Getting Started: Google Cloud Injector Quick Start

Prerequisites

- Knowledge of Ignition and Module installation process: Cloud and MQTT Module Installation.
- An existing Google Cloud Platform account with a Project, IoT Core registry, credentials, and device created.
    - Documentation on getting started with the Google Cloud IoT Core can be found here

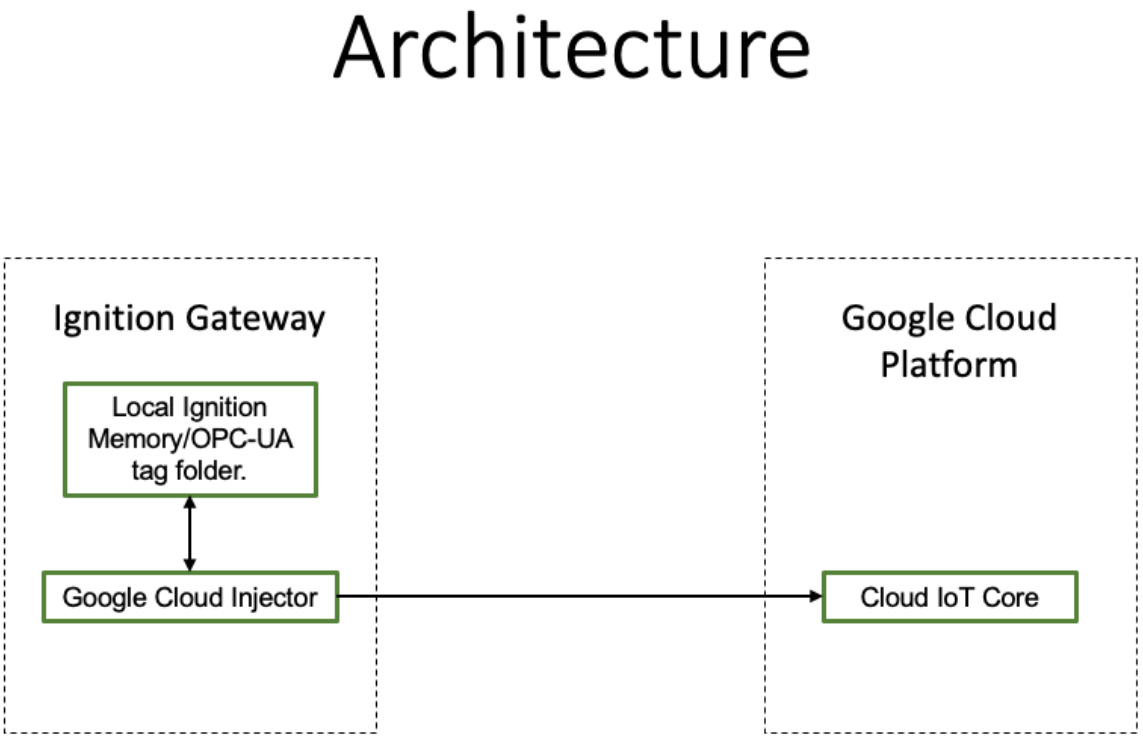
Summary

This tutorial will provide step-by-step instructions for the following:

- Installing the Ignition Gateway Industrial Application Platform with the Google Cloud Injector Module
- Configuring the Google Cloud Injector Module to connect to an existing Cloud IoT Core
- Publishing live Tag data and events to the connected Cloud IoT Core

Upon completion of this module you will have an Ignition Gateway connected and publishing live Tag data to a Cloud IoT Core.

Architecture



Tutorial

Step 1: Download and Install Ignition

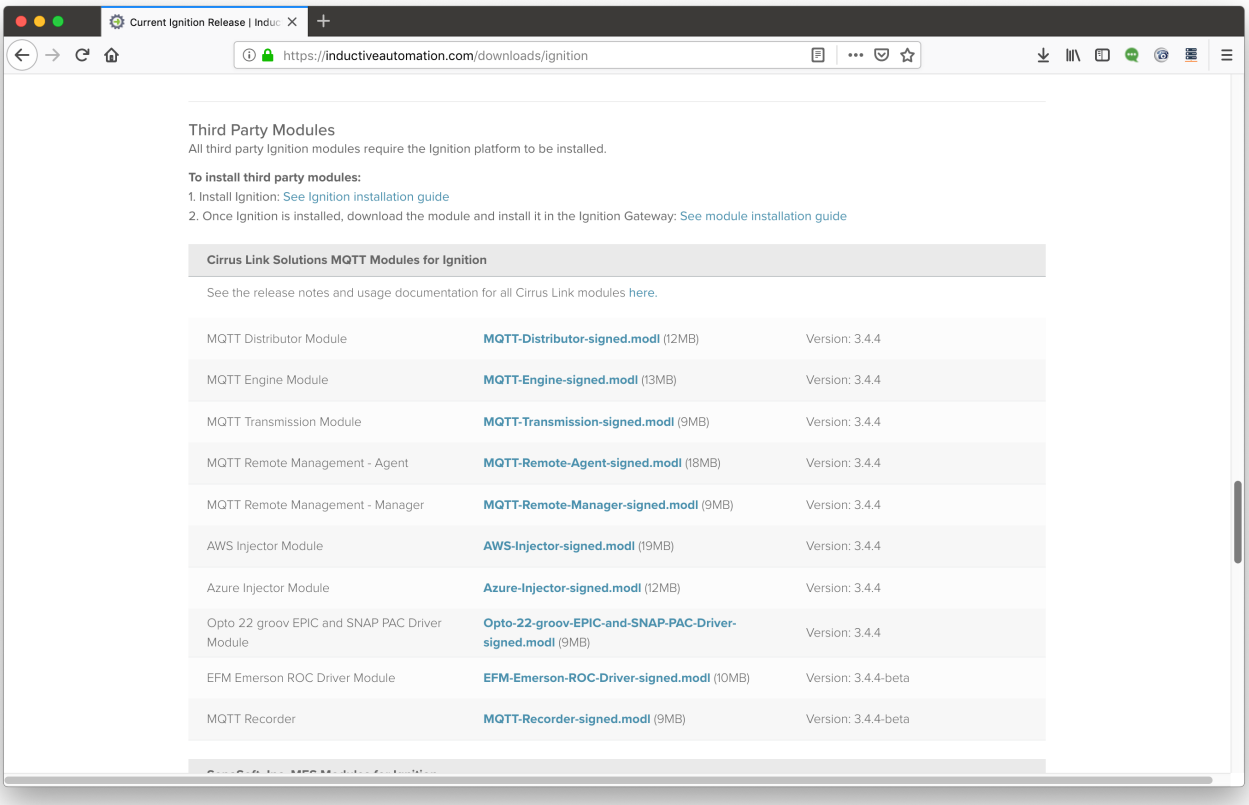
Ignition is an Industrial Application Platform that can be used to create SCADA and HMI solutions. A fully functional Ignition system can be downloaded and run in trial mode. Using Ignition as a tool in this way, we can install the Sparkplug MQTT Modules and observe everything working.

Go to the Inductive Automation download page and download the desired version (select version from the 'Ignition Version' dropdown) of the Ignition installer for Windows, Linux or MacOS; https://inductiveautomation.com/downloads/archive

Once the Ignition installer has been downloaded, follow the instructions provided by Inductive Automation to install and startup Ignition.

