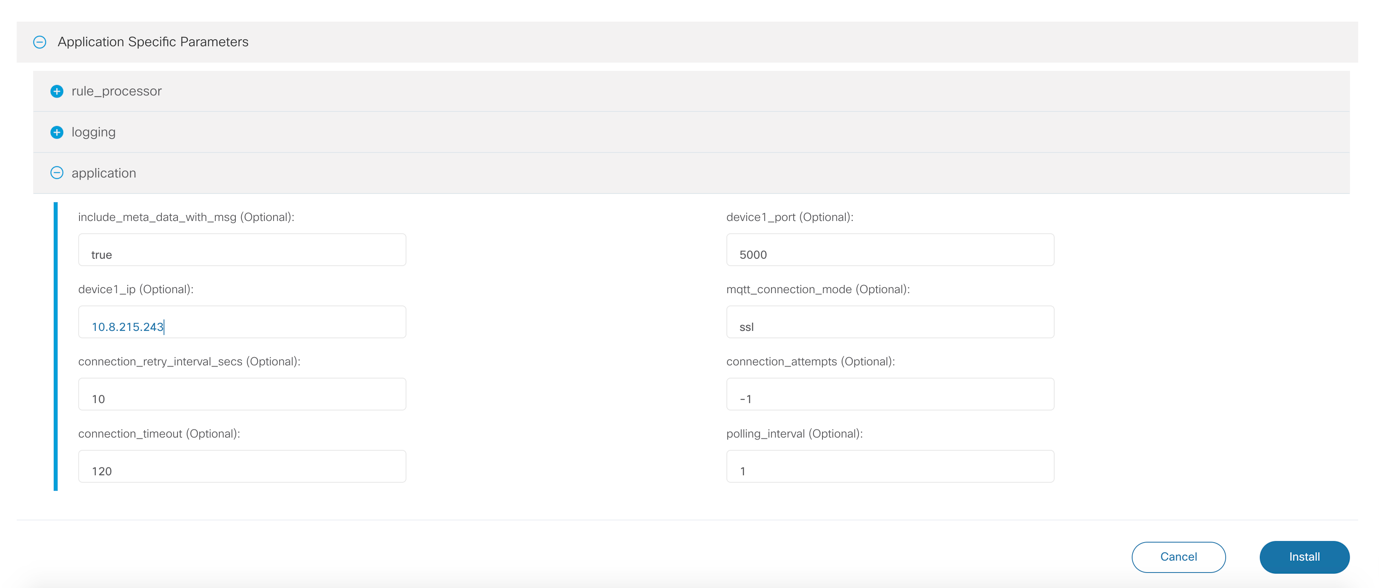


Figure : Configuring application specific information

## Extracting the data

All required files can be found in the Dashboard directoy

Now that the data is being sent to the Kinetic Cloud, we need to extract it with a separate script to simulate a data grabbing mechanism that would display or process it in some other place, e.g. a laptop you are taking with you for the demonstration. For this we have a script, ‘data.py’ which can be run with some parameters outlined below in Section 3.3.1. Beside these parameters, you will also need a TLS certificate to securely connect to the Kinetic server. This certificate file is enclosed in ‘cert.crt’. Be sure to keep both files in the same directory when running the script.

Before running this script, you will need a Python package called Pika. For this, run the terminal command (\*nix, for Windows you will need to use the package manager that comes with your distribution of Python).

sudo pip3 install pika

Next, we need to know the parameters needed to pass on to the script.

