

DDI Automation in Google Cloud for Private DNS

October 31, 2018

# Executive Summary

The purpose of this document is to give a high-level overview of configuring Google Cloud Ecosystem to automatically register and deregister VM Instance information into TCPWave IPAM and private DNS and to register the information in Google Datastore.

# Configuration Steps to send updates to TCPWave IPAM

## Prerequisites

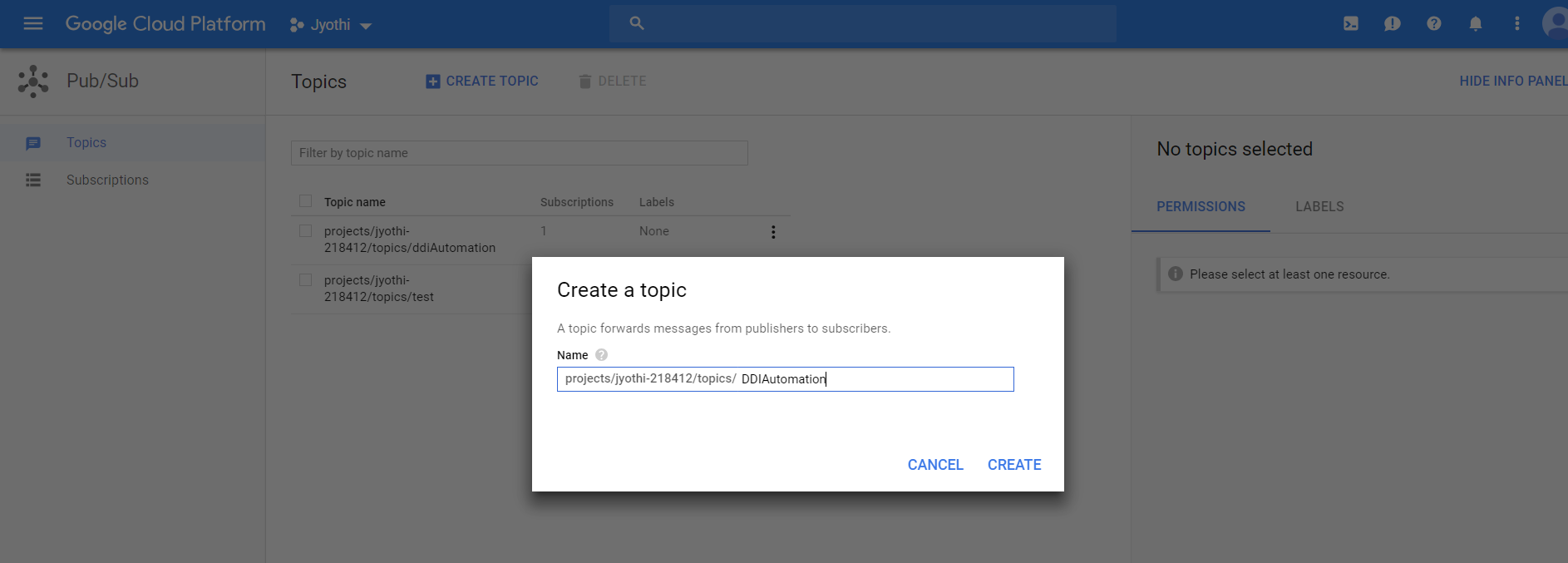
Project should be attacked to billing account.

Enable Firebase in the url: <https://console.firebase.google.com/> and add the project to it.

Enable Google Cloud Functions.

## Create a topic

Open Topics page from Big Data section in the menu and create a topic with a name.



## Create a function

Open Cloud Functions page from Compute section in the menu and create a function.

Enter Name of the function, Select Google Pub/Sub as Trigger, select the Topic that’s created in the above step.

Select ZIP upload option for the Source Code and upload UpdateIPAM.zip.

