CTA LAB 2018

Collaborative 5G Edge API hands-on session

Dallas, TX 2018

Table of Contents

1. Introduction

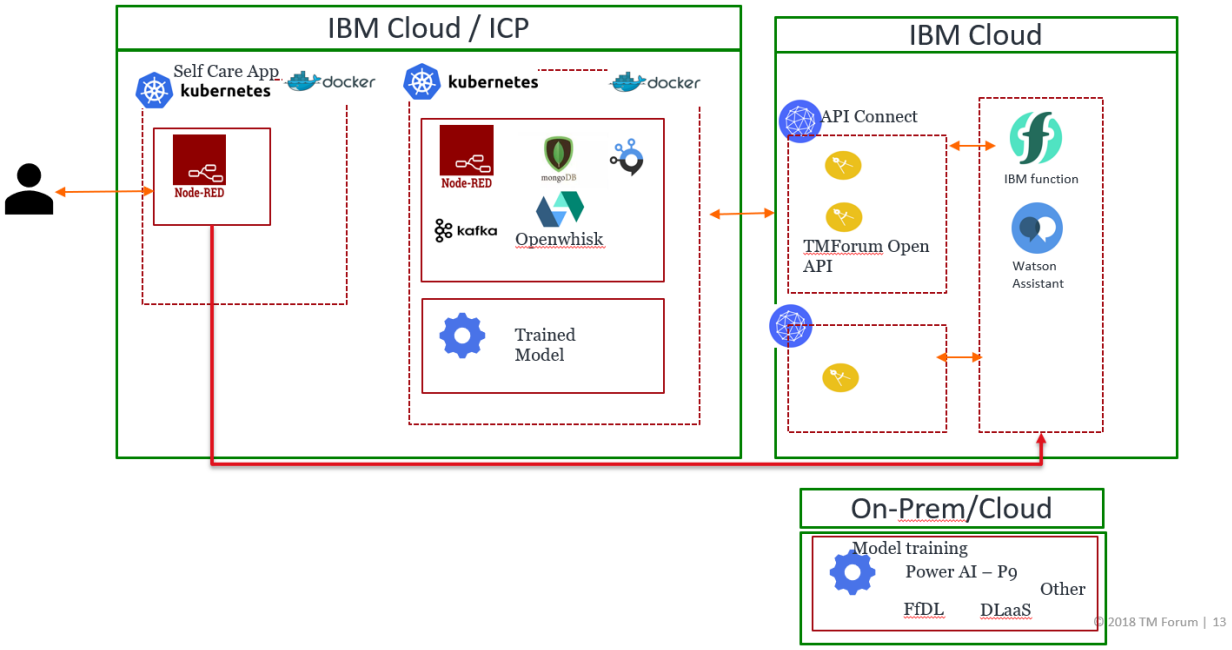
The Open Digital Lab provides a safe space for inter-company collaborative proof of concept generation leveraging open standards. With the pace of change in the industry continuing to accelerate, and partnerships with other companies to deliver new services and solutions increasing in importance, there has never been a greater need for an environment for collaborative co-creation of innovative new solutions and services. The TM Forum Open Digital lab hosted by IBM provides this environment where inter-company proof of concepts can be rapidly developed, tested and iterated.

The Open Digital Lab is a member benefit for the 900-member companies who are joined TM Forum. To access the lab, first sign up for a free IBM Cloud account and connect to the TM Forum space where a feast of tools available for rapid innovation. These include but are not limited to:

- Live reference implementations of TM Forum's Open APIs
- Easier access to rapid testing of in development or deployed TMF Open APIs
- Access a series of open source technologies
- In a safe place for co-creation and iteration

The first wave of innovative projects to use the Open Digital lab are the proof of concept catalyst projects being developed for Digital Transformation World 2018 & 2019.

2. Architecture



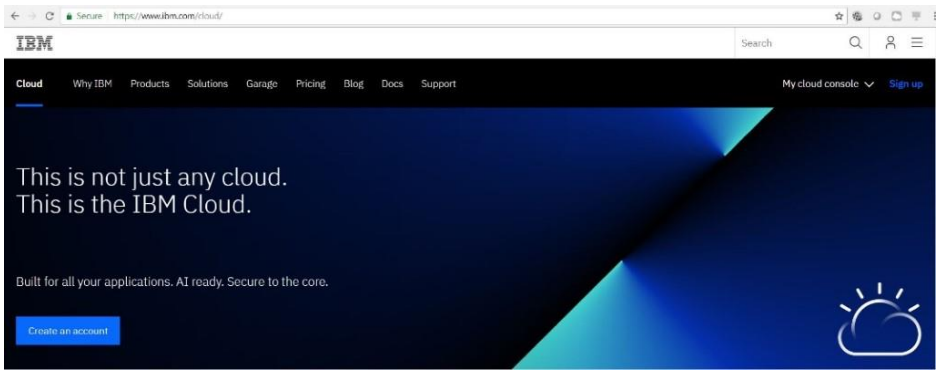
Github repository link for the lab: https://github.com/commgsc/CTA_LAB_2018

3. Environment Setup

3.1. Create an IBM Cloud Account (Skip if you have one)

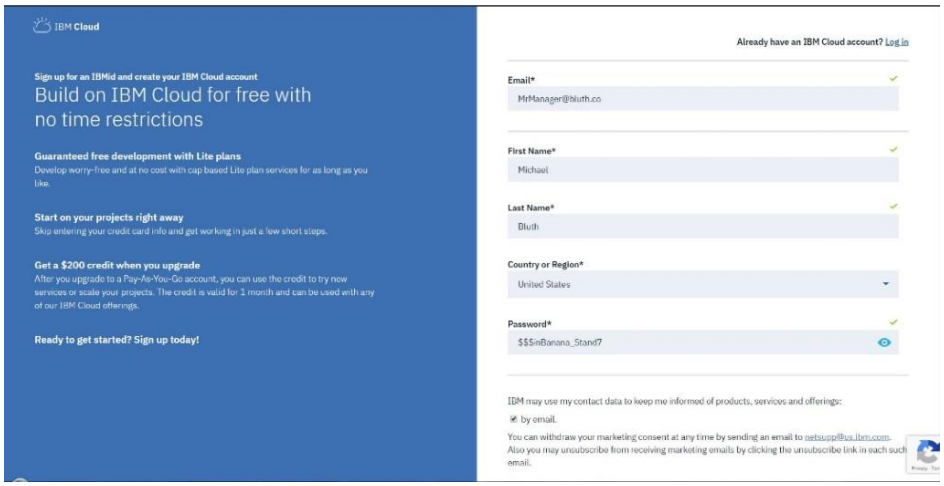
- Step (1)

Navigate to [www.ibm.com/cloud, Click the Blue “Create Account” Button]

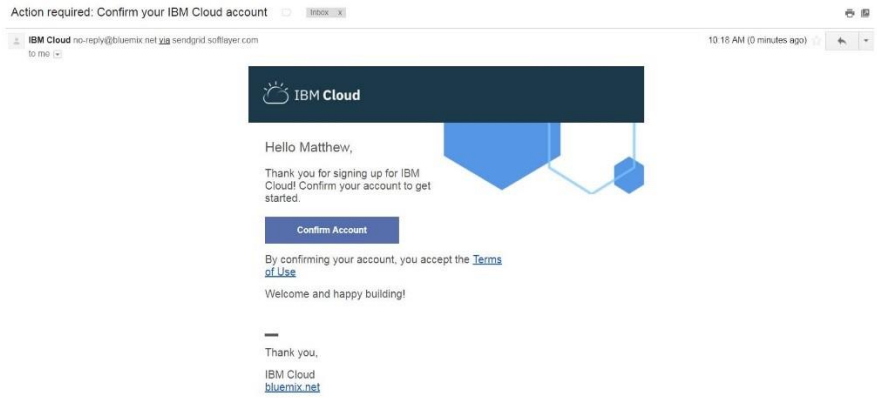


- Step (2)

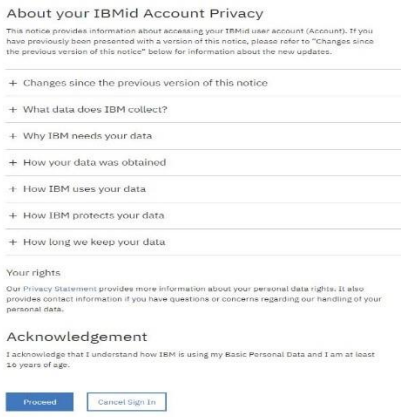
Create your account details and password



You will receive an Account Confirmation Email at the address you provided. Click the blue “Confirm Account” button within the email.



You will immediately be provided with a copy of the IBMid Privacy Policy. Read the policy, then click “Proceed”



Write down your organization information:

Table 1:

Organization:	
Space:	

3.2. Create a Kubernetes Instance on IBM Cloud

To create a free cluster:

1. Login to IBM Cloud using https://bluemix.net/, from the IBM Cloud Catalog select **Containers** category and click **Containers in Kubernetes Clusters**.
2. Read up on clusters, then click **Create**. Select Location as **US South** and enter a **Cluster Name**. The default cluster type is free. Next time, you can create a standard

cluster and define additional customizations, like the number of worker nodes. The cluster will be deleted after 30 days on the lite account.

3. Click **Create Cluster**. The details for the cluster open, but the worker node in the cluster takes about 30 minutes to provision. You can see the status of the worker node in the **Worker nodes** tab. When the status reaches Ready, your worker node is ready to be used.

References:
https://console.bluemix.net/docs/containers/container_index.html#container_index

3.3. Install IBM Bluemix CLI

IBM Cloud CLI provides the command line interface to manage applications, containers, infrastructures, services and other resources in IBM Cloud. Installing IBM Cloud CLI using below commands should install kubectl, docker, git and IBM Cloud plug-ins.

- For Mac and Linux, open a terminal and run the following command:

```
curl -sL https://ibm.biz/idt-installer | bash
```

- For Windows 10, run the following command as an administrator (Right-click the Windows PowerShell and select **Run as administrator**)

```
Set-ExecutionPolicy Unrestricted; iex(New-Object Net.WebClient).DownloadString('http://ibm.biz/idt-win-installer')
```

Did not work? Try copying the command using the button from the following link as shown below:

https://console-dal10.bluemix.net/docs/cli/index.html#overview

Step 1. Run the install command

- For Mac and Linux, run the following command:

```
curl -sL https://ibm.biz/idt-installer | bash
```

- For Windows 10 Pro, run the following command as an administrator:

```
Set-ExecutionPolicy Unrestricted; iex(New-Object Net.WebClient).DownloadStrin
```

Tip: Right-click the Windows PowerShell icon, and select **Run as administrator**.

3.4. Install Node.js

Install nodejs from https://nodejs.org/en/download/ which also installs npm.
(Please install version 8.12.0 (recommended))
You can verify the versions of the required or pre-installed software by running the following commands and ensuring that you have the following versions (or higher).

3.5. Install API Connect CLI

Install IBM API Connect v5 developer toolkit: IBM API Connect is an end-to-end API management solution that uses Loopback to create APIs and provides integrated build and deployment tools. Install IBM API Connect v5 developer toolkit using "**npm install -g apiconnect**" on windows and "**sudo npm install -g apiconnect**" on Mac or Linux.

You can verify the versions of the required or pre-installed software by running the following commands and ensuring that you have the following versions (or higher).

```
C:\Users\SaiGorti>apic --version
API Connect: v5.0.8.4-iFix (apiconnect: v2.8.39)
```

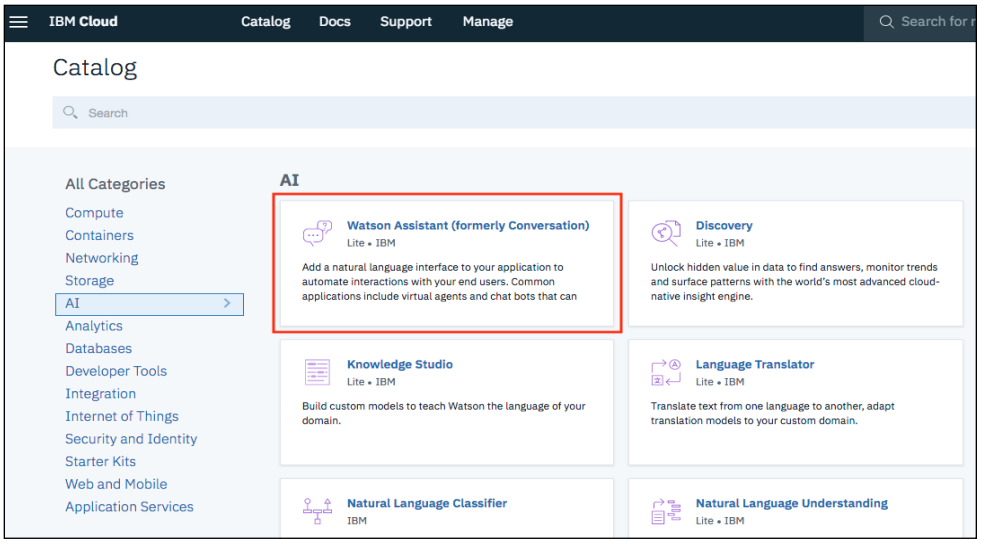
3.6. Install Kubernetes CLI

kubectl is the Kubernetes command-line tool to deploy and manage applications on Kubernetes. Using kubectl, you can inspect cluster resources; create, delete, and update components; and look at your new cluster and bring up example apps. Install kubectl using the link https://kubernetes.io/docs/tasks/tools/install-kubectl/ (only if installing IBM Cloud CLI in section 3.3 has not installed kubectl).