### Generating an API Key

The parameter mentioned above is an API Key, composed of a username and password, separated by a full stop, e.g. “X.Y” where X is the username and Y is the password. Both are alphanumeric strings. To create this key, go to the Kinetic dashboard and go to "Tools-API Keys":

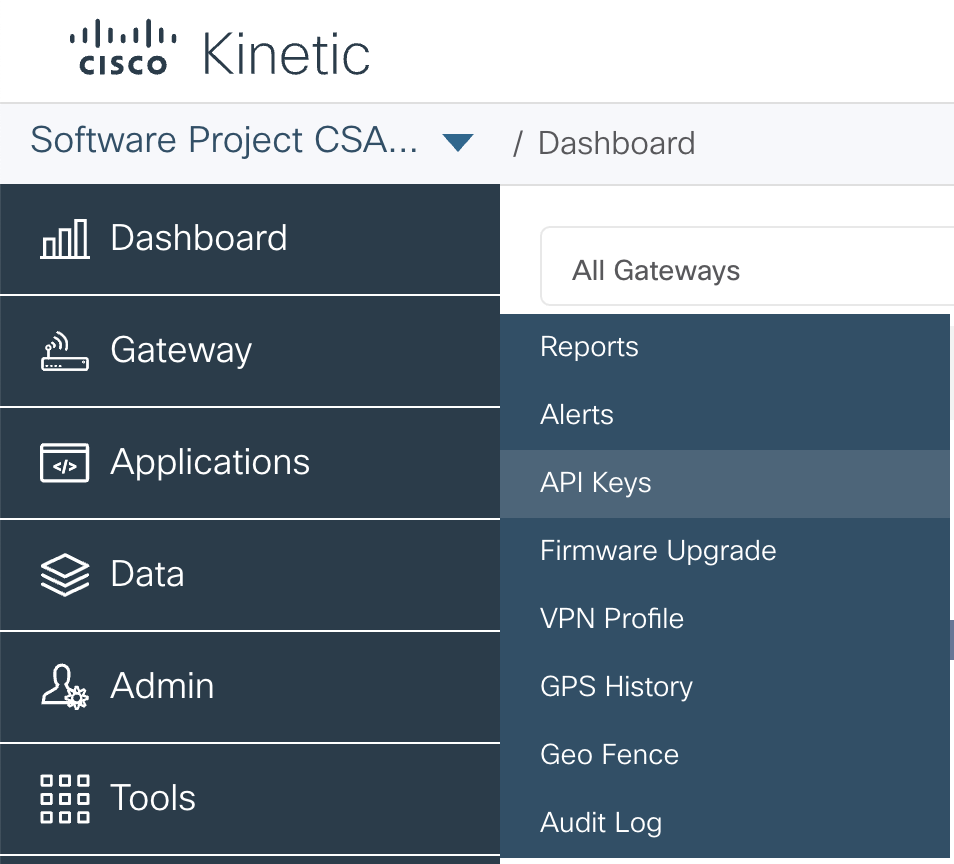
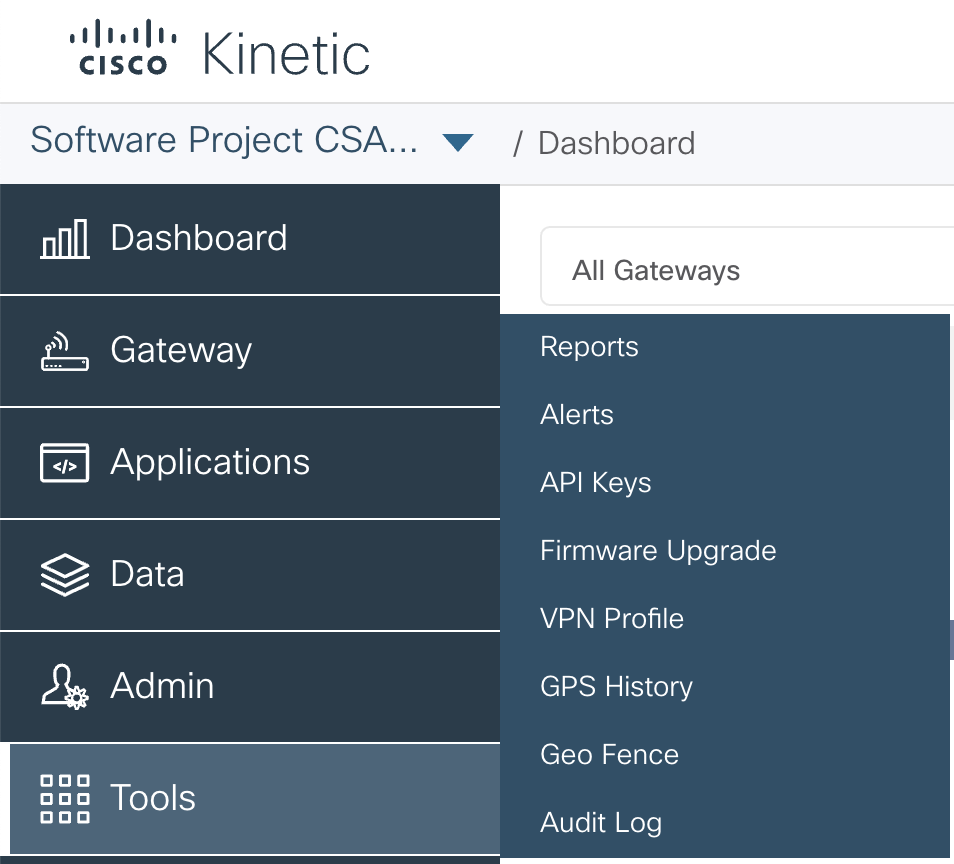