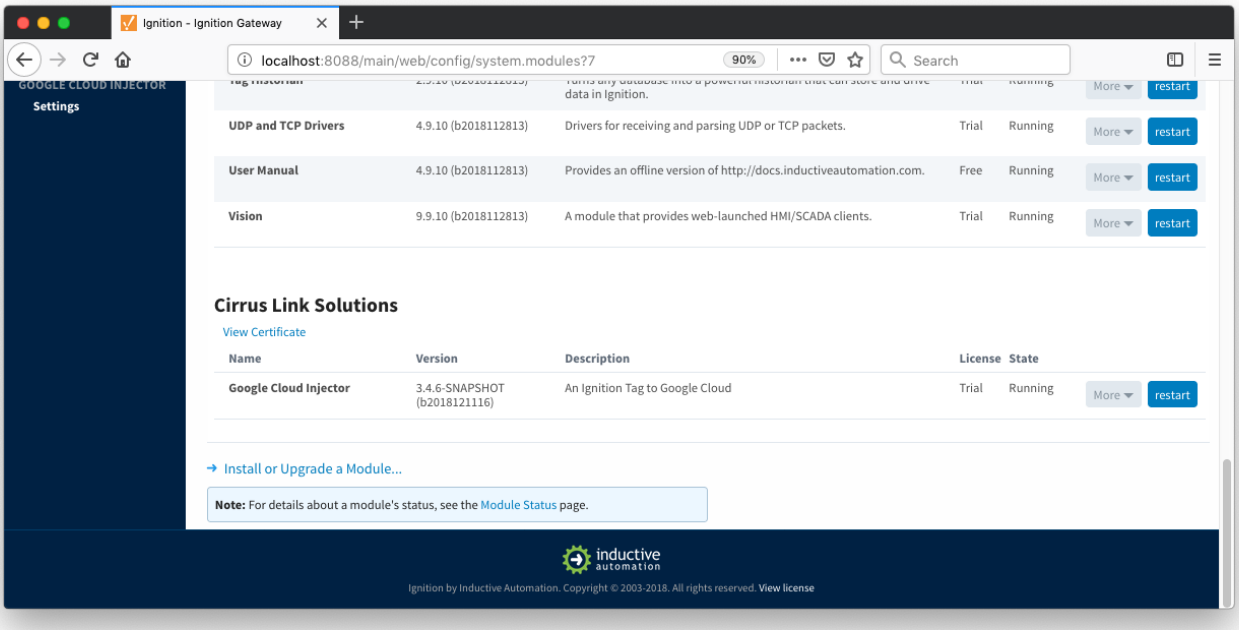
Step 2: Download and Install the Cirrus Link Google Cloud Injector Module

Go to the Inductive Automation download page again and scroll down to the Third Party modules section. Find the Cirrus Link modules section and download the Google Cloud Injector Module.
https://inductiveautomation.com/downloads/archive. The download links should look similar to what is shown below.