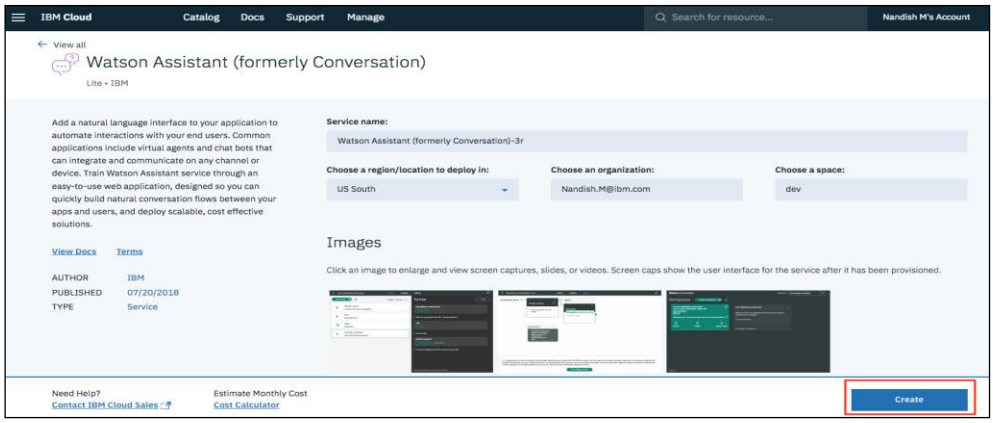
To verify installation of kubectl, open command line or terminal and run "**kubectl --help**". If it returns list of commands, then kubectl is installed already. If not got to the above link and install kubectl.

3.7. Create a Watson Assistance service

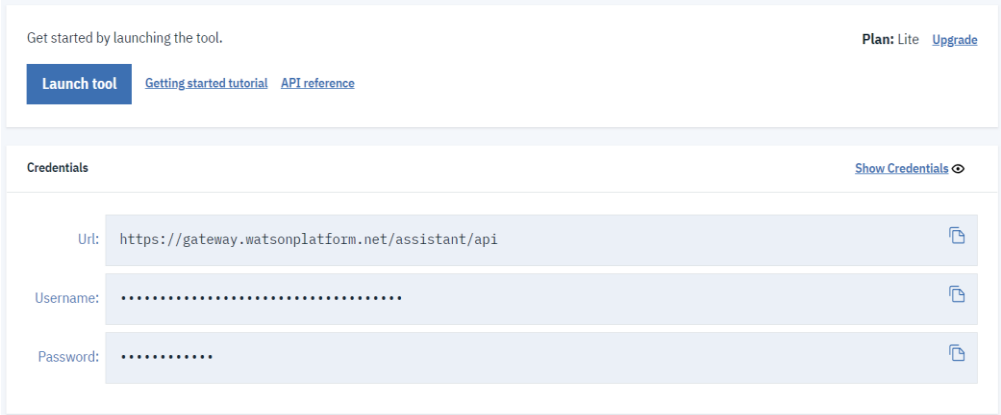
- Create Watson Assistant service
  Navigate to 'Catalog' section in IBM Cloud.
  Select 'Watson Assistant' in 'AI' Category (as shown in image below).



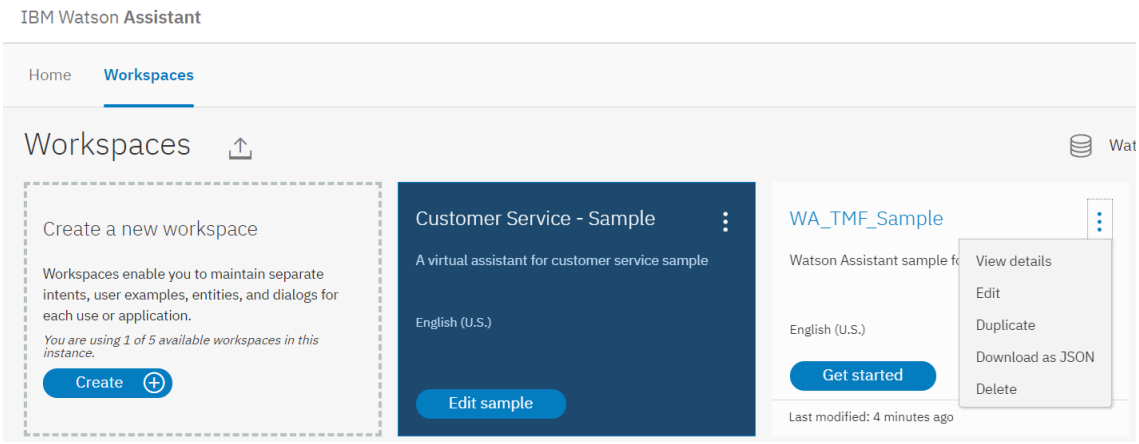
Click ‘Create’



- <u>Create a workspace:</u>
  Once Watson assistant service is created, you will be able to see this screen.



Click “Launch tool” and you will be able to access the workspaces. Click “View details” of your workspace to see the “Workspace_ID”



- Get the Service Credentials:
  Please make a note of username and password of the Watson assistant service and workspace id of the workspace created. Copy the username and password and make it readily accessible for further operations.

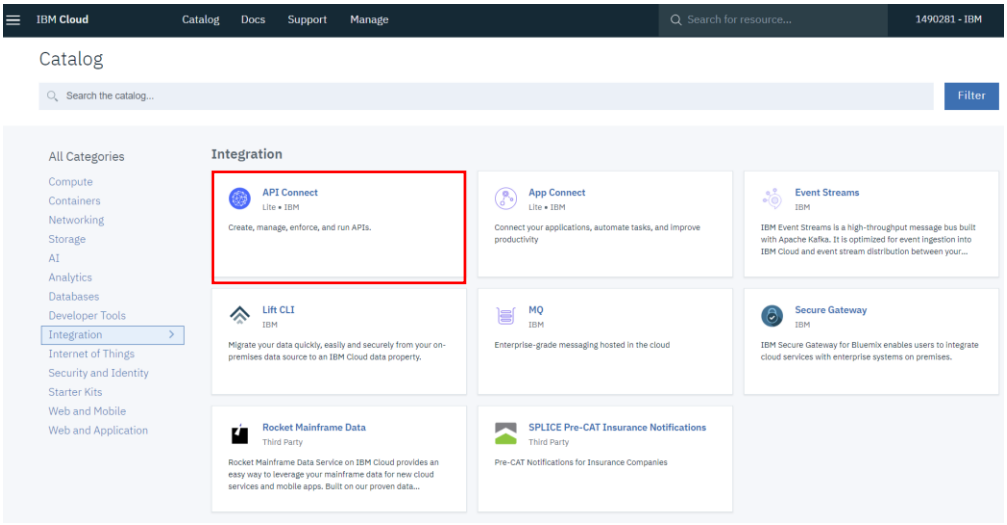
Table 2:

Username:	
Password:	
Workspace ID:	

3.8. Create APIC instance

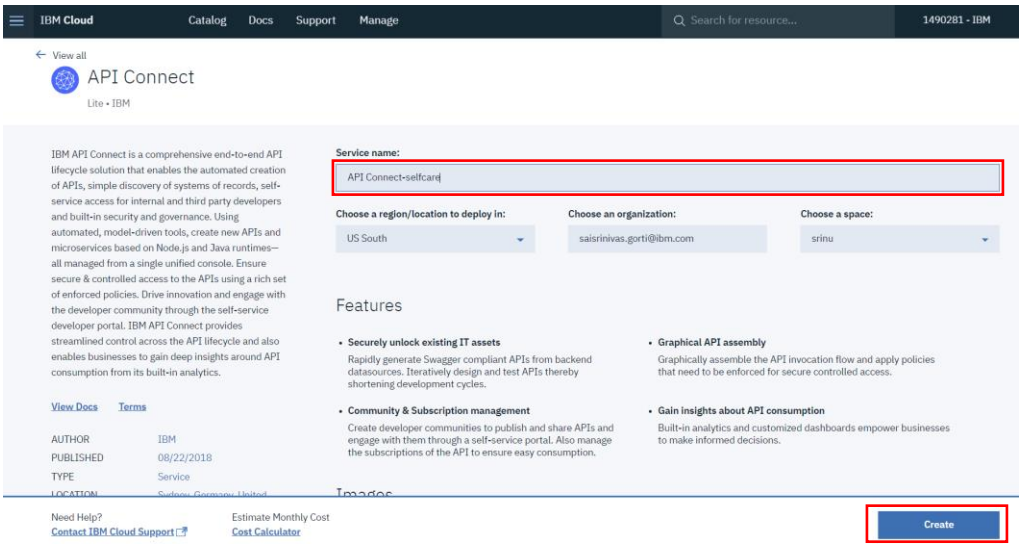
- Step (1)
  Within the IBM Cloud Dashboard, navigate to the Catalog using the "Catalog" Button in the upper toolbar

- Step (2)
  Once in the Catalog, Search for "**API Connect**" and click the Description Card. Alternatively, click the "**Integration**" Category in the Left-hand filter menu and look for "**API Connect**" as shown.



- Step (3)

  Give your API Connect service name and select the region, organization and space from the dropdown lists to create API Connect service.



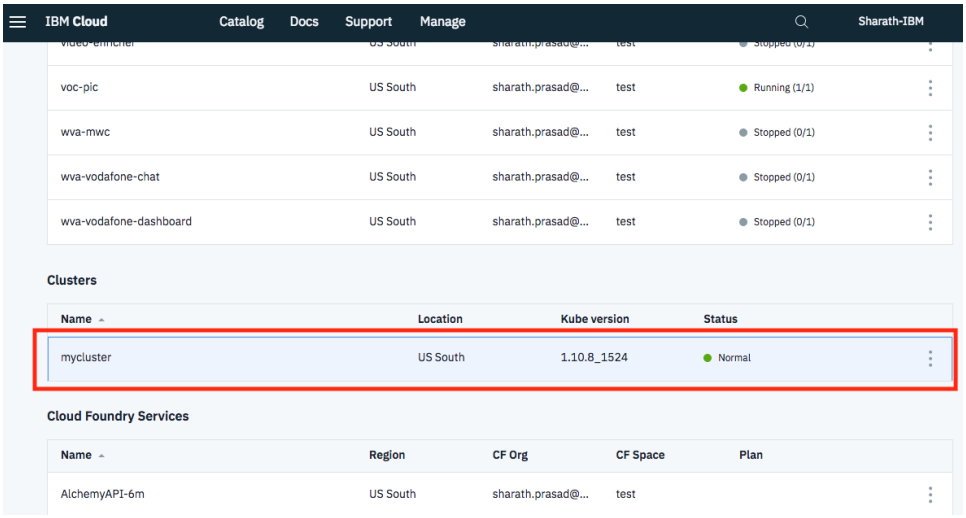
- Step (4)

  You will then be redirected to the IBM Cloud Dashboard. Your API Connect Service will appear under the “Cloud Foundry Services” Header.

4. Section1 - Deploy the TMF Open Digital on IBM Cloud on Kubernetes

After about 30 mins, your cluster on the Kubernetes service should be ready. Navigate to the IBM Cloud dashboard **https://console.bluemix.net/**

Find your cluster as shown below:



Gain access to your cluster:

Open a terminal and log into ibmcloud using the following command:

ibmcloud login -a https://api.ng.bluemix.net (or) **ibmcloud login --sso**

Once you have logged in, you need to set the environment variable and download the Kubernetes configuration files. Run the below command with your cluster name.

ibmcloud cs cluster-config *YOUR CLUSTER NAME*

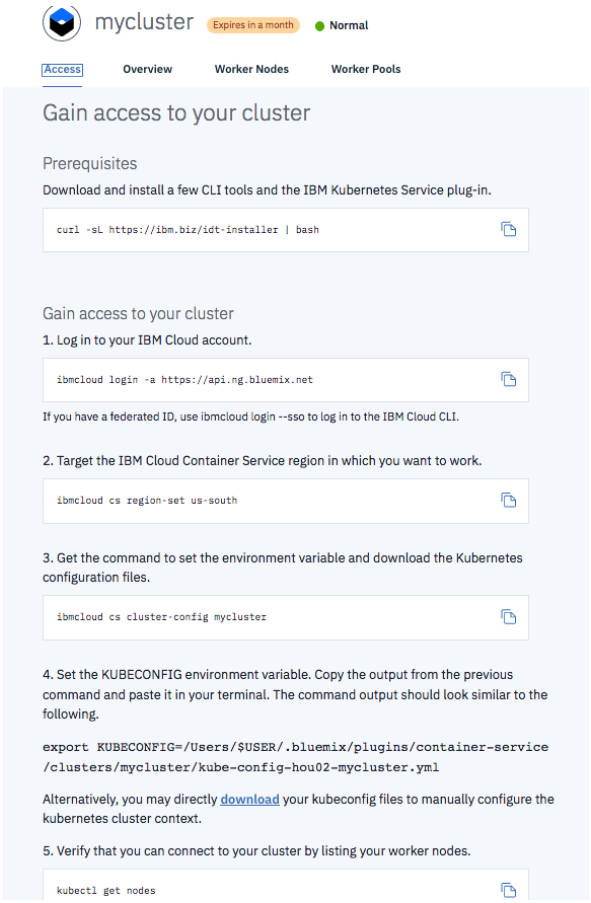
Set the KUBECONFIG environment variable. Copy the output from the previous command and paste it in your terminal. The command output should look like the following.

export KUBECONFIG=/Users/$USER/.bluemix/plugins/container-service/clusters/mycluster/kube-config-hou02-mycluster.yml

Verify that you can connect to your cluster by listing your worker nodes

kubectl get nodes

The above instructions are also available for your cluster in the access tab as shown below:



Deploy the Lab on cluster:

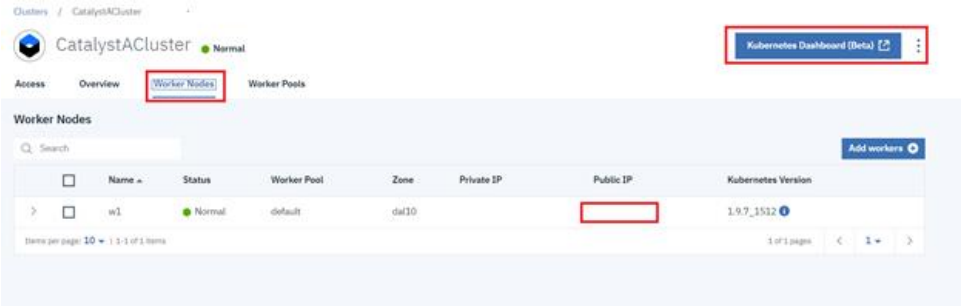
Go to the git hub repo and download the sandbox.yaml file. The link is provided below:

https://github.com/commgsc/CTA_LAB_2018/blob/master/Kubernetes/sandbox.yaml

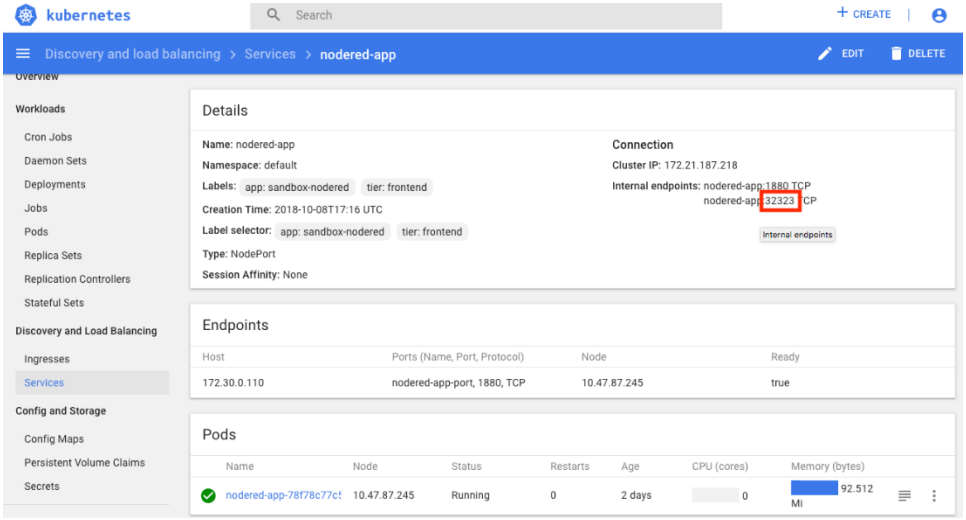
kubectl create –f https://raw.githubusercontent.com/commgsc/CTA_LAB_2018/master/Kubernetes/sandbox.yaml

The above command deploys the TM Forum Open Digital Lab which comprises of containerized microservices for mongodb, mqtt, kafka, openwhisk and environments for development on nodejs and node-red. In addition, it also contains a deep learning models for visual recognition.

Navigate to IBM Cloud dashboard using this link https://console.bluemix.net/. Click on your cluster in the clusters section. Click "**Worker Nodes**" tab to note your public IP.



Launch "**Kubernetes Dashboard (Beta)".** Click services in the menu on left and get the port number for nodered-app as shown below.



Use the Public IP and port number in the format below to access the node red application.

Node-red Flow editor:
Url - Public IP: port (Ex: 184.172.214.2:32323)
This url opens the node-red application with a pre-defined flow.

5. Section 2 – Creating a self-care app

5.1. Microservices

5.1.1. API Connect

IBM API Connect integrates IBM API Management and IBM **Strong Loop** into a single package with built-in gateway, which allows you to create, secure, run and manage APIs and Microservices. IBM API Connect utilizes Strong Loop capabilities to rapidly build API's and Microservices using Node.js – Loopback and Express frameworks. It uses Model driven approach to create API's, map models to back-end systems using available connectors.

For this lab, we took the swagger files from TMF OPEN API TABLE and using the definitions and paths defined in the swagger files, we were able to generate persistent models using API Connect (apic command line). These models generated

stores and retrieves information from MongoDB. Once the models are generated and linked to database, we test the models and were able to push it to **IBM API Connect** on **IBM Cloud**. Once these APIs are published to IBM Cloud, they can be accessed from anywhere.

5.1.2. Create an APIC project using CLI

Open command line or Terminal and run the loopback application generator "**apic loopback**" to create a new application and follow the prompts to give your application a name and hit "**enter**" to have a default directory of project, select "**3.x (current)**" as the loopback version and select "**empty-server"** option to create an empty LoopBack API.

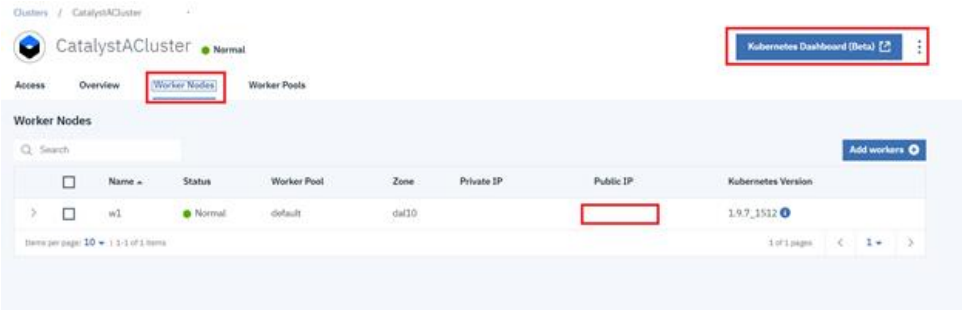
```
C:\Srinivas\IBM projects\apic-test>apic loopback
? What's the name of your application? AccountAPI
? Enter name of the directory to contain the project: AccountAPI
? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind?
> empty-server (An empty LoopBack API, without any configured models or datasources)
  hello-world (A project containing a controller, including a single vanilla Message and a single remote method)
  notes (A project containing a basic working example, including a memory database)
```

Running **apic** command for the very first time asks for a couple of prompts for review of license terms. Just hit "**yes**" and the next prompt would be for feedback, just hit "**no**".

5.1.3. Accessing database credentials

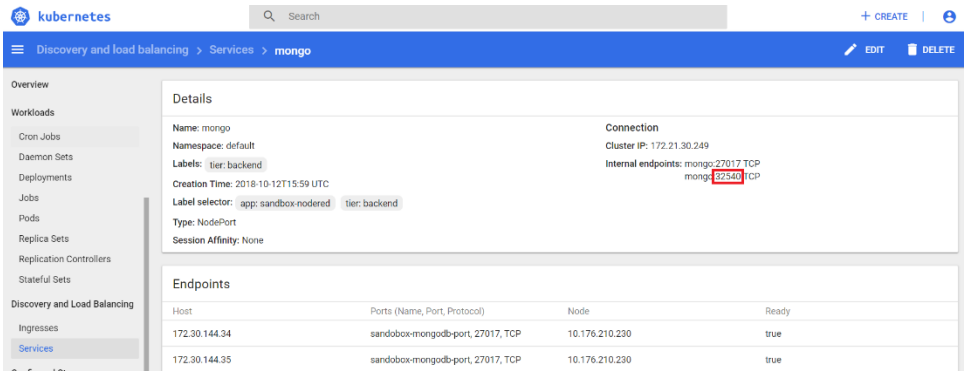
Navigate to IBM Cloud dashboard using this link https://console.bluemix.net/. Click on your cluster in the clusters section. MongoDB is installed into kubernetes cluster as a part of section 4. Below steps will explain how to get its credentials.

Click "**Worker Nodes**" tab to note your public IP.



Launch "**Kubernetes Dashboard (Beta)** "to get the port number of mongodb service. Navigate to "**Discovery and load balancing --> Services**".

Look for mongodb service endpoints, default port number for mongodb is 27017, note down the other port number.



These credentials are used to access the mongoDB published on kubernetes cluster. Note down the host and port of mongodb.

Table 3:

Host:	
Port:	

5.1.4. Create a data Source

This can be done using CLI as well as API designer. Use the above noted Public IP address as host and port to connect the database. Navigate to the application folder you just created using "**cd <application-folder-name>**".

Ex: **cd AccountAPI**

Command Line Interface:

Create a Data Source using apic CLI using "**apic create --type datasource**". Enter a data-source name and select "**Mongodb (supported by StrongLoop)**" as connector for db. Then follow the prompts to provide **host, port**. Just hit enter for the other configuration parameters. Select "**Yes**" for Install loopback-connector-mongodb.

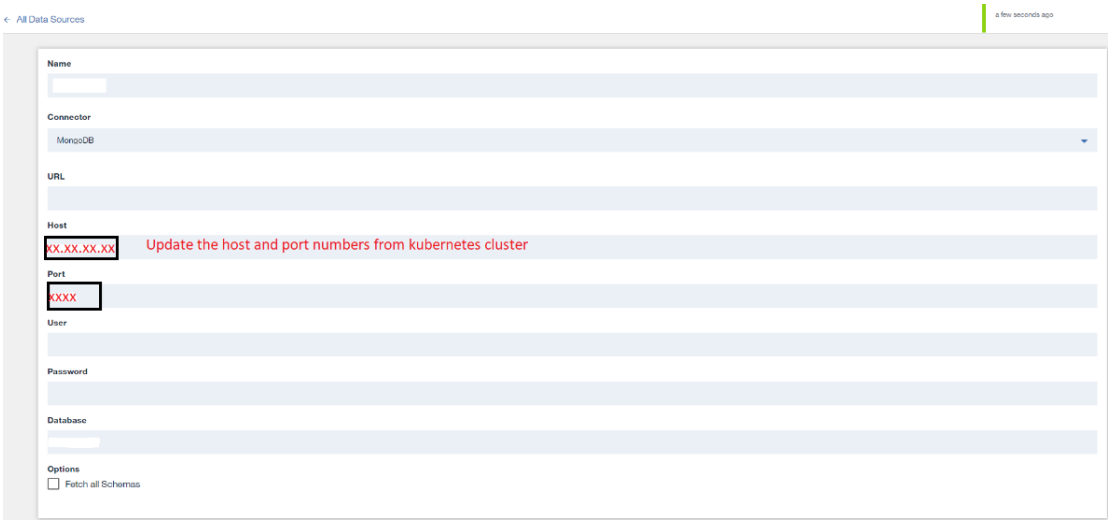
```
C:\Srinivas\IBM projects\October Lab\account>apic create --type datasource
? Enter the data-source name: test-db
? Select the connector for test-db: MongoDB (supported by StrongLoop)
Connector-specific configuration:
? Connection String url to override other settings (eg: mongodb://username:password@hostname:port/database):
? host: xxx.xxx.xxx.xxx
? port: xxxxx
? user:
? password:
? database:
? Install loopback-connector-mongodb@^1.4 (Y/n) Y
```

Use the host and port here,
hit enter for the rest.

API Designer:

The API Connect Designer is a GUI that allows developers to graphically create and manage their APIs. Navigate to your application folder in command line or

terminal using **“apic edit”** command. API Designer opens in a browser, Navigate to Data Sources section of API Designer. Provide details of the database to be used in the application. Use the above noted public IP address as host and port number to connect apic application to database.



5.1.5. Create API’s using a swagger file

Look for “TMFOpenApi” folder of the github repository to find simplified openAPIs. These API files can be downloaded or the url from github can be used to import these APIs. These Swagger files or openAPIs are a subset of the original TMForum APIs which can be downloaded from TM Forum Open API table using this link https://projects.tmforum.org/wiki/display/API/Open+API+Table. The original APIs are simplified according to our selfcare app use case.

Note: For simplicity download all three .yaml files and repeat this step of importing swagger files and linking mongodb data source for all of them.

Navigate to your loopback application folder in the command line or terminal and run “**apic loopback:swagger**” to import TM Forum API into the sample application you created. Provide the File path or url of the TM Forum Open API of the github repository. Follow the prompts to select all the models and link data sources created in the above step for those respective models. Select the pre-populated data source which we created in the above step.

```
C:\Srinivas\IBM projects\TMF_Test\selfcare-account>apic loopback:swagger
? Enter the swagger spec url or file path: ../AccountManagement.yaml
Loading ../AccountManagement.yaml...
? Select models to be generated: swagger_tmf-api_accountManagement_v2_, BillingAccount
? Select the data-source to attach models to: (Use arrow keys)
> selfcare-app-db (mongodb)
  (no data-source)
```

Execution of above command creates models and links them with the datasource.

```
C:\Srinivas\IBM projects\TMF_Test\selfcare-account>apic loopback:swagger
? Enter the swagger spec url or file path: ../AccountManagement.yaml
Loading ../AccountManagement.yaml...
? Select models to be generated: swagger_tmf-api_accountManagement_v2_, BillingAccount
? Select the data-source to attach models to: selfcare-app-db (mongodb)
Creating model definition for swagger_tmf-api_accountManagement_v2_...
Creating model definition for BillingAccount...
Model definition created/updated for swagger_tmf-api_accountManagement_v2_.
Model definition created/updated for BillingAccount.
Creating model config for swagger_tmf-api_accountManagement_v2_...
Creating model config for BillingAccount...
Model config created for swagger_tmf-api_accountManagement_v2_.
Model config created for BillingAccount.
Generating C:\Srinivas\IBM projects\TMF_Test\selfcare-account\server\models\swagger_tmf-api_accountmanagement_v2_.js
Models are successfully generated from swagger spec.
Done running LoopBack generator

Updating swagger and product definitions
Created C:\Srinivas\IBM projects\TMF_Test\selfcare-account\definitions\selfcare-account.yaml swagger description
```

