Ibn Tofail University

The National School of Applied Sciences, Kenitra

Mini project of IBM Certification

Done by: Idrissa TRAORE

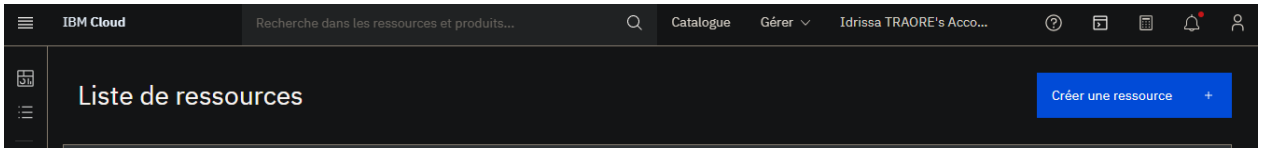
Professor: Mme Meriyem CHERGUI

Academic year 2022-2023

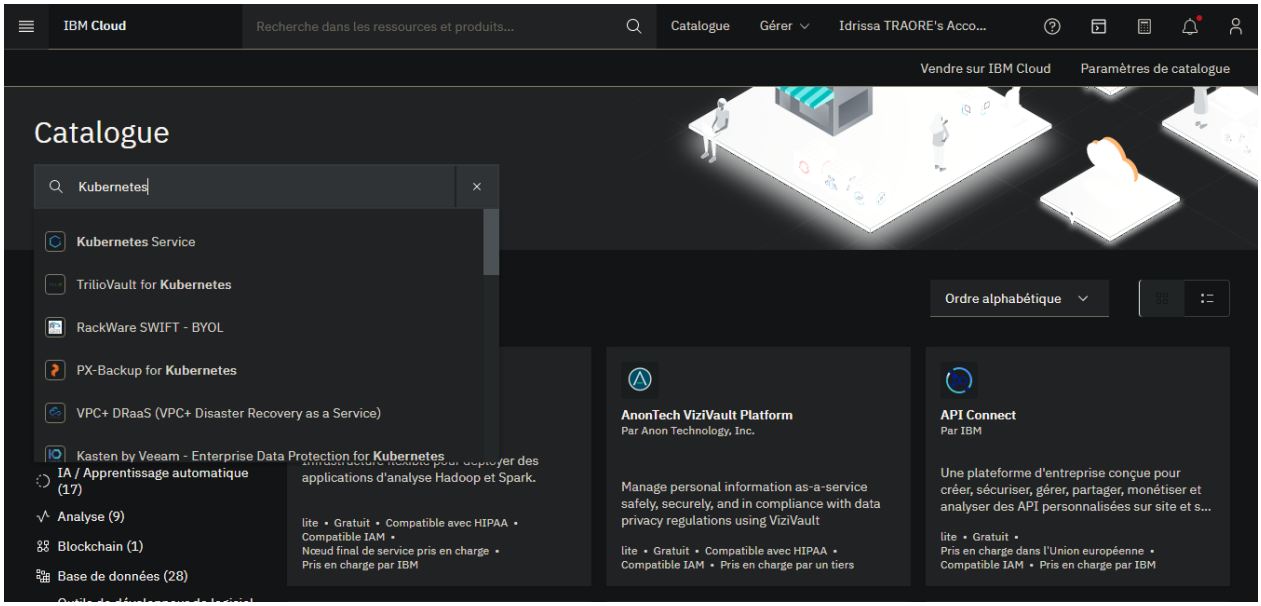
Table of Contents