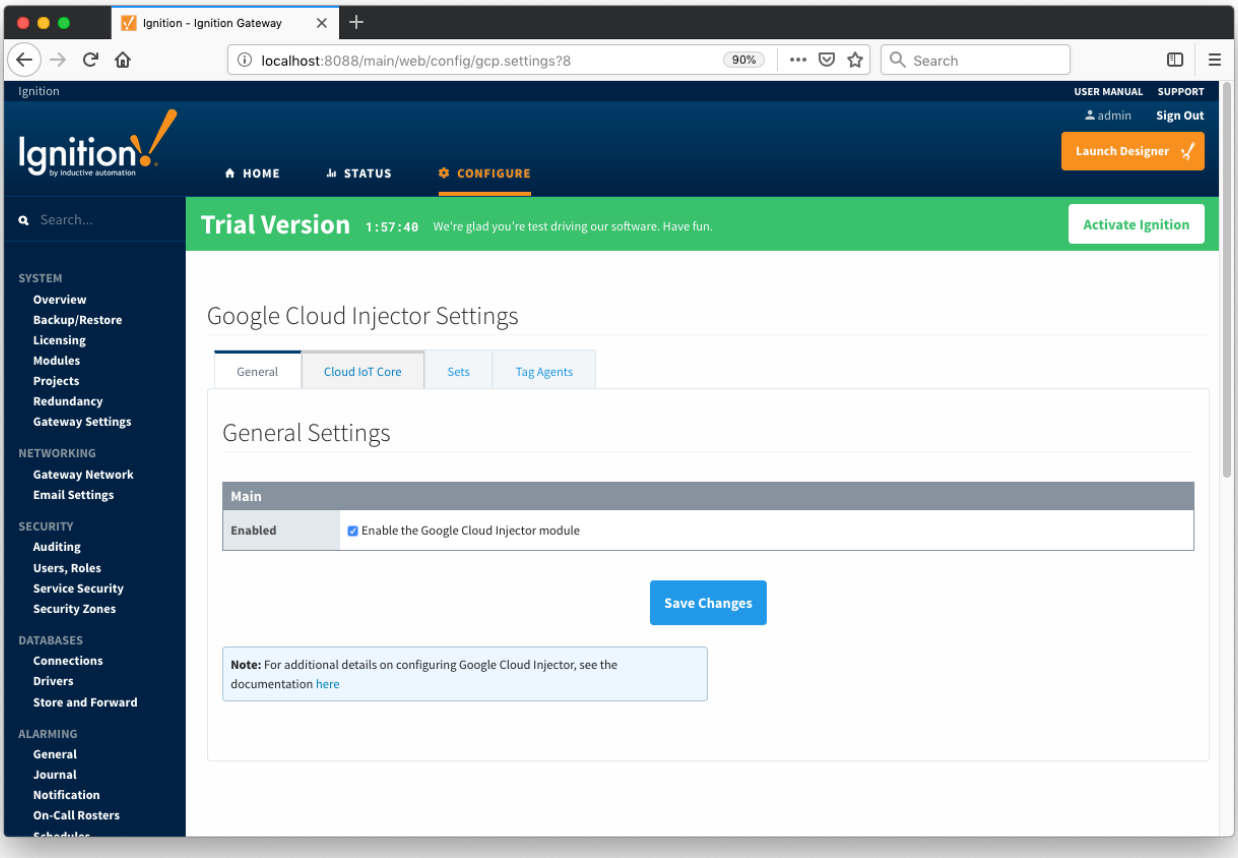


Step 3: Configure the MQTT Modules

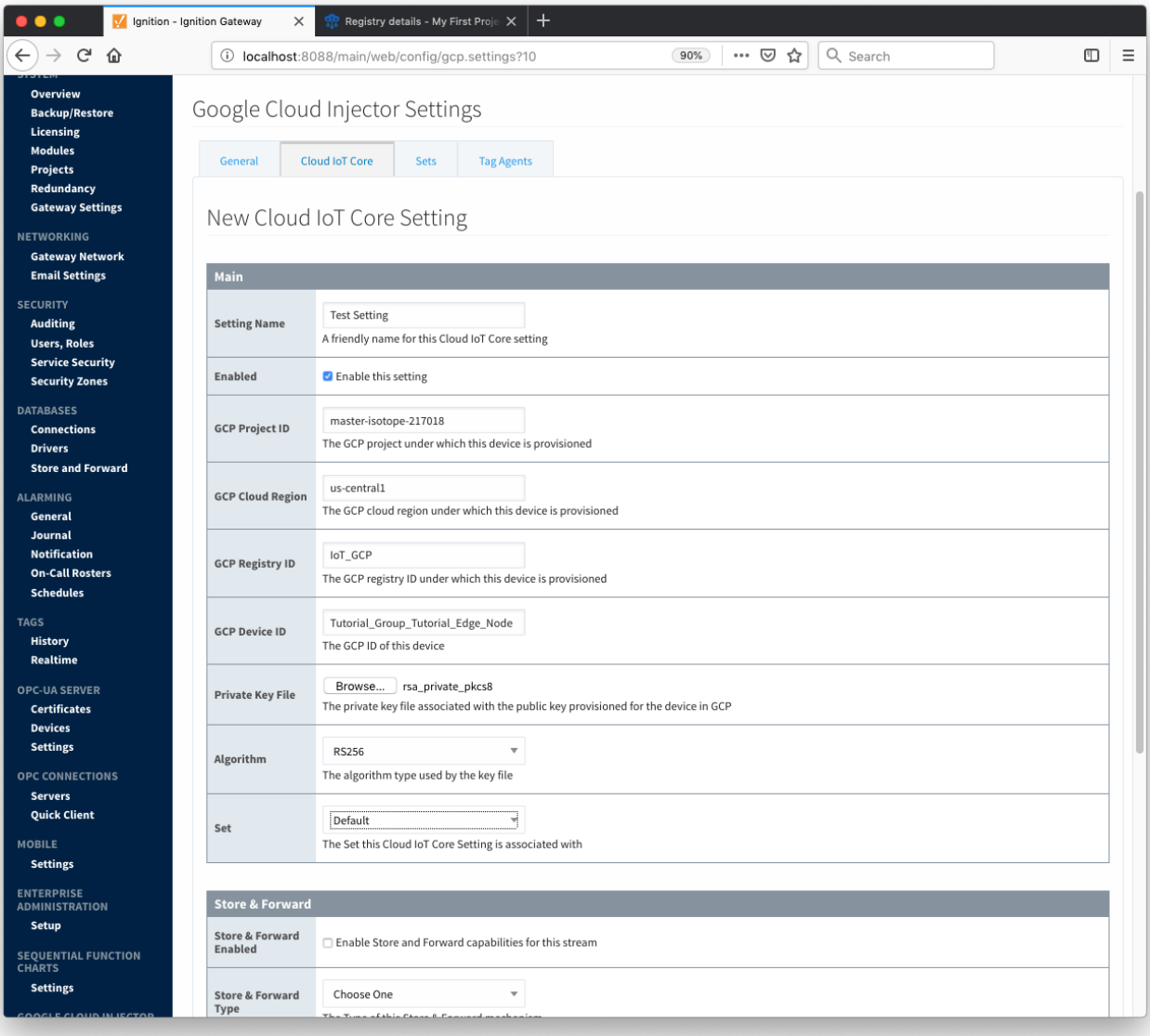
Once you have Ignition installed and running, and the Google Cloud Injector module downloaded, browse to the Ignition Gateway console (e.g. http://localhost:8088). Login using the default credentials of admin/password. Click on Configuration tab and then click on the Modules tab on the left side of the page. Scroll to the bottom of the Modules section and click on the Download/Upgrade modules button. When prompted, select the Google Cloud Injector module from the file browser and install it. When complete, the Ignition Gateway Web UI module section should look similar to what is shown below:



Select the "Google Cloud Injector" "Settings" link on the lower left of the page to navigate to the Google Cloud Injector Module's configuration page. A detailed explanation of each configuration tab can be found here. For this tutorial, we will only be adding a new Cloud IoT Core Setting.



Click on the "Create new Cloud IoT Core Setting..." link to bring up the following configuration form:



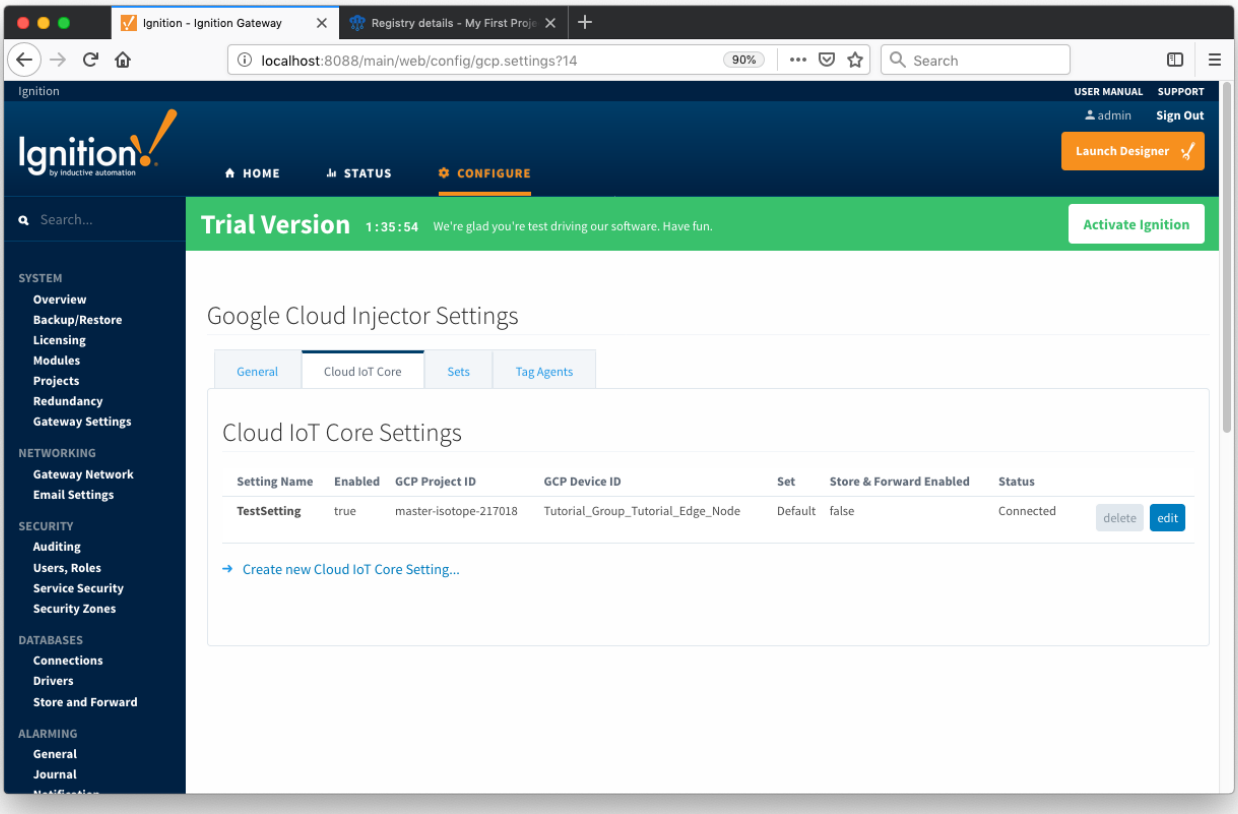
For the Setting Name you can enter any unique identifier, we will use "TestSetting".

For the GCP Project ID you will need to ID of the project that was created with the Google Cloud Platform account.