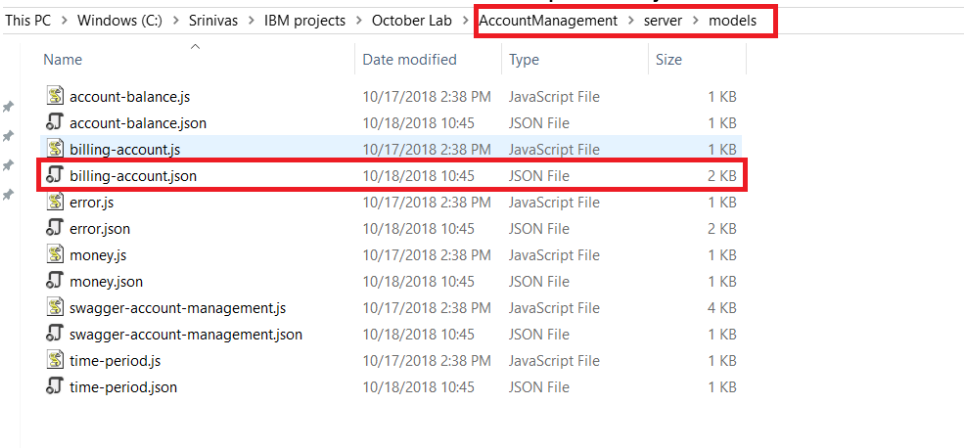
5.1.6. Edit Model and API

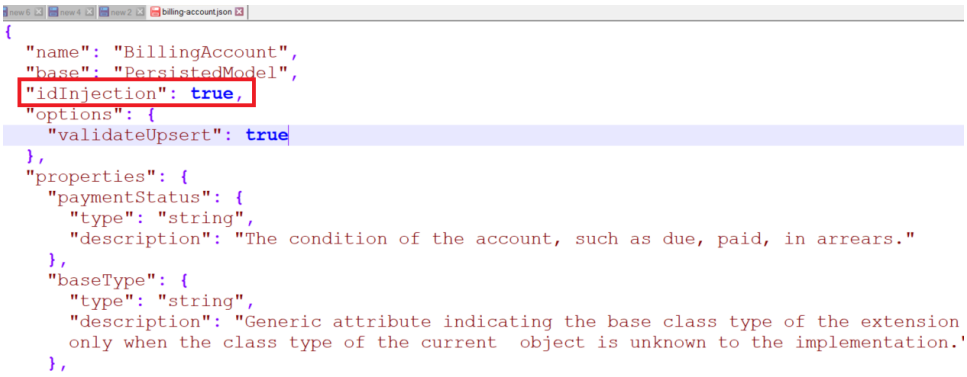
After importing the swagger file, open your apic application folder created above in windows explorer or finder in MAC. Navigate to "**server --> models**", According to the swagger file being used, use the below table to open respective json model files:

Table 4:

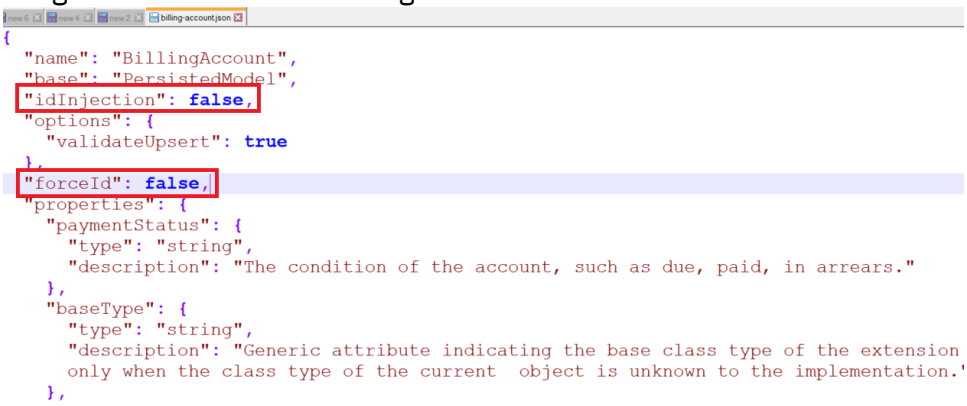
Swagger File	**Json data file**
AccountManagement.yaml	billing-account.json
UsageConsumption.yaml (optional)	Usage-consumption-report.json
UsageManagement.yaml (optional)	usage.json

Look for the similar folder structure and open the json file in text editor.

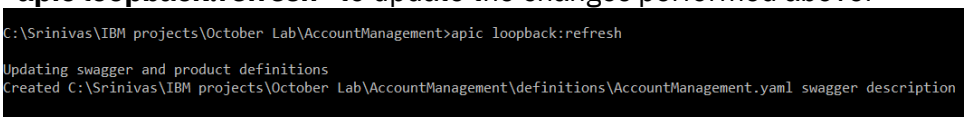




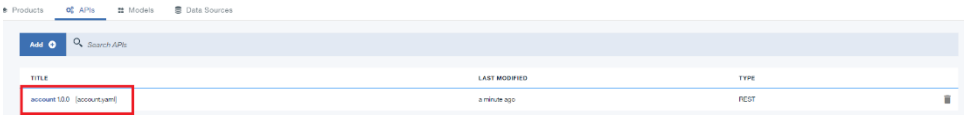
In text editor, you should be able to see a property named "**idInjection**" which is set to true by default. Change the property value to "**false"** and add a property "**forceID**" which should be set to "**false**" as well. The file should look like the image below after all the changes are done. Save it and close.

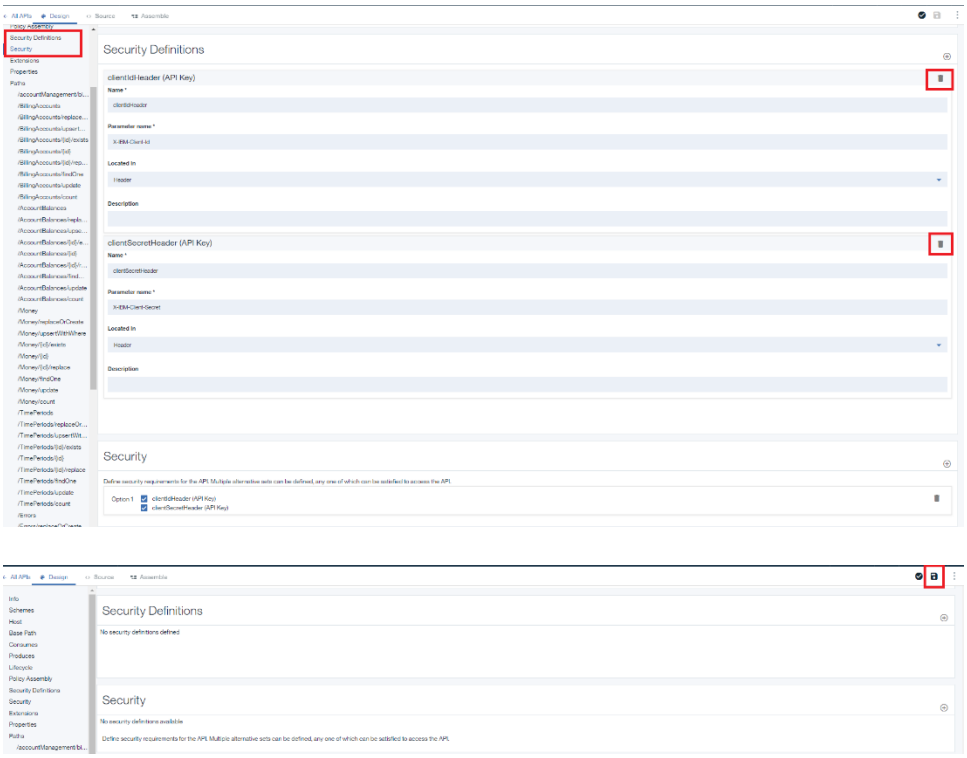


Back in command explorer or terminal execute the following command "**apic loopback:refresh**" to update the changes performed above.



Run "**apic edit**" to open the API designer. For simplicity of using and testing the APIs, let's remove the "**security definitions"** for the API. Click the API to go into design and source of the API.





Click “**Security definitions**” on the left side and delete the “**ClientIdHeader”** and “**ClientSecretHeader”** using the delete button and click “**Save**” on the top right corner.

5.1.7. Sample input data to be inserted into database

Open a browser window and navigate to the github repository for the lab. Go to “**mongodb**” folder and download all the json files. These are sample data files to be inserted into the database.

In the next step of testing and exploring your loopback application use these json files for data while using a **POST** operation.

Table 5:

Swagger File	**Json data file**
AccountManagement.yaml	billingAccount.json
UsageConsumption.yaml (optional)	usageConsumptionReport.json
UsageManagement.yaml (optional)	usage.json

5.1.8. Test and Explore your Loopback application

Ensure that you are in the loopback application directory you created earlier.

Run “**cd <loopback-application>**” in your command line or terminal.

Launch the API Connect designer

Launch the API Connect designer using "**apic edit**".
After a brief pause, the following message is displayed.
Express server listening on http://127.0.0.1:9000/

Start your LoopBack application
On the bottom left of the API Designer, hit the Play button to start your application.

After a short delay, your application will change to "Running".

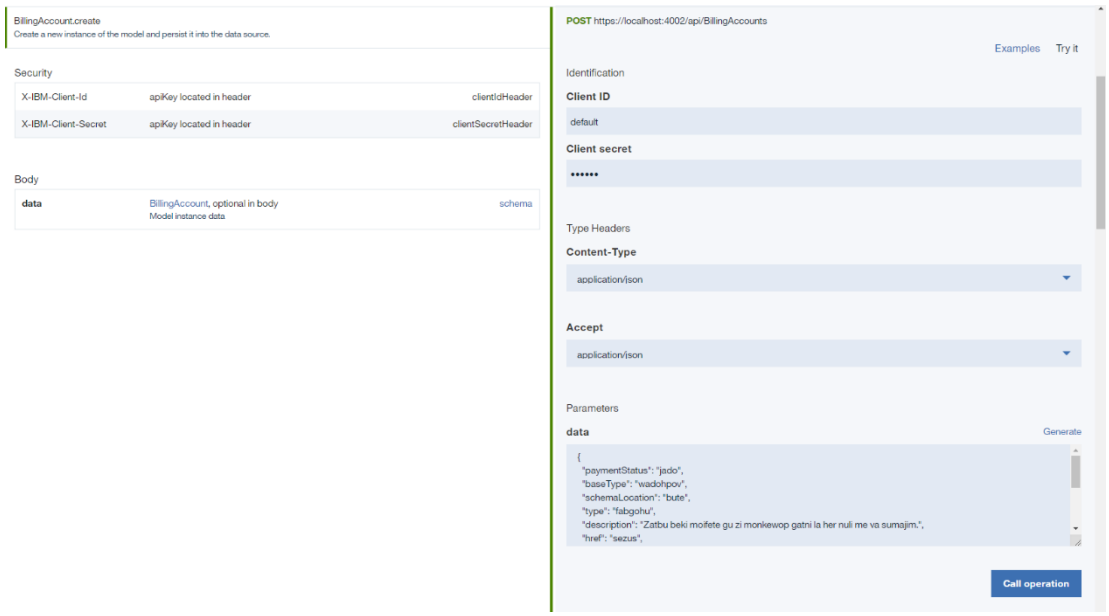
Test and Explore your Swagger-based APIs
Now that the application is running, let's try calling some of the APIs!
On the top right of the API Designer, hit the "**Explore**" button. This takes you to an API Explorer, allowing you to explore the APIs defined in your generated Swagger doc. Let's try calling a series of these operations.

Table 6:

Swagger File	Json data file	POST operation
AccountManagement.yaml	billingAccount.json	BillingAccounts.create
UsageConsumption.yaml (optional)	usageConsumptionReport.json	UsageConsumptionReports
UsageManagement.yaml (optional)	usage.json	Usages

For each of the above swagger files while creating their own applications, use the respective POST operations from above table at this stage after clicking "**explore**" in the API designer of the application. Below POST operation describes how to post data for AccountManagement.yaml

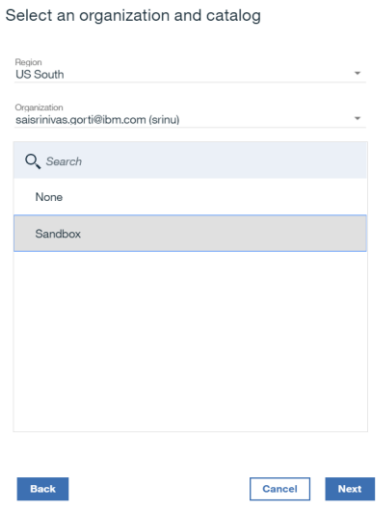
POST /BillingAccounts:
Navigate to the operation BillingAccount.create to create a database entry. Click "**Try it**" and scroll down to the "Call Operation" button, copy data from billingAccont.json and paste it in "**data**" section and click "**Call Operation**".



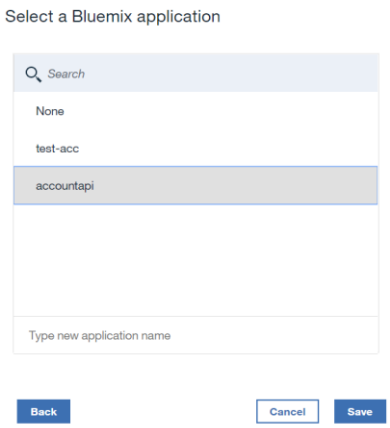
You should see a "**200 OK**" response, as well as a response body indicating that the operation was successful.
Perform similar POST operations for all the other swagger files (OpenAPI or yaml files) using respective data.

5.1.9. Publish in IBM Cloud

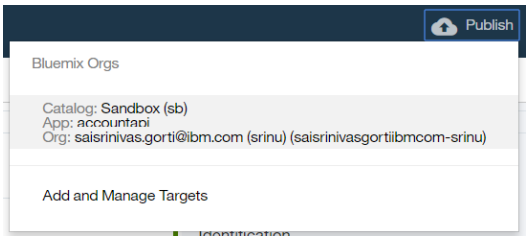
Publish the application using the "**Publish**" button on top right corner. Choose "**Add and Manage Targets**" and "**Add IBM Bluemix target**". Ensure that the organization is correct (it should correspond to your Bluemix email). Choose the "**Sandbox**" catalog.