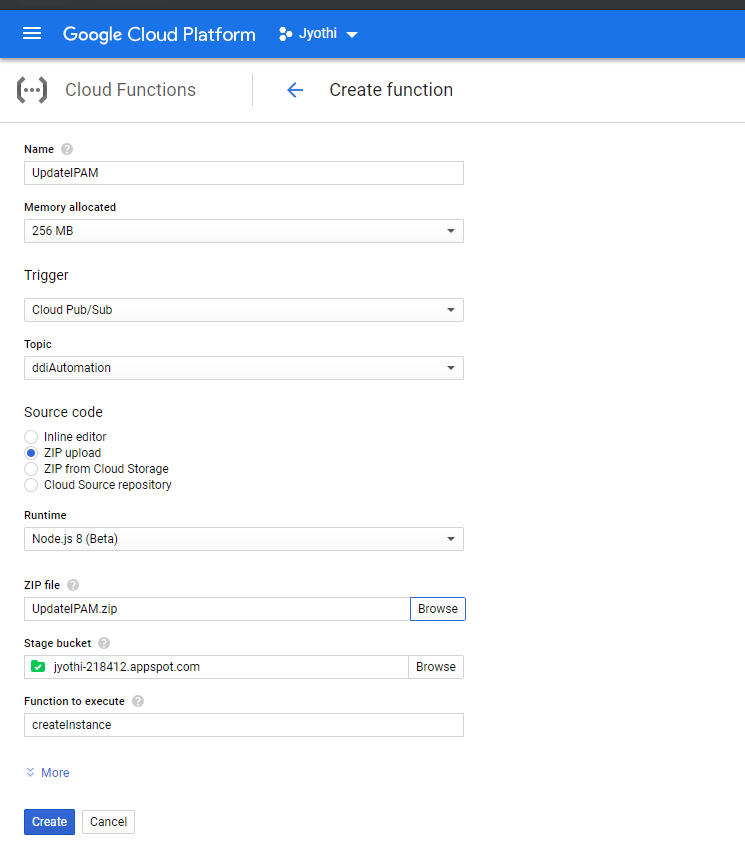
Note: Please unzip the file, update client.pem, client.crt and client.key with the certificates created for the IPAM , zip the file again.

Select Node.js 8(Beta) as the Runtime.

Select first option in the Stage bucket dropdown.

Give createInstance as the function to execute.

Click on Create.



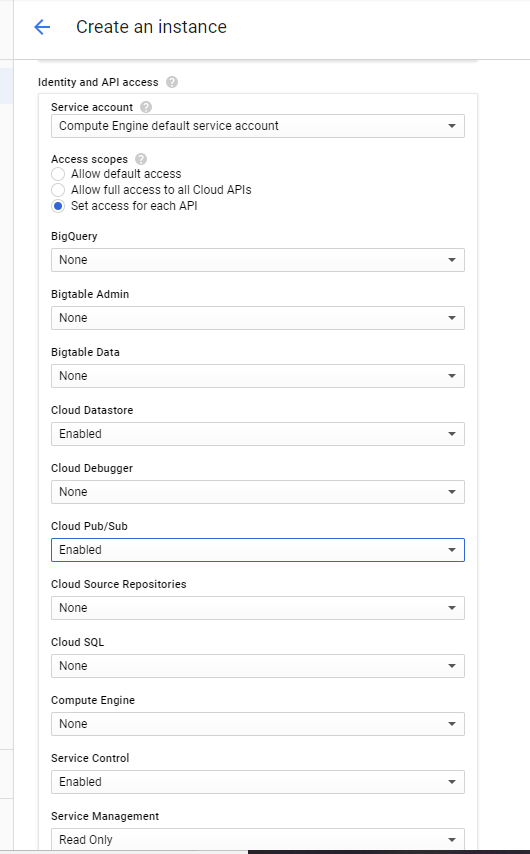
## Create Instance using Startup/Shutdown scripts

Go to Compute Section of the menu and open VM Instances page from Compute Engine.

Click on Create Instance

After entering the Machine type, region etc, select **Set Access for each API** as option for Access Scopes.

And select Cloud DataStore and Cloud Pub/Sub and enable them.