The GCP Cloud Region, Registry ID, and Device ID should all be obtained from the Cloud IoT Core that was created.

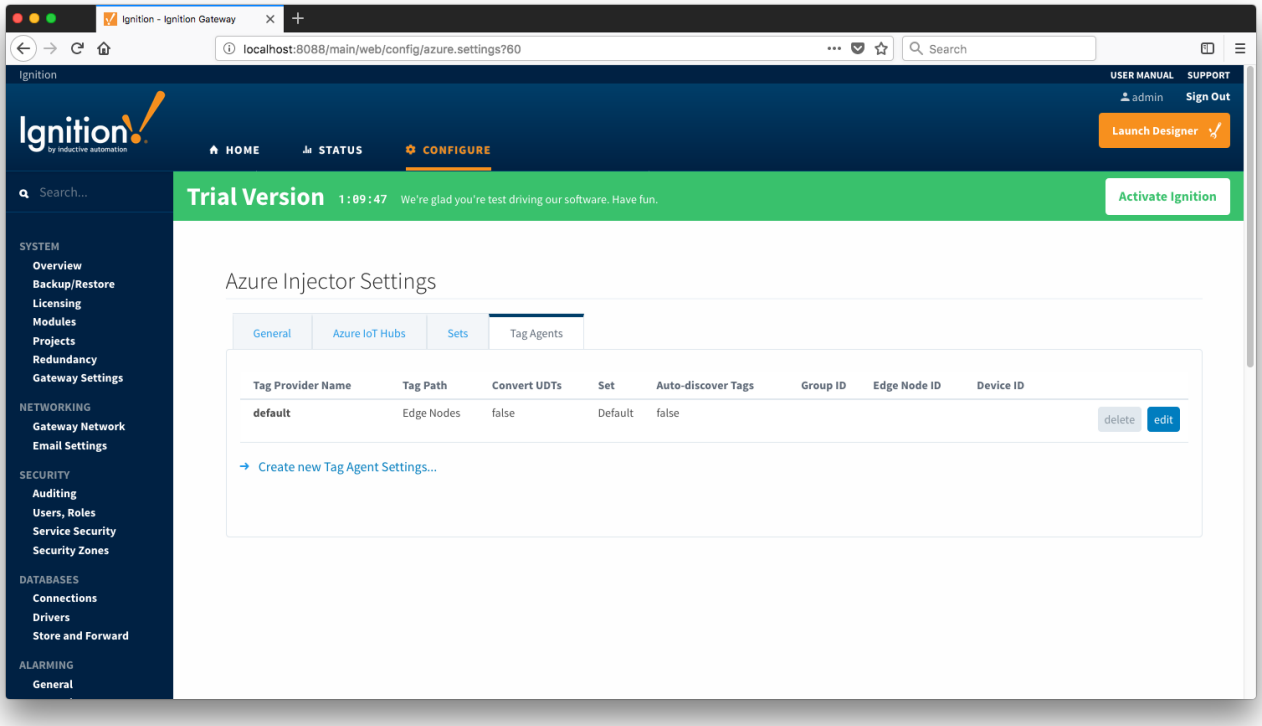
The Private Key File must be a DER encoded PKCS8 private key. NOTE: If the private key was generated using the Google Cloud IoT Core Quickstart guide (https://cloud.google.com/iot/docs/quickstart) it may need to be converted. The following command is helpful for converting from PEM to PKCS8 DER encoding:

```
openssl pkcs8 -topk8 -inform PEM -outform DER -in rsa_private.pem -out rsa_private.der -nocrypt
```



Now the Google Cloud Injector module is connected to the Cloud IoT Core and ready to push Tag data.

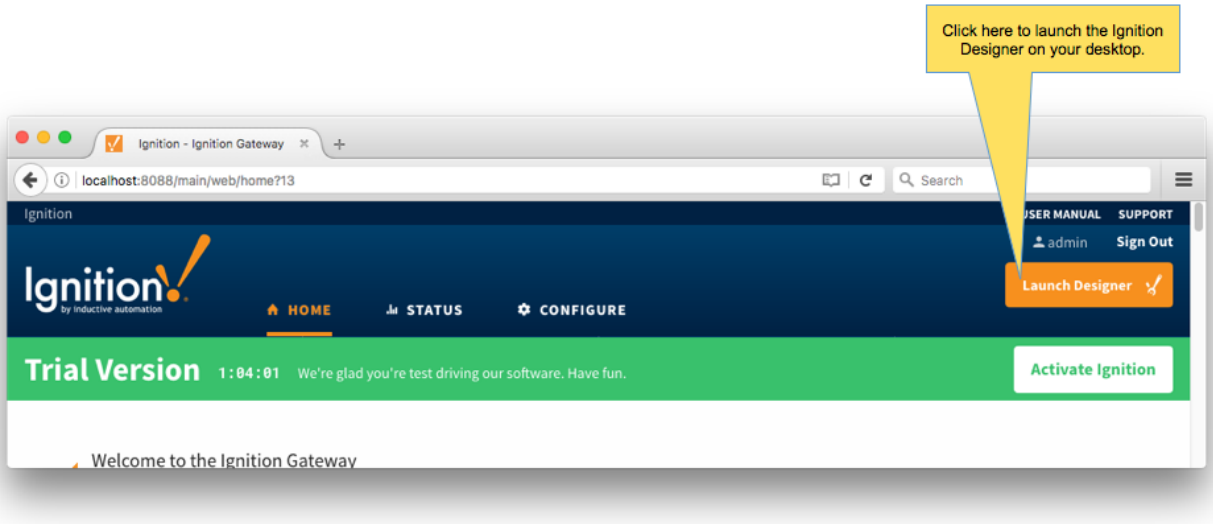
If you click on the "Tag Agents" tab you will see that out-of-the-box the Google Cloud Injector module will have one default Tag Agent defined. For this tutorial we will not need to make any configuration changes to the Tag Agents.