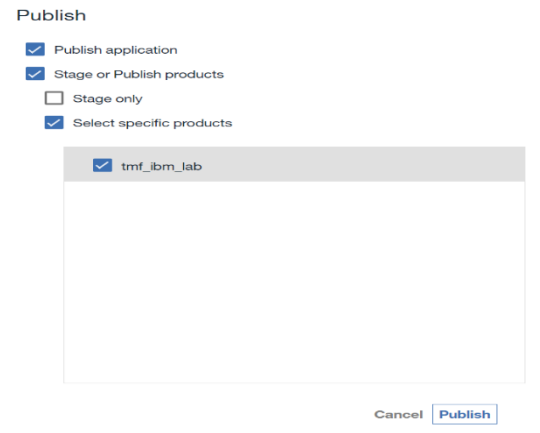
Type a new application name: "**AccountAPI**". Click the (+) button and click "**Save**".



You've created a publish target, deploy it to IBM Cloud by clicking "**Publish**" button and choosing the target you just created.

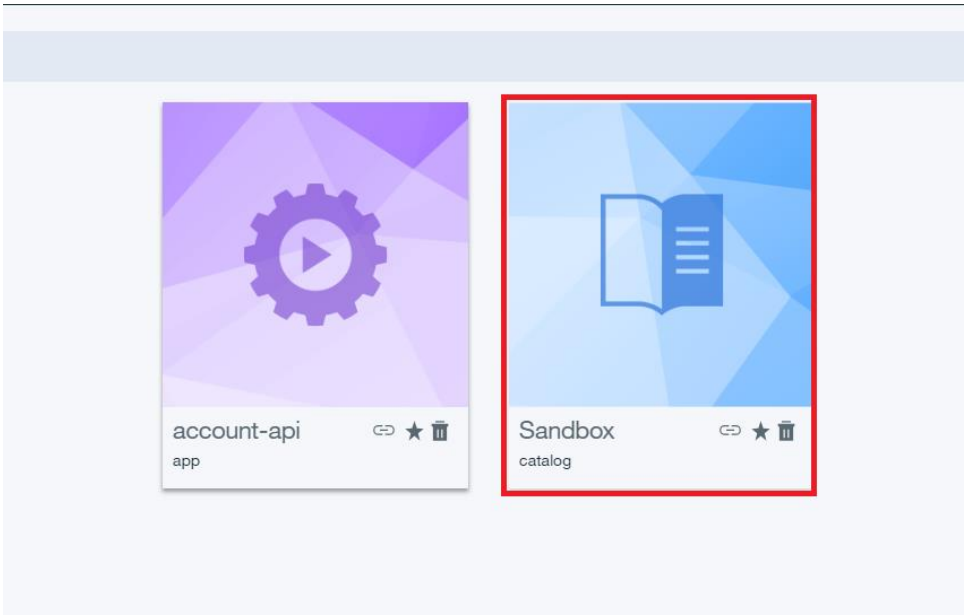


Select the below options and click "**publish"**.

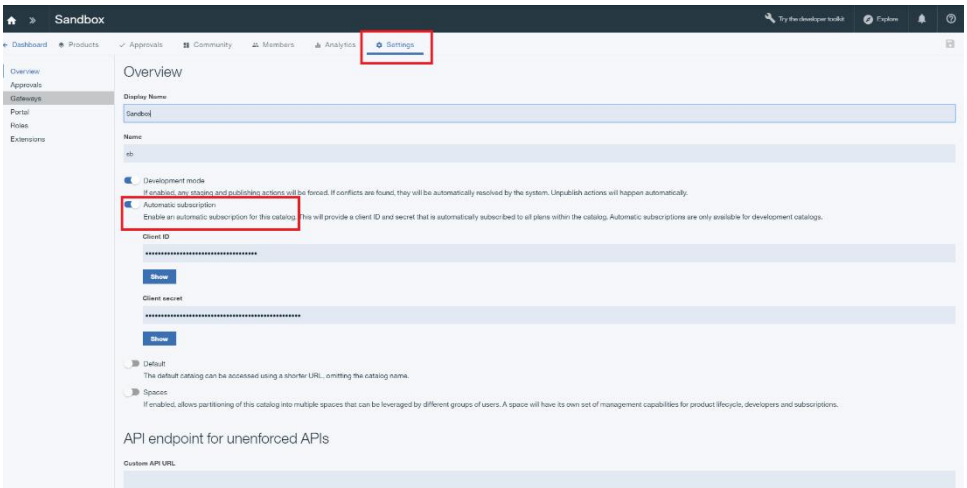


Once published and the application's state is "**started**", the application is started and your APIs are being managed by API Connect.

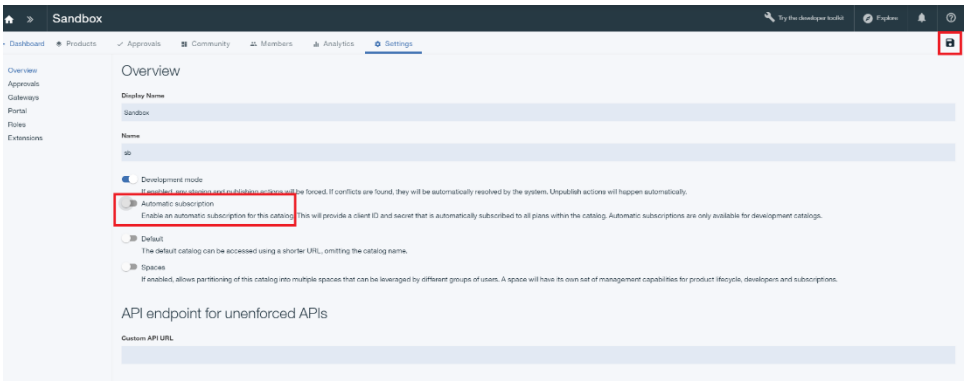
Navigate to IBM Cloud dashboard using https://console.bluemix.net and look for the API Connect instance created. You should be able to see your catalog (sandbox) and the application which was just published to API Connect.



Click on the “**sandbox**” catalog and navigate to “**settings**” tab as shown below. Look for the “Automatic Subscription” radio button and toggle the button to turn off the automatic subscription. This enables a clientId and secret to access any URLs. To test the APIs in a simple manner let’s turn off this feature.

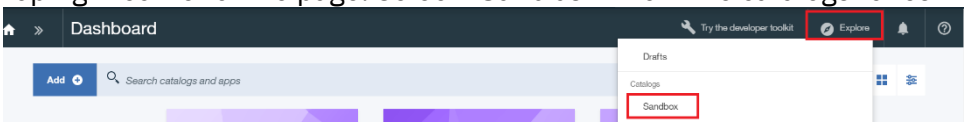


Now the settings tab should look like the image below. Click “**Save**” button on the right top corner to save the settings to our catalog.



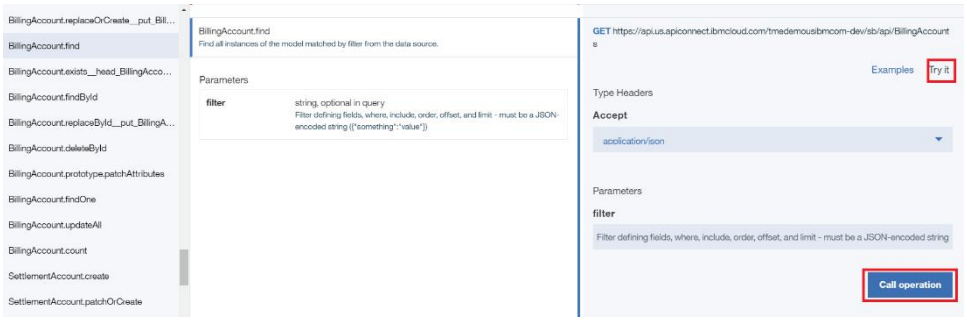
5.1.10. Test API getting data

Navigate to IBM Cloud dashboard using https://console.bluemix.net and look for the API Connect instance created. Select your catalog and click **"explore"** on the top right corner of the page. Select "**Sandbox**" from the catalogs to test APIs.



GET /BillingAccounts:

Navigate to the operation BillingAccount.find to get a list of existing BillingAccounts. Click "**Try it**" and click "**Call Operation**".



You should see a "**200 OK**" response, as well as a response body with the list of billingAccounts indicating that the operation was successful.

5.2. Functions as a service (Serverless applications)

5.2.1. IBM Functions

IBM Cloud Functions (based on Apache OpenWhisk) is a Function-as-a-Service (FaaS) platform which executes functions in response to incoming events. Given Cloud Functions' event-driven nature, it offers several benefits for user-facing applications, whereas the HTTP requests coming from the user's browser serve as the events. Cloud Functions applications use compute capacity and billed only when they are serving user requests. Idle standby or waiting mode is nonexistent. This feature makes Cloud Functions considerably less expensive when compared to traditional containers or

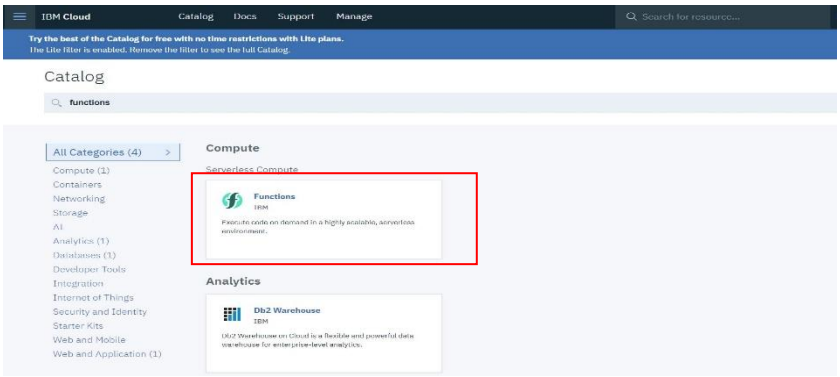
Cloudfoundry applications. Both of which can spend most of their time idle, waiting for inbound user requests, and being billed for all that “sleeping” time.

IBM has a growing list of runtime execution options, which now includes, Java, Swift, Node, Python, PHP and other languages of your choice using Docker.

5.2.2. Access IBM Functions

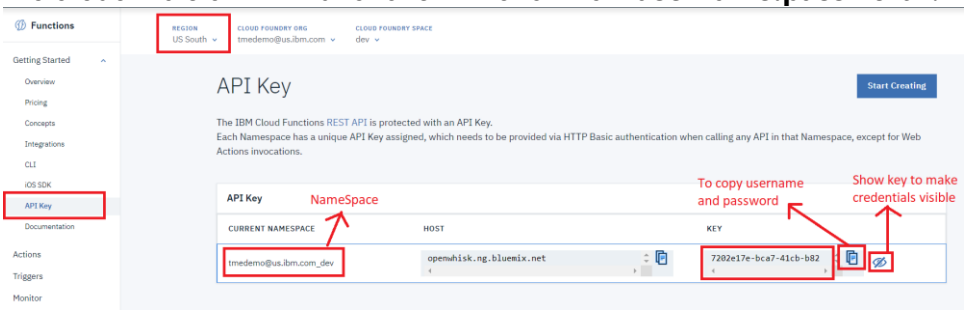
Navigate to catalog.mybluemix.net to reach the IBM Cloud Catalog.
Search “**Functions**” to find IBM Functions Service.

****Note: Please make sure that the actions being created on IBM Functions as a part of this lab are on the same cloudfoundry organization and same space as the Watson assistant service created above. This way Watson assistant and IBM Functions can call each other.**



5.2.3. Get IBM Functions credentials

Navigate to “**API Key**” of Getting started menu (left hand side), Change the region to **US South** on the top. Look for the API Key section on the right having Current namespace, host and key. The hidden key (click the show key to make it visible) has the credentials of IBM Functions in the form of “**username:password**”.



Copy and paste these credentials in a notepad or any text editor as these are required in Watson Assistant to call IBM Functions.

Table 7:

IBM Functions Namespace:	
Username:	
Password:	

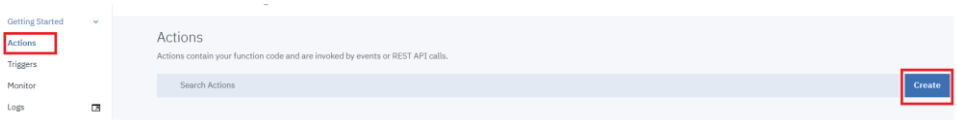
Take your namespace from above table 7 and remove '@' and '.' from it and replace underscore with hyphen (replace '_' with '-'). For instance, '**tmedemo@us.ibm.com_dev**' would be replaced as '**tmedemousibmcom-dev**'. Note this replaced text as API Connect namespace.