Expand **Management, security, disks, networking, sole tenancy** section and enter the below script in the Startup script text box in Automation section. Give the name of the topic that is created in the gcloud publish command.

|  |
| --- |
| #! /bin/bash  orgName='Internal'  domainName='tcpwave.com'  ttl='1200'  action='add'  instanceid=$(curl http://metadata/computeMetadata/v1/instance/id -H "Metadata-Flavor: Google")  hname=$(curl http://metadata/computeMetadata/v1/instance/name -H "Metadata-Flavor: Google")  iip=$(curl http://metadata/computeMetadata/v1/instance/network-interfaces/0/ip -H "Metadata-Flavor: Google")  projId=$(curl "http://metadata.google.internal/computeMetadata/v1/project/project-id" -H "Metadata-Flavor: Google")  gcloud pubsub topics publish DDIAutomation --attribute id=$instanceid,name=$hname,ip=$iip,orgName=$orgName,domainName=$domainName,ttl=$ttl,action=$action,projectId=$projId |

In the Metadata section, add key value pair, shutdown-script as the key and below script as the value. Give the name of the topic that is created in the gcloud publish command.

|  |
| --- |
| #! /bin/bash  orgName='Internal'  domainName='tcpwave.com'  ttl='1200'  action=’delete’  instanceid=$(curl http://metadata/computeMetadata/v1/instance/id -H "Metadata-Flavor: Google")  hname=$(curl http://metadata/computeMetadata/v1/instance/name -H "Metadata-Flavor: Google")  iip=$(curl http://metadata/computeMetadata/v1/instance/network-interfaces/0/ip -H "Metadata-Flavor: Google")  projId=$(curl "http://metadata.google.internal/computeMetadata/v1/project/project-id" -H "Metadata-Flavor: Google")  gcloud pubsub topics publish DDIAutomamtion --attribute id=$instanceid,name=$hname,ip=$iip,orgName=$orgName,domainName=$domainName,ttl=$ttl,action=$action,projectId=$projId |

Create instance with the above settings.

With the above successful settings, an object in IPAM will be created when instance is started and object will be deleted when the instance is stopped.

# Configuration Steps to send updates to Google Datastore

## Prerequisites

Project should be attached to billing account.

Enable Google Cloud Functions.

## Create Entity

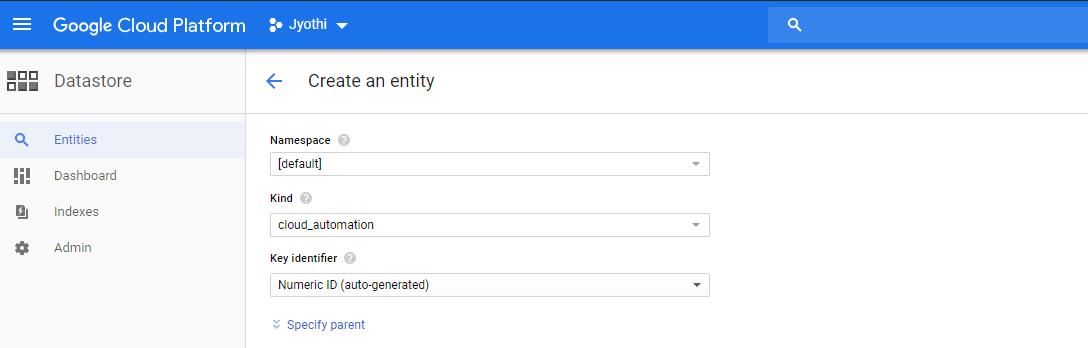
Open Entities page from Datastore menu of Storage section from main menu.

Click on Create Entity.

Enter cloud\_automation in the Kind input box.

Select Numeric ID as the Key identifier.