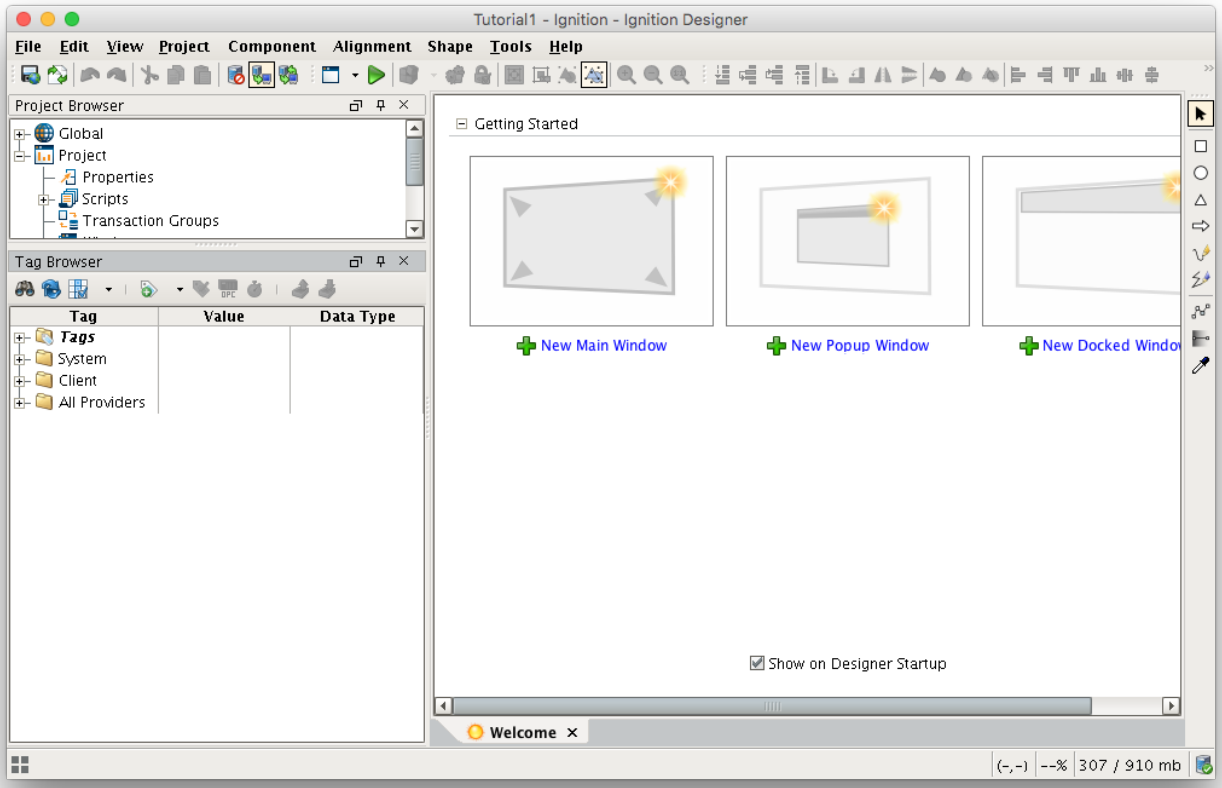
The Default Tag Agent will monitor tags that are in the "Edge Nodes" folder of the "default" Tag Provider. In the next step we go into more detail about the tags in this folder.

Step 4: Use Ignition Designer to Examine the Initial Tag Structure

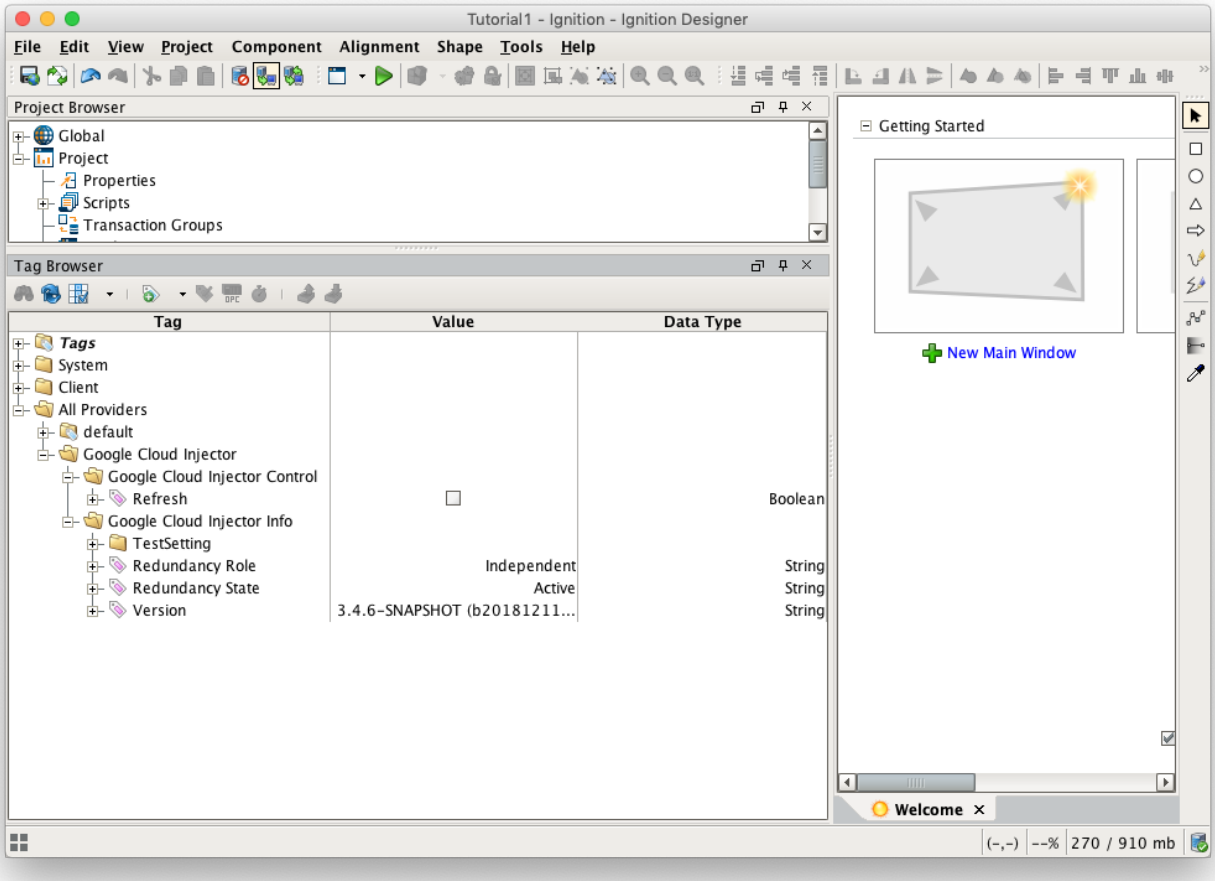
With Ignition running and the Google Cloud Injector module loaded now we can open the Ignition Designer to create/observe the initial Tag structure. Regardless of the OS Ignition is running on, there is a “Launch Designer” button on the Ignition Gateway Console. From here you can launch your Designer on any machine. This is shown below. The default credentials for the designer are the same as the Gateway Console, admin /password. Once you have logged into the Designer enter a new project name and open the project. The project name that we used for this tutorial is simply called “Tutorial1”.



After Designer opens, you will see the default Designer screen as shown below.