Table 8:

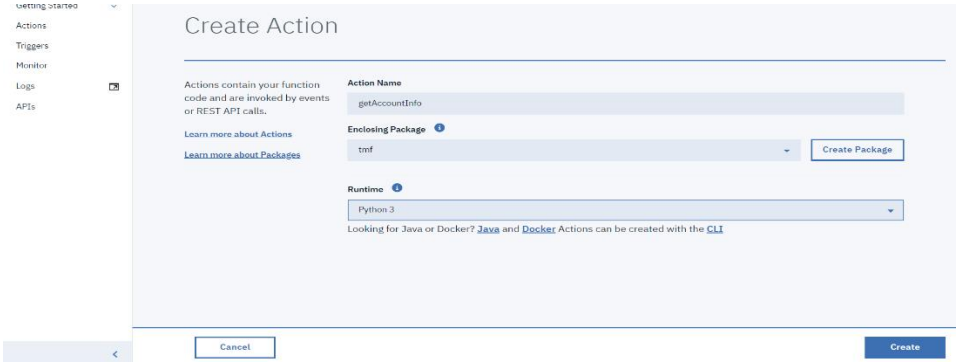
IBM Functions Namespace:	
Username:	
Password:	
API Connect Namespace:	

5.2.4. Create action 1 and test

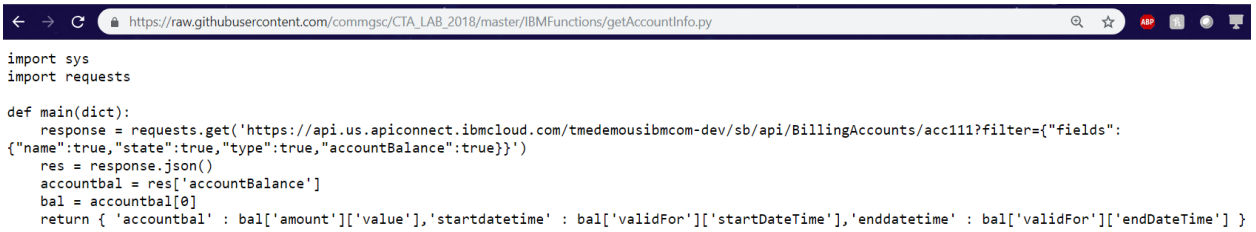
Navigate to "**Actions**" in the Left-hand menu. Click the Blue "**Create**" Button on the right-hand side and click "**Create Action**" to create a new action.



Enter "**getAccountInfo**" as the Action Name.
Click "**create Package**" and enter package name as "**TMF**".
Select the package created above using the dropdown. Select Python 3 as your Runtime in the Dropdown Menu. Then click the "**Create**" Button in the lower right corner.



In a separate browser tab, navigate to: Within the Git repository, navigate to CTA_Lab_2018/IBMFunctions/getAccountInfo.py. Click "**Raw**" and copy the code from "getAccountInfo.py"



Back in your IBM Functions Tab, paste the code from the Git Repository in the "code" editor in the center of the page and click "**Save**"

Change API Connect endpoint:
Please perform this section only if a loopback application has been created for AccountManagement.yaml as demonstrated in section 5.1. Skip this if you haven't published loopback application to IBM Cloud.
In the above IBM Functions code, replace the text between **ibmcloud.com** and **sb** with the API Connect namespace from table 8 of section 5.2.3.

For instance,
Original URL: https://api.us.apiconnect.ibmcloud.com/tmedemousibmcom-dev/sb/api/BillingAccounts/acc111?filter={"fields":{"name":true,"state":true,"type":true,"accountBalance":true}}

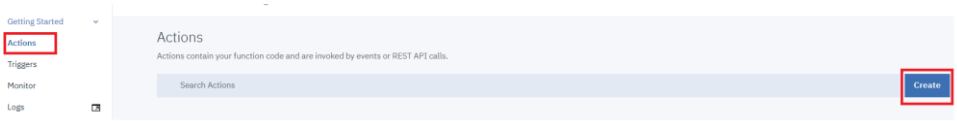
Modified URL: https://api.us.apiconnect.ibmcloud.com/***Replace with API Connect namespace from table 8***/sb/api/BillingAccounts/acc111?filter={"fields":{"name":true,"state":true,"type":true,"accountBalance":true}}

Validating created IBM Function:
Back on the main page, Click the "Invoke" Button above the code editor in the center of the page. You can see the results of your test in the "Activations" Window shown at right in the image below:

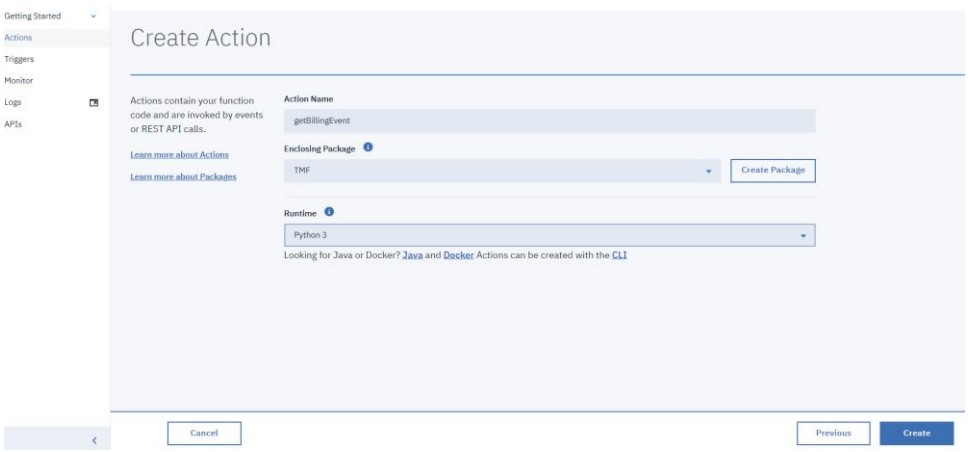
If the function is executed properly without any errors, we will be able to see results as shown above.

5.2.5. Create action 2 and test

Navigate to “Actions” in the Left-hand menu. Click the Blue “**Create**” Button on the right-hand side and click “Create Action” to create a new action.



Enter “**getBillingEvents**” as the Action Name. Select the package name created above using the dropdown. Select Python 3 as your Runtime in the Dropdown Menu. Then click the blue “**Create**” Button in the lower right corner.

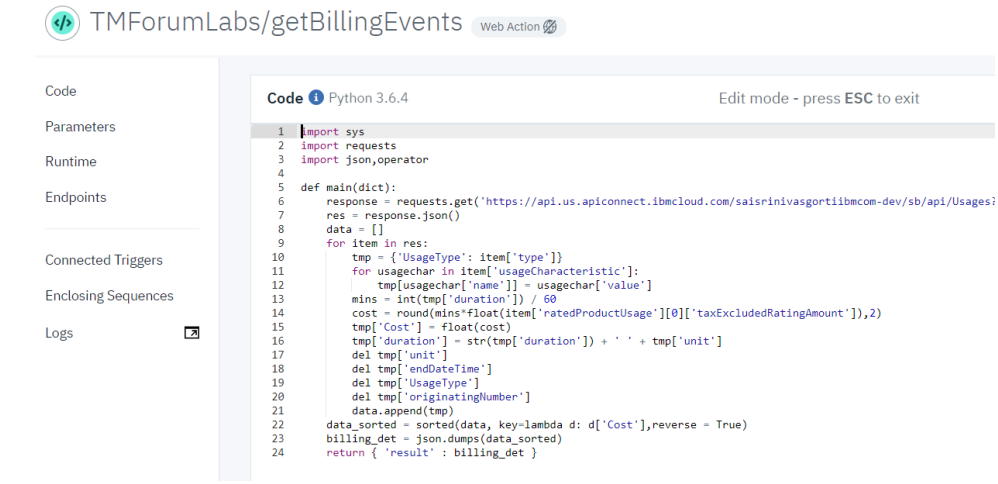


In a separate browser tab, navigate to: Within the Git repository, navigate to CTA_Lab_2018/IBMFunctions/getBillingEvents.py. Click “**Raw**” and copy the code from “getBillingEvents.py”

https://raw.githubusercontent.com/commgsc/CTA_LAB_2018/master/IBMFunctions/getBillingEvents.py

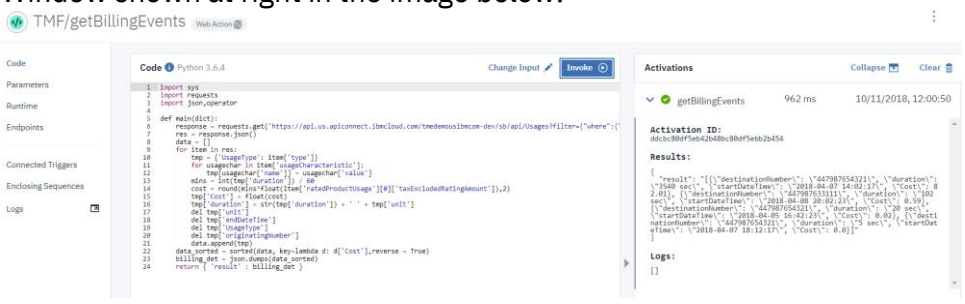
```
import sys
import requests
import json,operator

def main(dict):
    response = requests.get('https://api.us.apiconnect.ibmcloud.com/tmedemousibmcom-dev/sb/api/Usages?filter={"where":
{"relatedParty.role":"customer","relatedParty.id":"cst123","type":"voice","date":{"gt":"2018-04-01"}}}')
    res = response.json()
    data = []
    for item in res:
        tmp = {'UsageType': item['type']}
        for usagechar in item['usageCharacteristic']:
            tmp[usagechar['name']] = usagechar['value']
        mins = int(tmp['duration']) / 60
        cost = round(mins*float(item['ratedProductUsage'][0]['taxExcludedRatingAmount']),2)
        tmp['Cost'] = float(cost)
        tmp['duration'] = str(tmp['duration']) + ' ' + tmp['unit']
        del tmp['unit']
        del tmp['endDateTime']
        del tmp['UsageType']
        del tmp['originatingNumber']
        data.append(tmp)
    data_sorted = sorted(data, key=lambda d: d['Cost'],reverse = True)
    billing_det = json.dumps(data_sorted)
    return { 'result' : billing_det }
```

Back in your IBM Functions Tab, paste the code from the Git Repository in the “code” editor in the center of the page.



Validating created IBM Function:

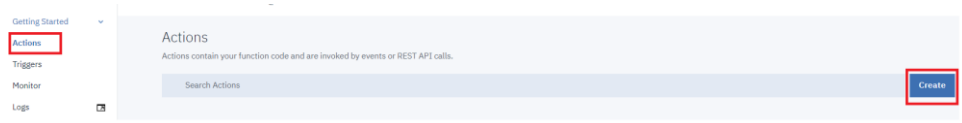
Back on the main page, Click the “Invoke” Button above the code editor in the center of the page. You can see the results of your test in the “Activations” Window shown at right in the image below:



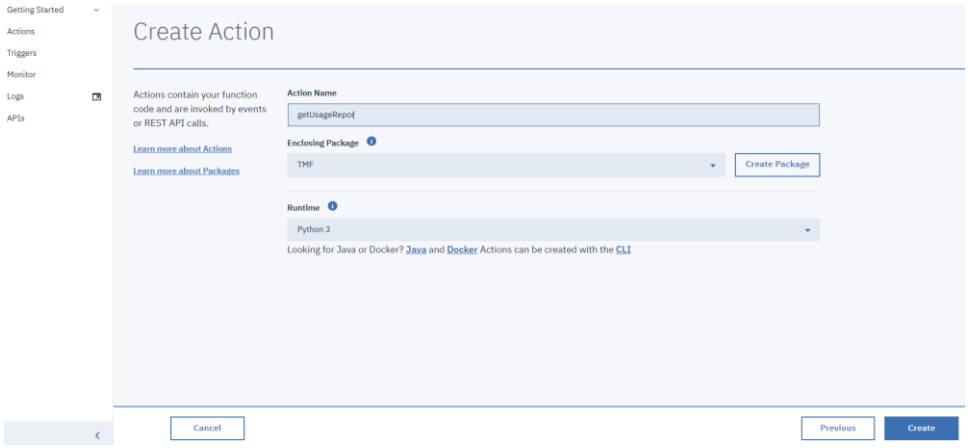
If the function is executed properly without any errors, we will be able to see results as shown above.

5.2.6. Create action 3 and test

Navigate to “Actions” in the Left-hand menu. Click the Blue “**Create**” Button on the right-hand side and click “Create Action” to create a new action.



Enter “**getUsageReport**” as the Action Name. Select the package name created above using the dropdown. Select Python 3 as your Runtime in the Dropdown Menu. Then click the blue “**Create**” Button in the lower right corner.



In a separate browser tab, navigate to: Within the Git repository, navigate to CTA_Lab_2018/IBMFunctions/getUsageReport.py. Click “**Raw**” and copy the code from “getUsageReport.py”


```
import sys
import requests
import json
import datetime

def main(dict):
    response = requests.get('https://api.us.apiconnect.ibmcloud.com/tmedemousibmcom-dev/sb/api/UsageConsumptionReports?filter={"where":
{"bucket.product.id":"mobile999"}}')
    res = response.json()
    buckets = res[0]['bucket']
    data = []
    for bucket in buckets:
        print(bucket)
        tmp = {'UsageType': bucket['usageType']}
        tmp['Usage'] = bucket['bucketCounter'][0]['value']
        tmp['Balanceleft'] = str(bucket['bucketBalance'][0]['remainingValue']) +' '+ bucket['bucketBalance'][0]['unit']
        data.append(tmp)
    usage_det = json.dumps(data)
    return { 'result':usage_det }
```

Back in your IBM Functions Tab, paste the code from the Git Repository in the “code” editor in the center of the page.

Validating created IBM Function:

Back on the main page, Click the "Invoke" Button above the code editor in the center of the page. You can see the results of your test in the "Activations" Window shown at right in the image below:

If the function is executed properly without any errors, we will be able to see results as shown above.

5.3. AI powered chatbot

5.3.1. Watson Assistant

Watson Assistant is a robust platform that allows developers and non-technical users to collaborate on building conversational AI solutions. Its graphical UI, powerful NLP and familiar developer features allow the rapid creation of anything from simple chatbots to complex enterprise grade solutions for customer service and more. With the IBM Watson Assistant service, you can build a solution that understands natural-language input and uses machine learning to respond to customers in a way that simulates a conversation between humans.