Figure : Generating API Keys

Now click on "Generate API Key"

And fill in the name and Organisation details. Tick the "Data Management" tickbox and click "Generate". The API key will be presented below on the same page. **Be sure to keep it in a known place, as it cannot be generated again!**

In the file ‘run.sh’ you will see the following line:

python3 data.py -r REGION.ciscokinetic.io -e ORGID-EXC -u UNAME -P PASS

Replace:

* ‘REGION’ with ‘eu’ or ‘us’ (dependent on your location)
* ‘ORGID’ with the number at the end of the url when logged into the Kinetic dashboard (e.g. the X at the end of the URL https://(eu/us).ciscokinetic.io/organizations/**X**)
* ‘UNAME’ and ‘PASS’ with the username and password from the API key as described above

The script will run perpetually and outputs data into a file in the same directory, called ‘data.txt’. Only the most recent values from the sensors will be stored in this file, i.e. no historical data is saved. You can run the script at this point and will see the data.txt file change when opening it, but it is preferable to display it in a dashboard!

**At times the script may crash, resulting in the data not updating. We believe this lies in a serverside issue. Simply rerun the script by executing ‘run.sh’ again.**

## Displaying data in Freeboard

Now that we can save the data locally, we need to display it in a nice graphical way. This can be accomplished with the free open source software Freeboard, a simple to setup dashboard frontend. To get the most up-to-date version of this, go to <https://github.com/Freeboard/freeboard>

We also recommend getting HighCharts, a third party plugin for prettier time series, which is also required when using our alternate dashboard: <https://github.com/onlinux/freeboard-dynamic-highcharts-plugin>. If you intend to use the premade dashboards that are included, you will need this plugin.

To make everything work together, copy all the files in the DataExtraction directory to the freeboard directory (the same directory as index.html is located).

Now execute ‘run.sh’ with the terminal command

./run.sh

If a permission error occurs, use the command sudo chmod +x run.sh

before executing it again. **If you are using Windows, executing the ‘run.sh’ file will not work. Copy and paste the line in the file directly into the terminal and hit enter to get it running.**

Now use the command

python3 server.py

to run a small server that presents the dashboard and is able to grab the data out of the textfile. Go to <localhost:8003#source=dashboard.json> and a preconfigured dashboard will be shown, using polled data that changes each second. To make alterations to this, click the wrench icon at the top. Play around with the widgets to see what works well for a demonstration and save the settings by clicking the ‘save freeboard’ button in the top left. To reuse this, save it in the same directory as the freeboard is being served from, and use the URL above, replacing ‘dashboard.json’ with the filename of the newly configured dashboard. An alternate dashboard is also available, requiring the plugin, at the link [http://localhost:8003#source=dashboardalt.json](http://localhost:8003/#source=dashboardalt.json)

1. <https://github.com/CiscoDevNet/iotsp-device-sdk-java/tree/master/examples/iotsp-edge-rule-engine> [↑](#footnote-ref-2)