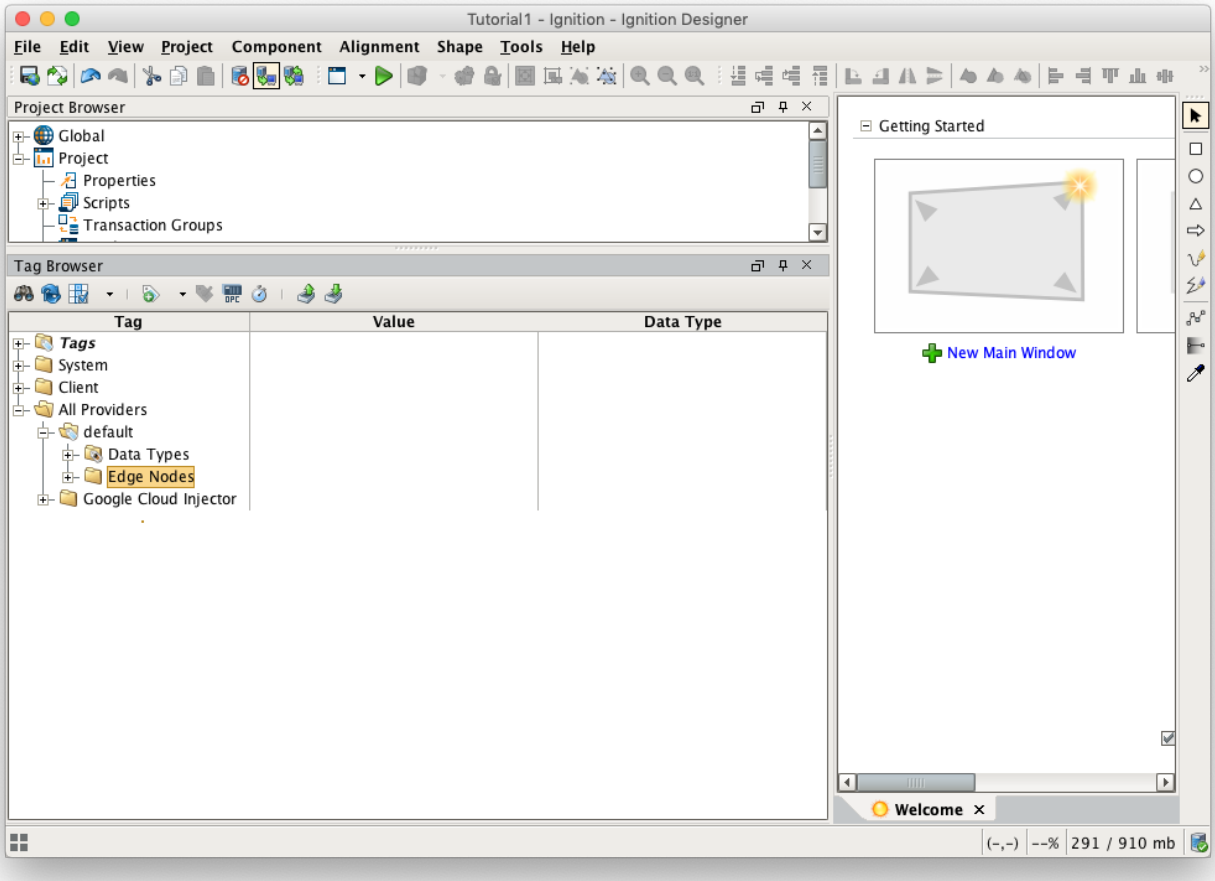


With the Google Cloud Injector module installed in Ignition, a new folder is created under the "All Providers" folder and is called “Google Cloud Injector”. This folder will contain both information tags about the module's version and state, as well as control tags for refreshing the module and Tag Agents.



Next, we need to create a folder structure where we will create a virtual Edge device and some tags to be published by the Google Cloud Injector module. When the Google Cloud Injector module is installed in Ignition, a folder is automatically created in the Ignition tag structure with the following path:

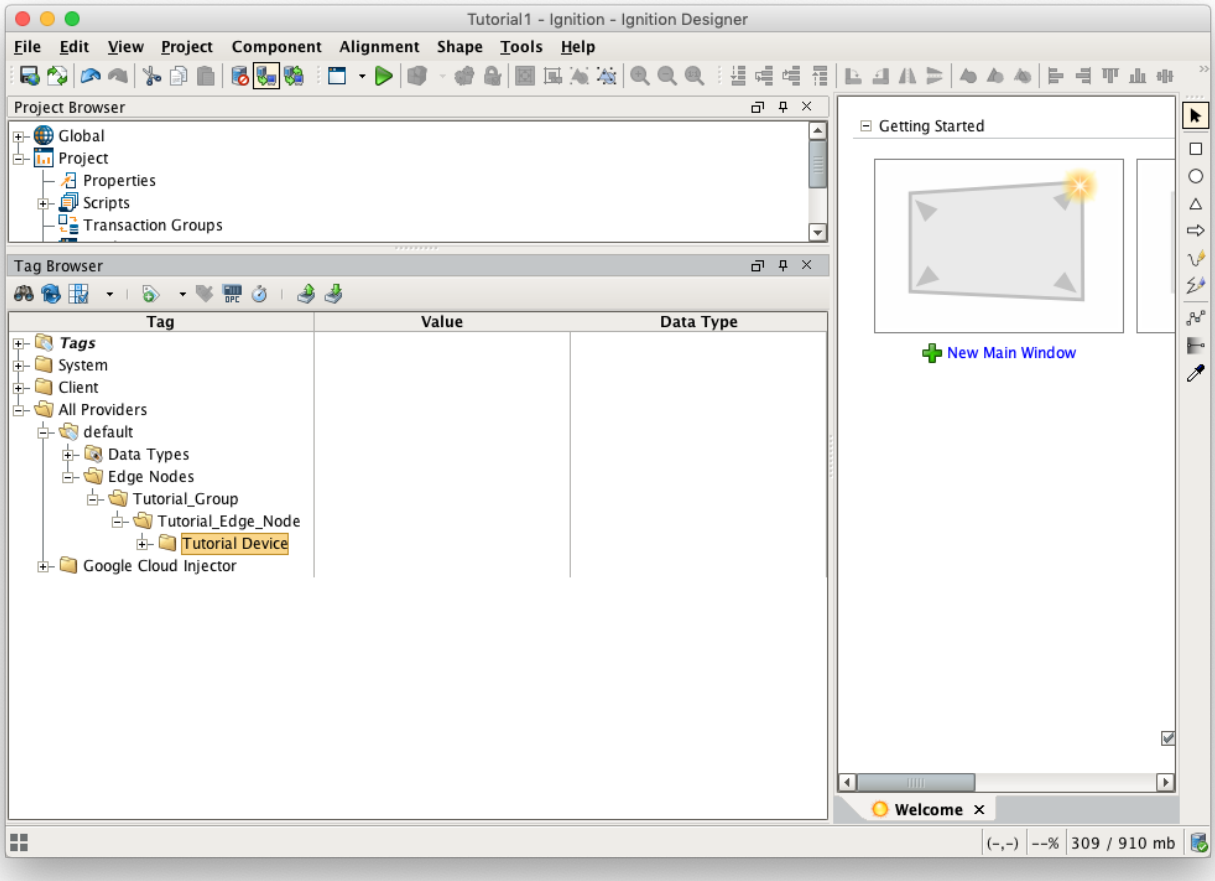
- All Providers/default/Edge Nodes



Step 5: Use Ignition Designer to Create New Tags

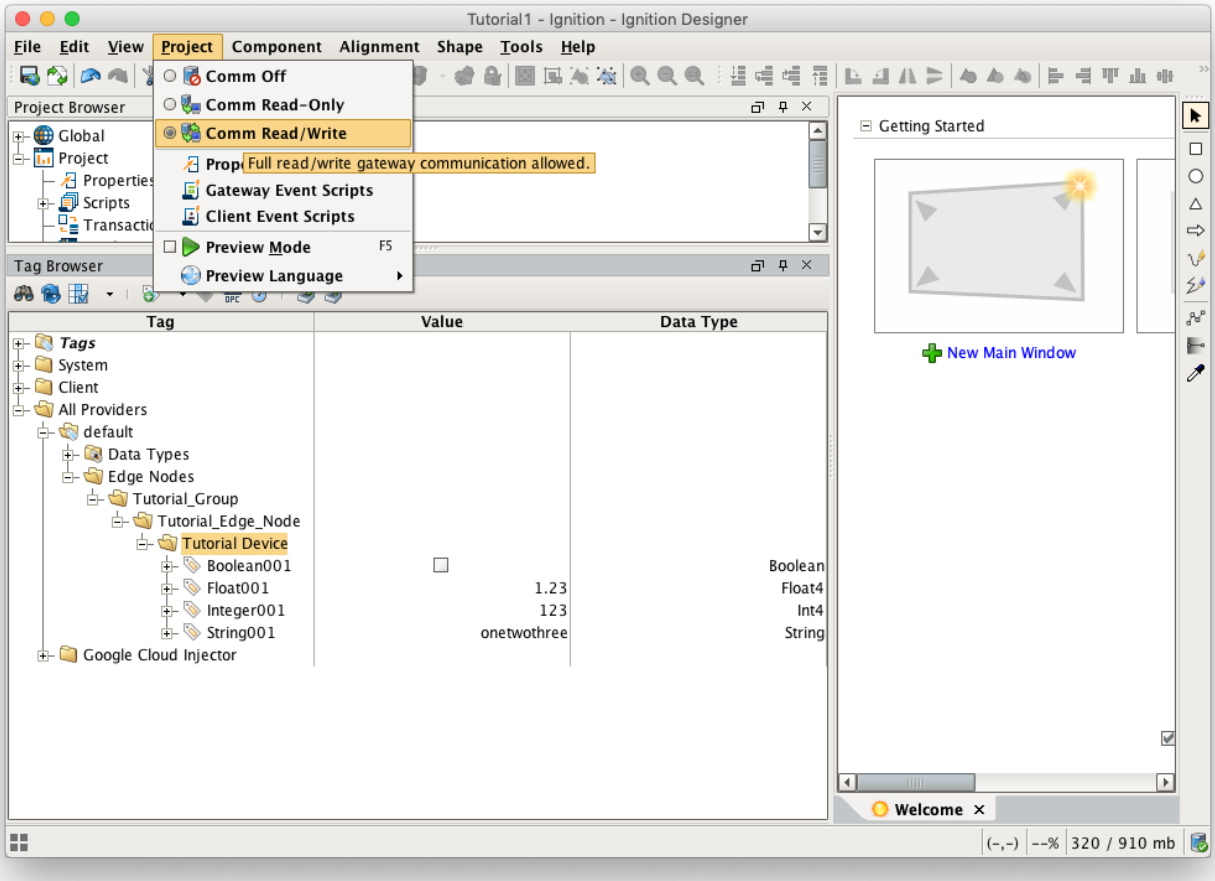
For this tutorial, right click on the Edge Nodes folder and create a new folder called Tutorial Group. Then right click on the Tutorial Group folder and create another new folder called Tutorial Edge Node. Finally, right click on the Tutorial Edge Node folder and create another new folder called Tutorial Device. This folder structure creates the same hierarchy that is described in the Sparkplug B specification of Group ID, Edge ID, and Device ID.

With this folder structure in place, now we can create some memory tags of various data types to publish. Right click on the Tutorial Device folder and select 'New Tag'/'Memory Tag'. In the tag editor change the Name of the tag to "Boolean001", and change the Data Type to Boolean. Follow this same procedure for new memory tags called "Integer001" of type Integer, "Float001" of type Float, and "String001" of type String. The resulting folder structure should look as follows.



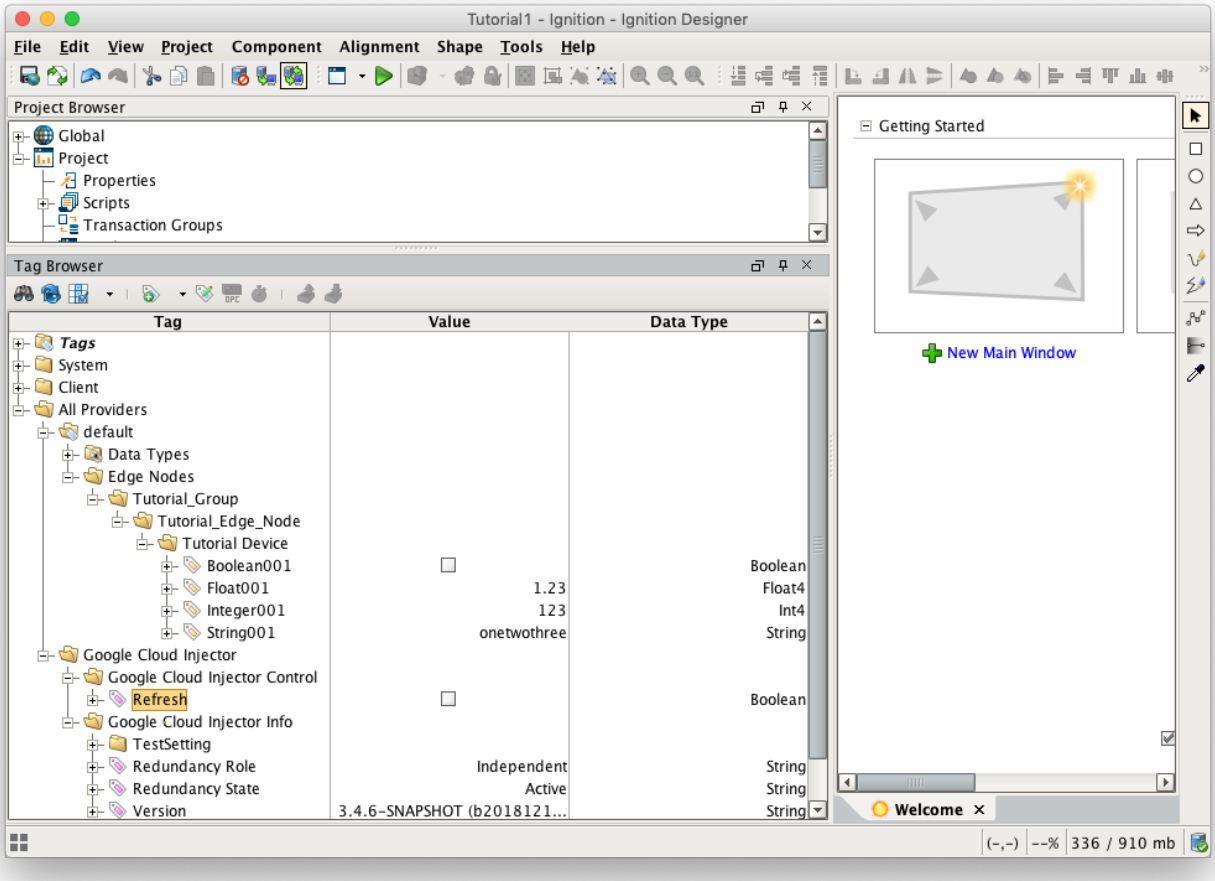
Step 6: Use Ignition Designer to Publish Tag Data (Current Tag Values)

Now that we have a folder structure with some tags we can refresh the Google Cloud Injector module. Make sure that the Ignition Designer has read/write communications turned on by selecting Project/Comm Read/Write.



To refresh the default Tag Agent with the folder structure we've created, open the folder "All Providers/Google Cloud Injector/Google Cloud Injector Control" and click on the Refresh Boolean. Note the Boolean tag will not change to true. This is really a one-shot and as a result, the tag will not change to true.