Create a Kubernetes cluster on which to deploy the Node-RED app

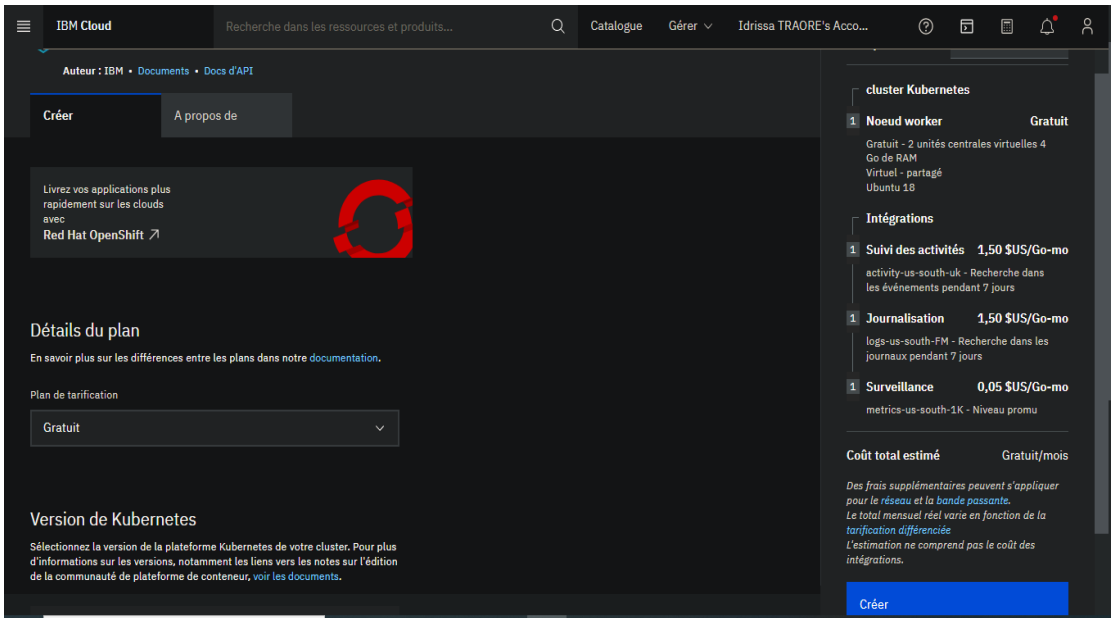
1. After being logged in my IBM account, I click on list of resource



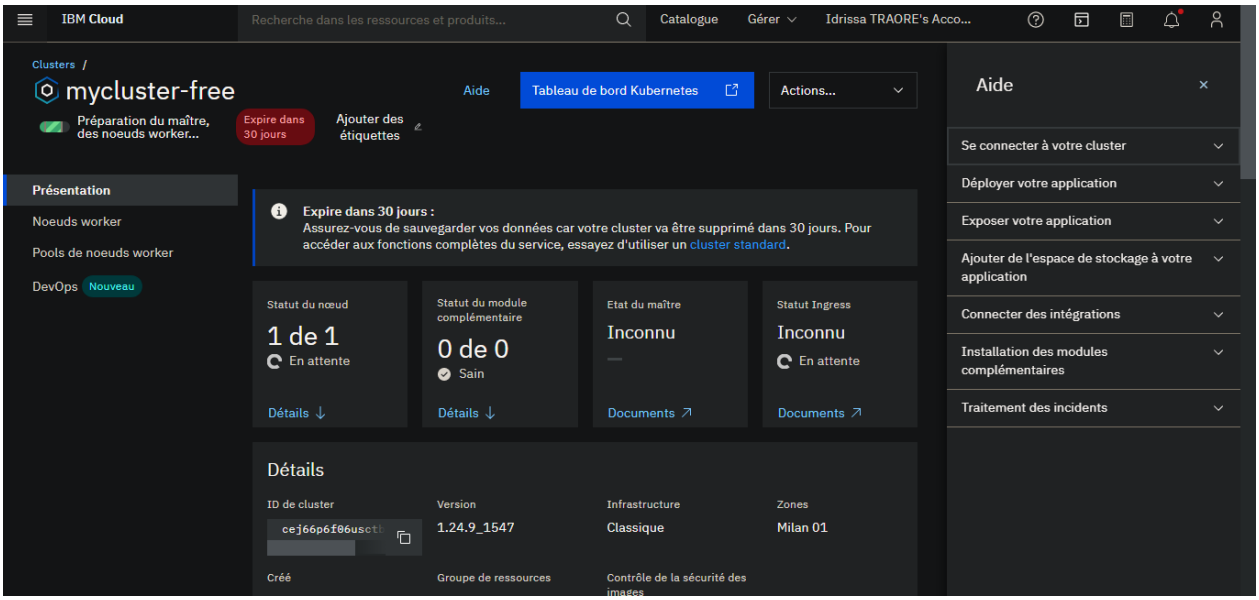
2. I search the catalog for “Kubernetes”, then click Kubernetes Service