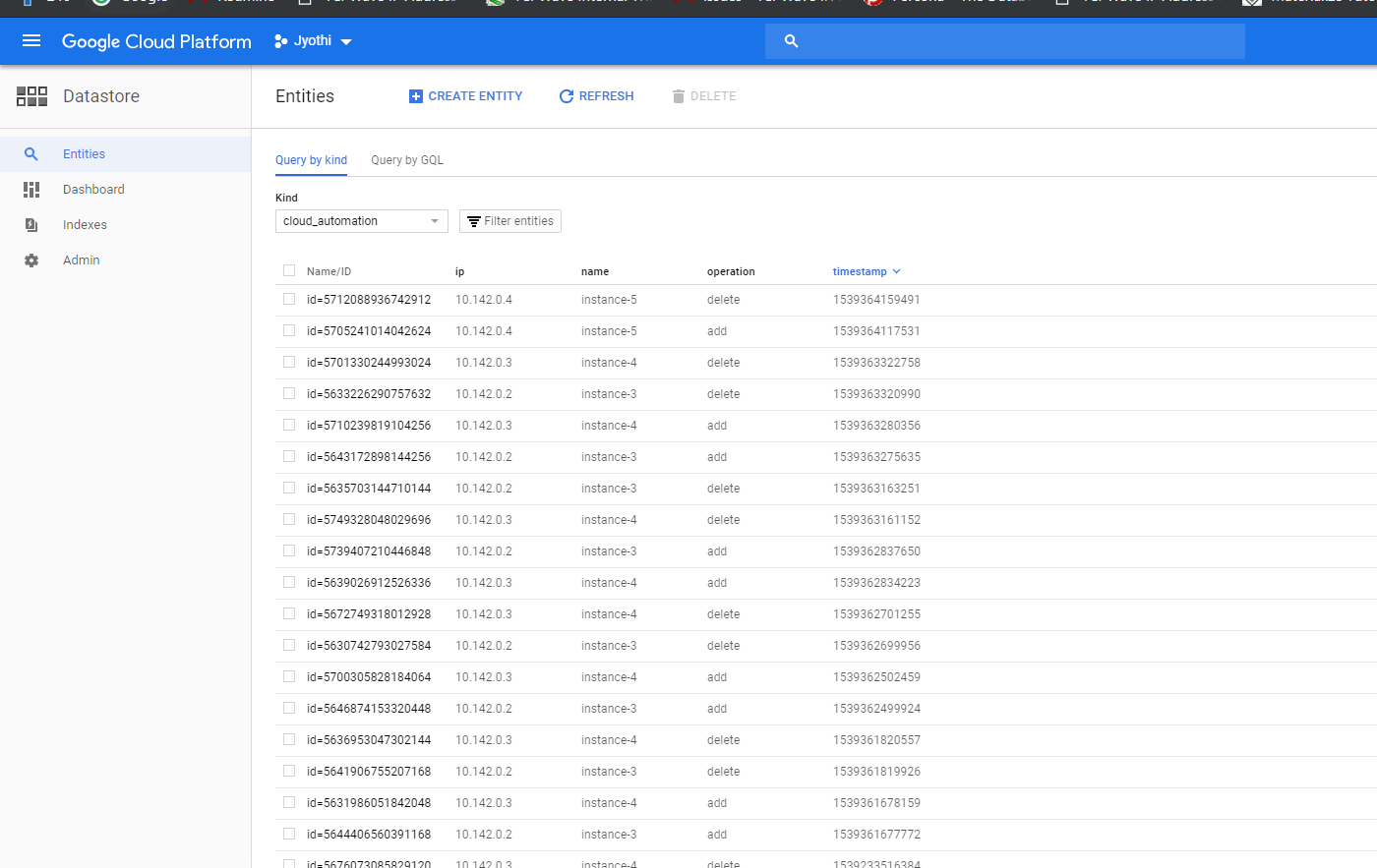
Click on Create.



Repeat the steps that are present in the section: Configuration Steps to send updates to TCPWave IPAM with UpdateDatastore.zip as the function to be uploaded. Function to execute is updateDataStore.

As the zip file doesn’t contain certificate files, there is no need to update them.

After the completion of above steps, if the instance is started/stopped, the row with instance details will be added in the cloud\_automation table.



# Configuration Steps to update QIP

## Prerequisites

1. updateDataStore function must be created and added as a trigger to the Cloud Pub/Sub topic. This topic must be mentioned in the start-up and shutdown scripts while creating instances.
2. Node js must be installed.

## Steps

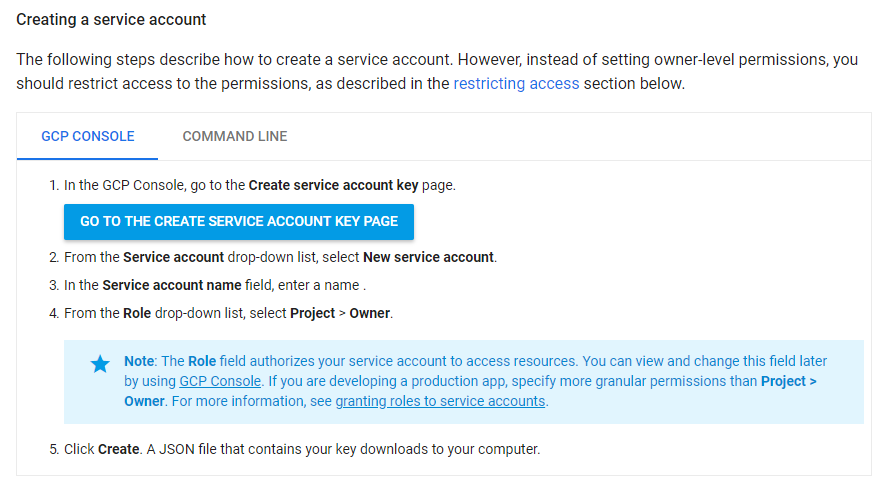
### Install google-cloud/datastore packages using below command