When this happens, the Tag Agent will scan the "Edge Nodes" folder and find the new Memory Tags that we have created, construct JSON payloads representing those tags with their current values and publish the payload to the Cloud IoT Core that we have configured.



The Google Cloud Injector Tag Agent will publish two JSON payloads to the Cloud IoT Core. The format of these messages closely follows the Sparkplug B Specification's payload structure.

The first payload represents the Edge Node and will contain the following:

- The Sparkplug elements: Namespace, Group ID, Edge Node ID. They will be grouped under "topic".
- A "timestamp" for when the payload was constructed.
- A "bdSeq" sequence number to track the "session" of the Tag Agent.
- Any Edge Node tags defined in the "Tutorial_Edge_Node" folder (in our example we have none).

It will look something like this:

First Payload

```
{
  "topic": {
    "namespace": "spBv1.0",
    "groupId": "Tutorial_Group",
    "edgeNodeId": "Tutorial_Edge_Node"
  },
  "payload": {
    "timestamp": 1504739061495,
    "metrics": [
      {
        "name": "bdSeq",
        "timestamp": 1504739061495,
        "dataType": "Int64",
        "value": 0
      }
    ],
    "seq": 0
  }
}
```

The second payload represents the Device and will contain the following:

- The Sparkplug elements: Namespace, Group ID, Edge Node ID, Device ID. They will be grouped under "topic".
- A "timestamp" for when the payload was constructed.
- Any Device tags defined in the "Tutorial Device" folder.

It will look something like this:

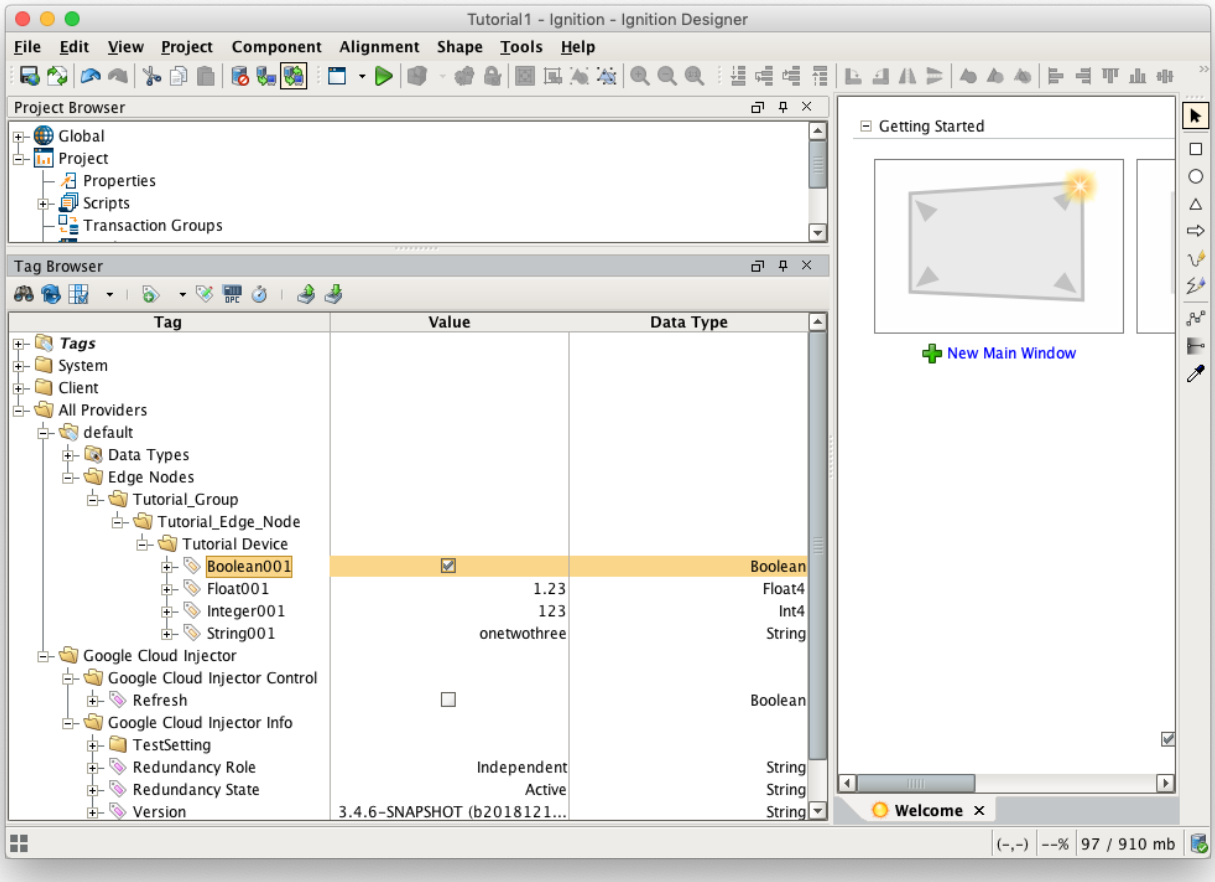
Second Payload


```
{
  "topic": {
    "namespace": "spBv1.0",
    "groupId": "Tutorial_Group",
    "edgeNodeId": "Tutorial_Edge_Node",
    "deviceId": "Tutorial Device"
  },
  "payload": {
    "timestamp": 1504739061501,
    "metrics": [
      {
        "name": "Boolean001",
        "timestamp": 1504739061546,
        "dataType": "Boolean",
        "properties": {
          "Quality": {
            "type": "Int32",
            "value": 192
          }
        },
        "value": true
      },
      {
        "name": "String001",
        "timestamp": 1504739061546,
        "dataType": "String",
        "properties": {
          "Quality": {
            "type": "Int32",
            "value": 192
          }
        },
        "value": "onetwothree"
      },
      {
        "name": "Integer001",
        "timestamp": 1504739061546,
        "dataType": "Int32",
        "properties": {
          "Quality": {
            "type": "Int32",
            "value": 192
          }
        },
        "value": 123
      },
      {
        "name": "Float001",
        "timestamp": 1504739061546,
        "dataType": "Float",
        "properties": {
          "Quality": {
            "type": "Int32",
            "value": 192
          }
        },
        "value": 1.23
      }
    ],
    "seq": 1
  }
}
```

Step 7: Use Ignition Designer to Publish Tag Data (Live Tag Values Changes)

Now we can change the values of the new Memory Tags and generate payloads that contain the Tag change events.

Click on the value of the "Boolean001" Memory Tag to change the value.



This will result in the following payload to be constructed to represent this Tag change event and pushed to the Cloud IoT Core:

Change Event Payload

```
{
  "topic": {
    "namespace": "spBv1.0",
    "groupId": "Tutorial_Group",
    "edgeNodeId": "Tutorial_Edge_Node",
    "deviceId": "Tutorial Device"
  },
  "payload": {
    "timestamp": 1504740884529,
    "metrics": [
      {
        "name": "Boolean001",
        "timestamp": 1504740883526,
        "dataType": "Boolean",
        "value": true
      }
    ],
    "seq": 2
  }
}
```

Step 8: Google Cloud Platform Applications

It is beyond the scope of this tutorial to show how to design an application in Google Cloud Platform to handle the payloads as they are pushed in to the Cloud IoT Core. For additional information on developing applications to consume this data see https://cloud.google.com/iot-core/.