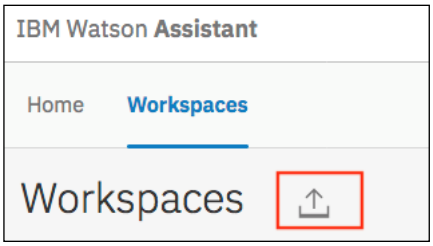
5.3.2. Creating a workspace by uploading a JSON

In a separate browser window navigate to the below githib link and download 'WA_CTA.json' or copy the content to create a file with json extension.

Github link:
https://raw.githubusercontent.com/commgsc/CTA_LAB_2018/master/WatsonAssitant/WA_CTA.json

Navigate to https://console.bluemix.net/ and look for the watson assistant created in section 3.7. Select your watson assistant from the services and click "**Launch Tool**" to launch your workspaces.

Click "**Workspaces**" and import "**WA_CTA.json"** file (json downloaded from the github repository) using the import button as shown below.



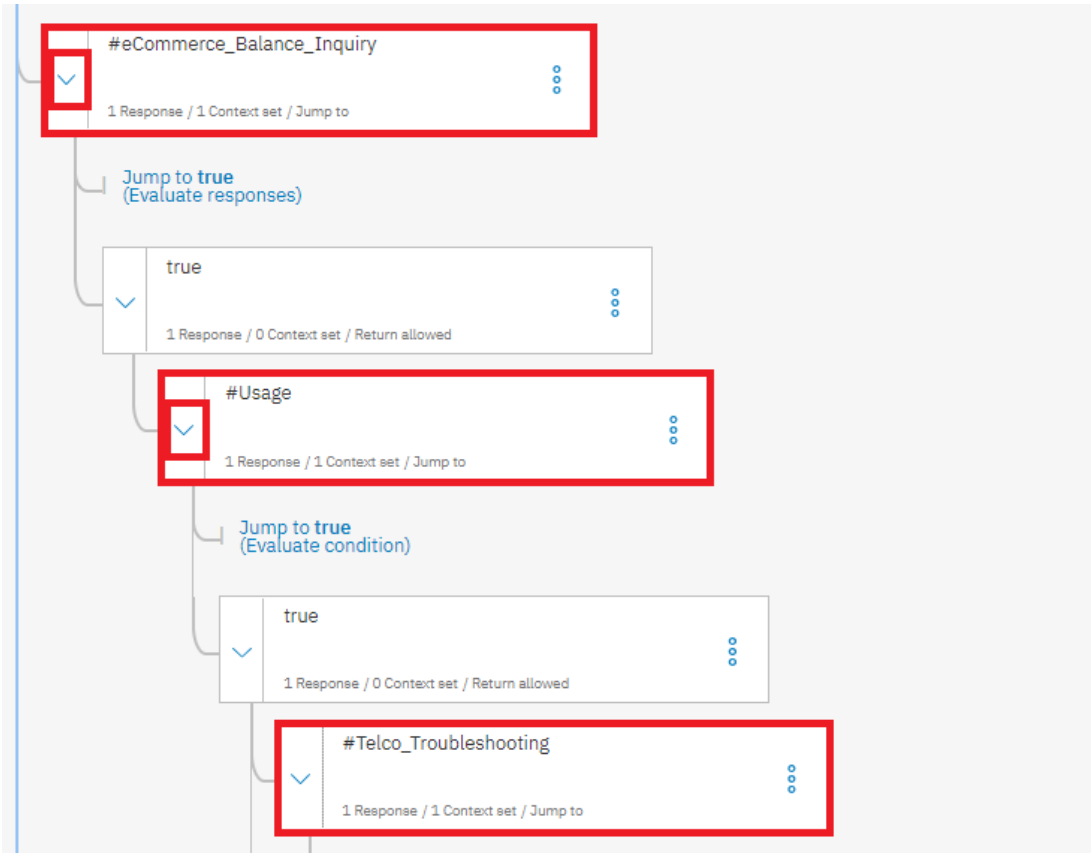
Once json file has been imported, you could view ‘Intents’, ‘Entities’ and ‘Dialog’.

5.3.3. Edit the dialog Nodes to point to the correct IBM Functions

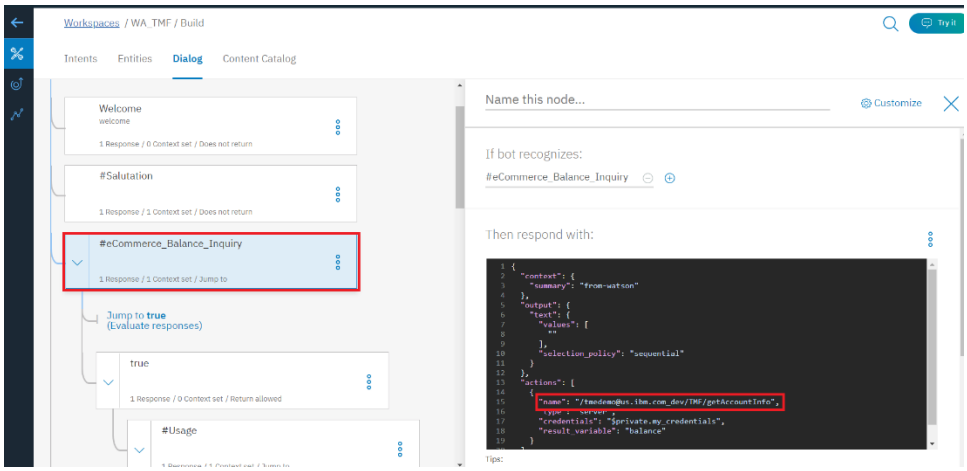
Once the workspace is imported, click on the workspace and then click on “**Dialog**”.

Look for the below intents in the dialog and expand by clicking the arrow as shown below.

- #eCommerce_Balance_Inquiry
- #Usage
- #Telco_Troubleshooting



For each of the above 3 intents, look for the attribute “**name**” in “**actions**”, this name defines which IBM Function to be called when bot recognizes this intent. The format of functions should be “**Namespace/package/function_name**”.



The above image shows the node with intent "#**eCommerce_Balance_Inquiry**", and on the right side you could see that this node calls "**getAccountInfo**" function of package "**TMF**" and namespace "tmedemo@us.ibm.com_dev". Replace these IBM Function credentials to namespace noted down in **section 5.2.3 (Table 7)** and package created in **section 5.2.4**. Note that it should match the format of IBM Functions defined above.

For instance,

Original action:

"name": "/tmedemo@us.ibm.com_dev/TMF/getAccountInfo"

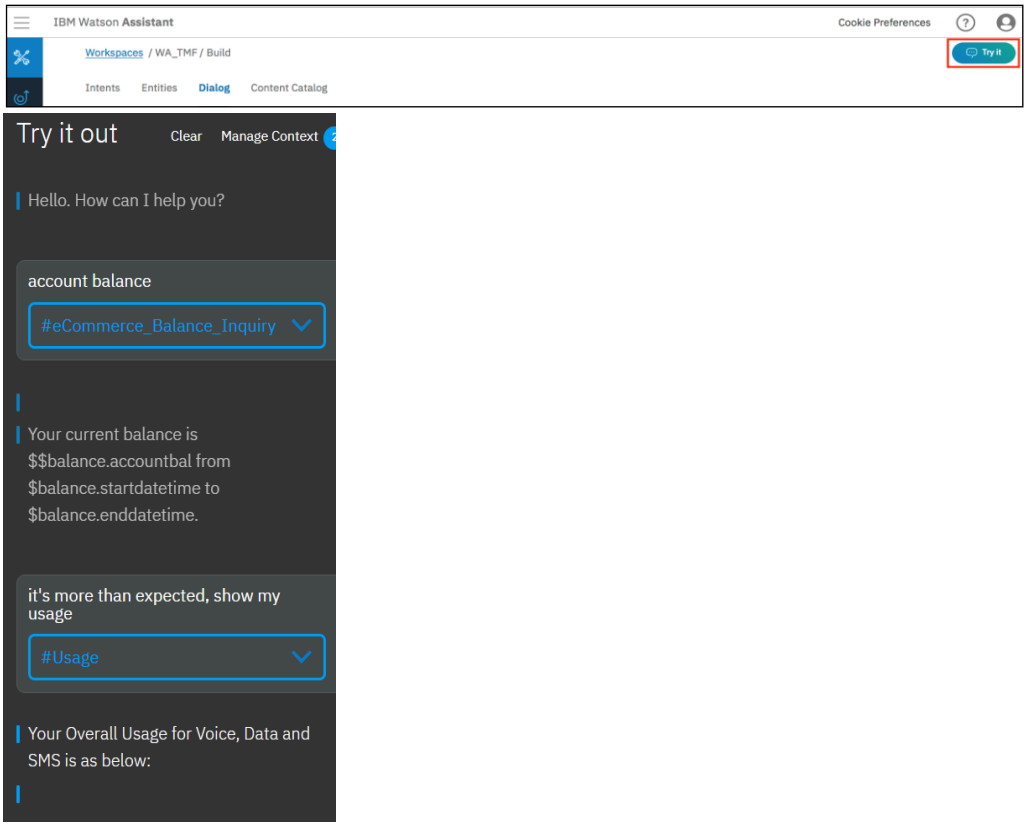
Modified action:

"name": "/***Namespace from table 7***/TMF/getAccountInfo"

Assuming that names of the IBM Functions are not changed, Change these credentials(namespace and package) for all the 3 nodes shown above.

5.3.4. Test the robot

Try out the conversation, you just created! As the credentials were not provided in the WA, we will not be able to see the actual results but when invoked through the node-red application, credentials would be provided through the application, so that we can see actual results on web application UI.



5.4. Interfacing with the end user

5.4.1. Node-red UI

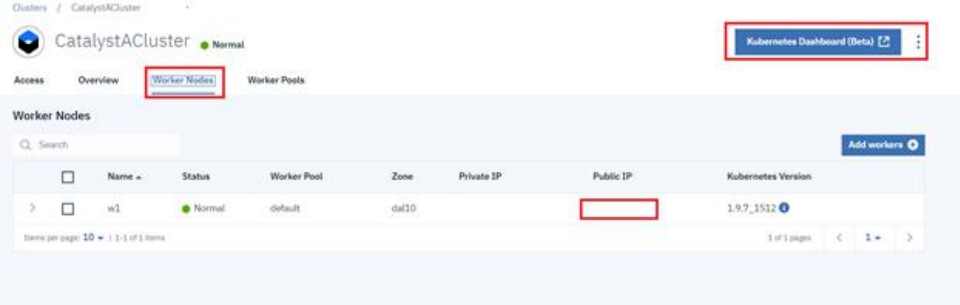
Node-RED is a powerful tool for building Internet of Things (IoT) applications with a focus on simplifying the 'wiring together' of code blocks to carry out tasks. It uses a visual programming approach that allows developers to connect predefined code blocks, known as 'nodes', together to perform a task. The connected nodes, usually a combination of input nodes, processing nodes and output nodes, when wired together, make up a 'flows'.

Node-RED provides a browser-based flow editor, which can be used to create JavaScript functions. Elements of applications can be saved or shared for re-use. The runtime is built on Node.js. The flows created in Node-RED are stored using JSON.

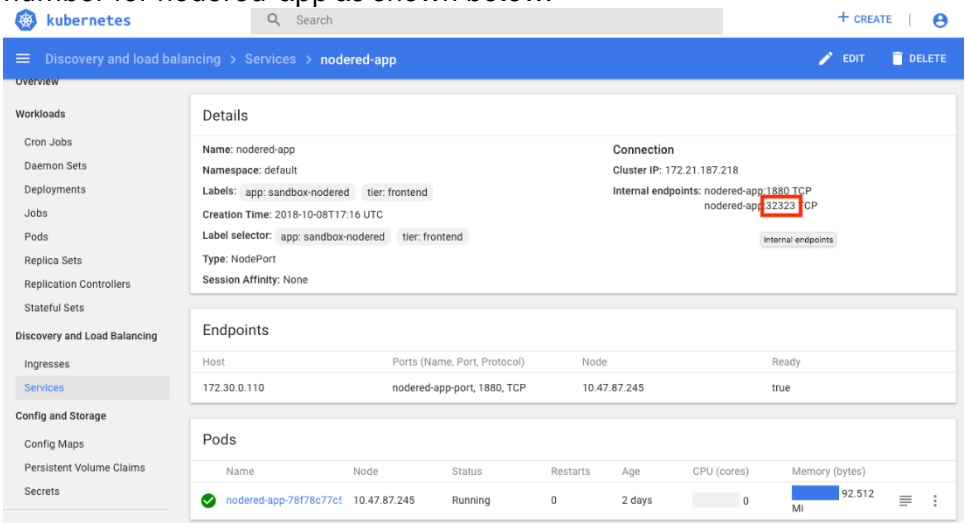
5.4.2. Navigate to Node-Red flow editor

User submits an input using node-red application UI, a http request is made which calls the JavaScript node of node-red flow to make an ajax call to Watson assistant. Watson assistant will call respective IBM Function in its context and sends a response either as a text message or an object. This response is processed by the JavaScript and response sent back using HTML template.

Navigate to IBM Cloud dashboard using this link https://console.bluemix.net/.
Click on your cluster in the clusters section.
Click “Worker Nodes” tab to note your public IP.



Launch “**Kubernetes Dashboard (Beta)**”. Navigate to services and get the port number for nodered-app as shown below.



Use the Public IP and port number in the format below to access the node red application.

Node-red Flow editor:

Url - Public IP: port (Ex: 184.172.214.2:32323)

This url opens the node-red application with a pre-defined flow.

Node-red Application UI:

Url - **Public IP: port/bot** (Ex: 184.172.214.2:32323/bot).

This url opens up the node-red application UI which is integrated with the pre-defined configuration for watson assistant, IBM functions.