|  |
| --- |
| npm install --save @google-cloud/datastore |

### Create Google cloud service account

Follow the steps provided in the below screenshot from the documentation and download the JSON to the current folder and give this path as keyFileName in the node js function created in the next step.

*https://cloud.google.com/docs/authentication/production#obtaining\_and\_providing\_service\_account\_credentials\_manually*



### Create a node.js function

Create a function with below code and give a name to it.

This function gets data from Google Datastore table ‘cloud\_automation’ and processes the data and delete the processed rows from cloud\_automation table.

In a row, if the value of operation is add, the function adds objects in QIP. If the operation is delete, it deleted object from QIP.

A CRON job can be created to execute this function at a regular interval of time.

**Note:** In the function, modify the HTTPS\_PROXY, domain, projectId and keyFileName before executing it.

Command to execute the function is below

|  |
| --- |
| node test.js(File name) |

File content is below

|  |
| --- |
| const Datastore = require('@google-cloud/datastore');  const exec = require('child\_process').execSync;  const moment = require('moment');  exec('. /opt/qip/etc/shrc');  exec('export HTTPS\_PROXY=https://cspapiproxy-lab.nam.nsroot.net:80');  exec('touch /tmp/process-add');  exec('/bin/chmod +x /tmp/process-add');  exec('touch /tmp/process-delete');  exec('/bin/chmod +x /tmp/process-delete');  var domain= 'aws.namdev.nsrootdev.net';  exec('echo "ObjectAddress,ObjectName,DomainName,ObjectClass,ObjectDescription,Aliases,NameService,DynamicDNSUpdate" > /var/tmp/format');  const datastore = new Datastore({  projectId: 'jyothi-218412',  keyFilename: 'Jyothi-630530137299.json'  });  const query = datastore.createQuery('cloud\_automation');  datastore.runQuery(query, function(err, entities) {  // entities = An array of records.  // Access the Key object for an entity.  const firstEntityKey = entities[0][datastore.KEY];  console.log("Number of rows fetched from Google datastore: "+entities.length);  for(var i in entities)  {  var e = entities[i];  var ip = e.ip;  var ts = e.timestamp;  var op = e.operation;  var n = e.name;  var key = e[datastore.KEY]  var id = key.id;  var d = new Date(Number(ts));  if(op == 'add')  {  var v = ip+','+n+','+domain+',server,,,\'A,PTR\',\'A,PTR,CNAME,MX\'';  var cmd = 'echo \"'+v + '\" > /tmp/process-add';  exec(cmd);  console.log('Adding object: '+ip+' '+n+' '+d);  const addObj = exec('qip-setobject -f /var/tmp/format -d /tmp/process-add');  addObj.stdout.on('data', function(data){  console.log('Added object: '+ip+' '+n+' '+d);  if(data == '0')  {  datastore.delete(key, function(err) {  if (!err) {  console.log('Record deleted successfully. Deleted record id: '+key.id);  }  });  }  });  addObj.stderr.on('data', function(data){  console.log('Failed to add object: '+ip+' '+n+' '+d+" "+data);  });  }  else if(op == 'delete')  {  var v = 'qip-del -t object -a '+ip;  var cmd = 'echo \"'+v + '\" > /tmp/process-delete';  exec(cmd);  console.log('Deleting object: '+ip+' '+n+' '+d);    const deleteObj = exec('/tmp/process-delete');  deleteObj.stdout.on('data', function(data){  console.log('Deleted object: '+ip+' '+n+' '+d);  if(data == '0')  {  datastore.delete(key, function(err) {  if (!err) {  console.log('Record deleted successfully. Deleted record id: '+key.id);  }  });  }  });  deleteObj.stderr.on('data', function(data){  console.log('Failed to delete object: '+ip+' '+n+' '+d+" "+data);  });  }    }  }); |