Table 9:

Public IP (host):	
Port:	

5.4.3. Change credentials

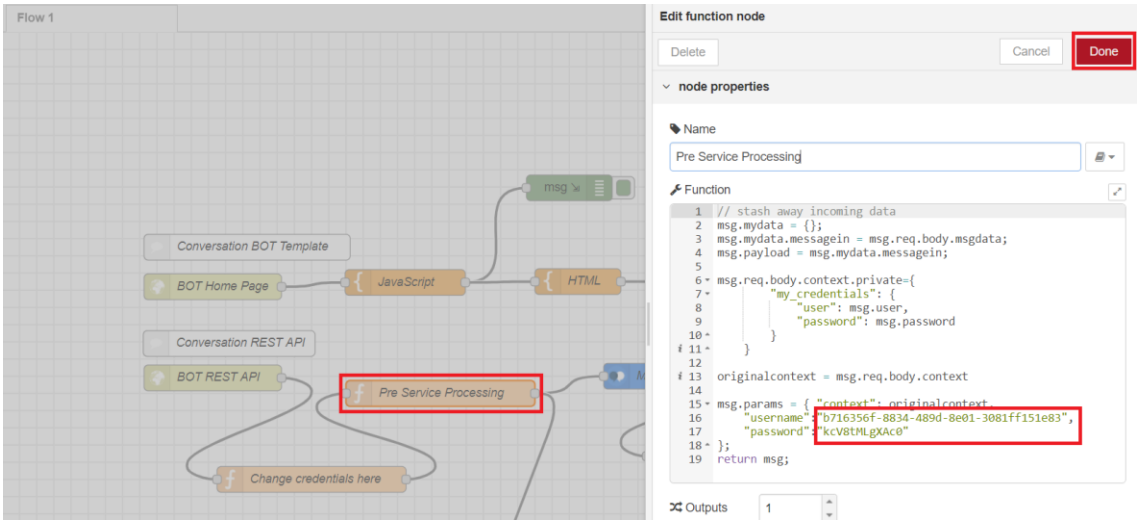
This section helps you to change credentials in IBM Functions and Watson Assistant to match with the services which were created in sections 5.2, 5.3. The pre-defined node-red flow has credentials embedded in it, this runs as such without changing any credentials.

So, skip this part if you don't want to change the credentials. But for the node-red to use the services that you just created in the above sections, IBM Functions and Watson Assistant credentials have to be changed in the existing flow.

Open the node-red flow editor using **ip:port** listed in table 9. Change the IBM Functions credentials by double clicking on "**Change credentials here**" node to open the node and change the username and password to the credentials noted down in table 7 of section 5.2.3. Click "Done" to save these changes.

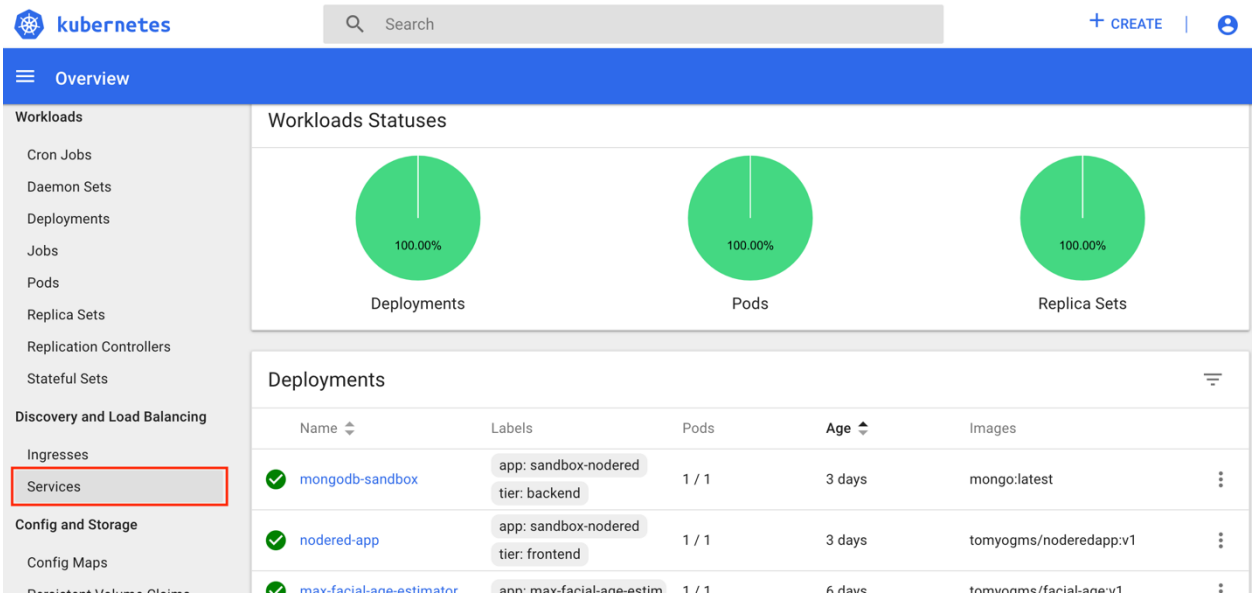
Change the Watson Assistant credentials by double clicking on "**My Bot**" node to open the node and change the Workspace_ID as noted down in table 2 of section 3.7. Click "Done" to save this change.

Double click on "**Pre Service Processing**" node. As shown below, Change the username and password over here to the Watson assistant credentials noted down in table 2 of section 3.7. Click "Done" to save this change.

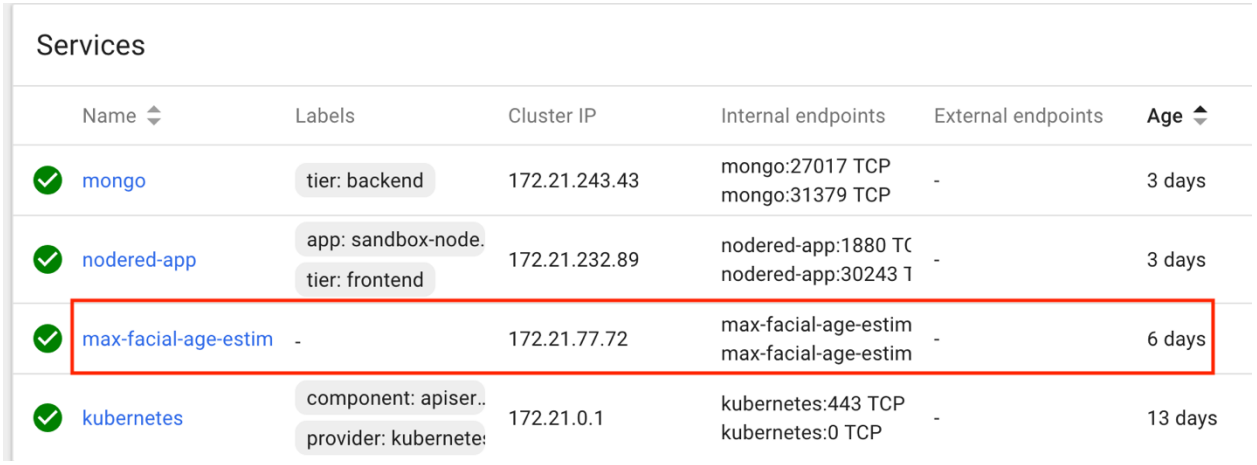


If you want to use your own deep learning model to estimate facial age, follow these steps:

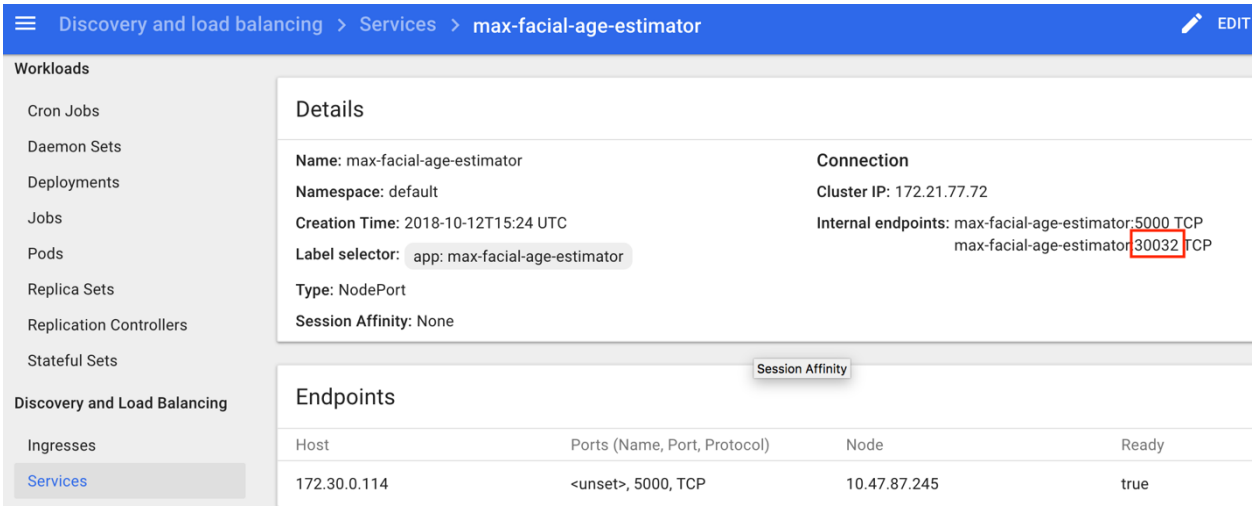
Click on services in your Kubernetes dashboard as shown below:



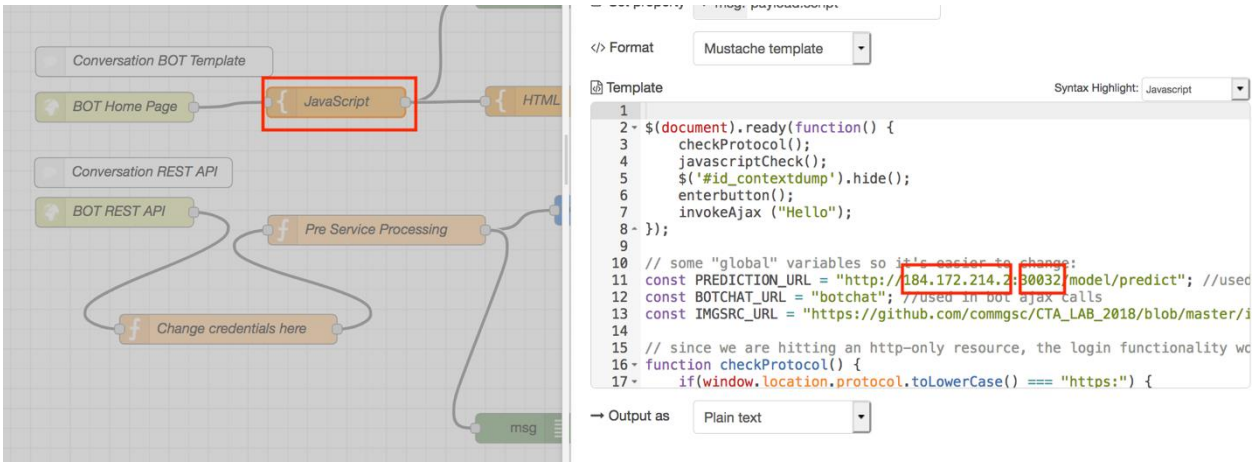
Click on the max-facial-age-estim service as shown below.



Obtain the public port for the service



Now go back to the node red flow and update the JavaScript node as shown below. Update the "**PREDICTION_URL**" variable with the public IP of your cluster and the port number you obtained from the above screenshot.



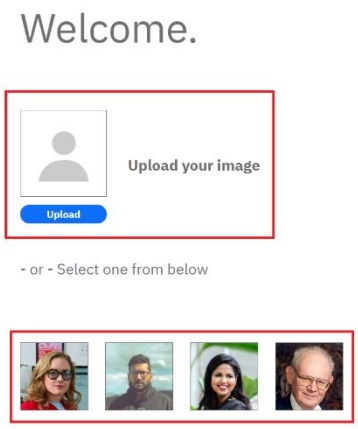
Click save and then click "**Deploy**" on the top right corner of the page. After the successful deployment, "Successfully deployed" alert would be. At this point the application is successfully deployed on IBM Cloud.

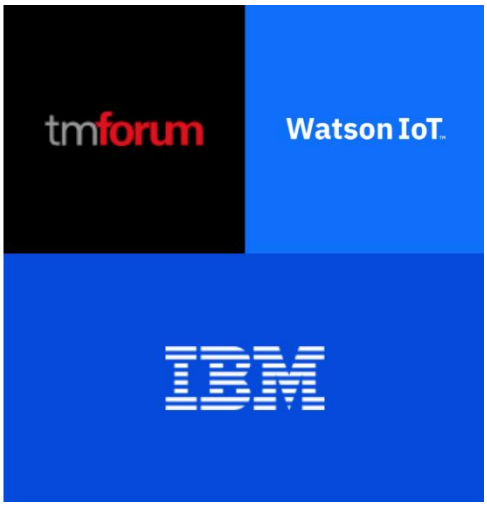
5.4.4. Access the UI and test the use case

Node-red flow editor opens up the node-red flow where individual nodes can be updated, whereas node-red application UI can be accessed using node-red application UI link from the above section 5.4.2.

Refer to above section 5.4.2 to copy the url of the application UI. Open a new browser tab, paste the url and press enter to launch the node-red application.

The selfcare node red application has a facial age detection model as its first step. This model takes image of a human being as an input to predict his/her age and then re-directs you to selfcare app. Please upload your image using the **“upload”** button or click anywhere in that area to upload an image or you can select the pre-populated images.





Once you upload your picture or select an image from the list, it takes you to next screen where the bot can assist you in self-care app instructions.
Try some inputs like “**account balance**” or “**show usage**” to initiate a conversation. These instructions will trigger intents in Watson assistant dialog which would call IBM Functions within the Watson Assistant you just created.