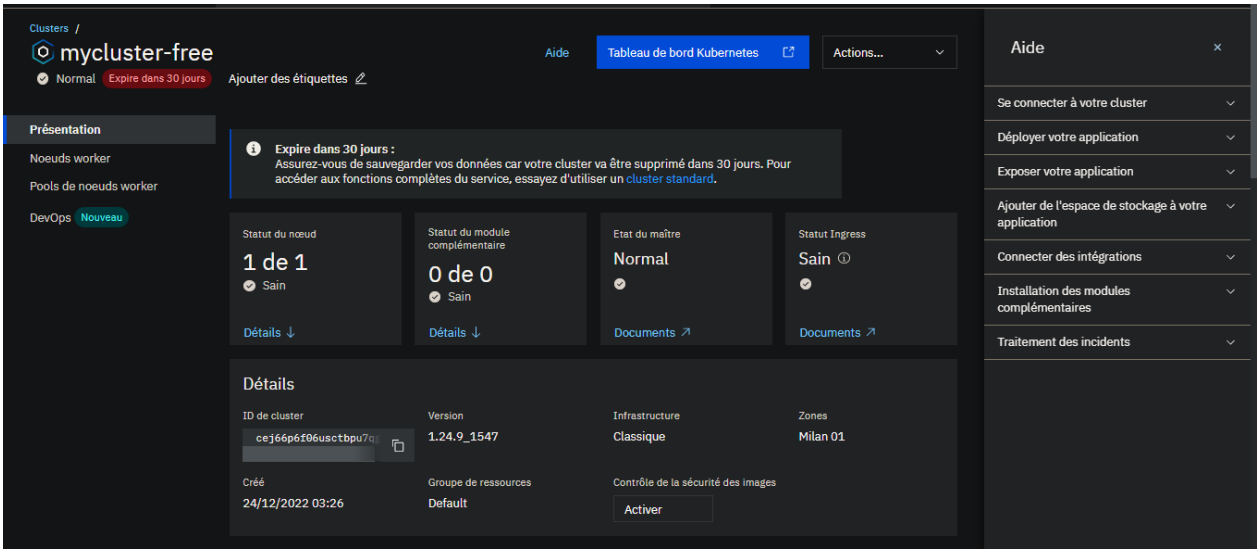
3. I select the Free pricing plan and leave the default values for the other fields, then click Create



4. I wait until the cluster is deployed, that is, cluster status is Normal

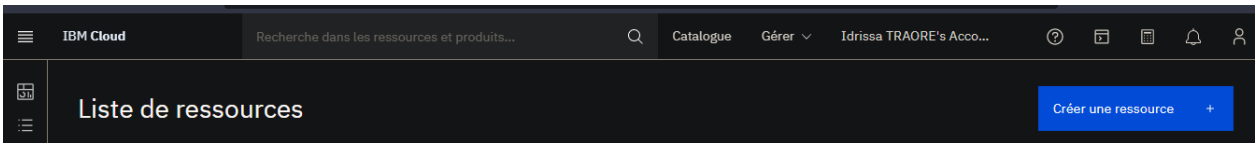


Finally, it is, now

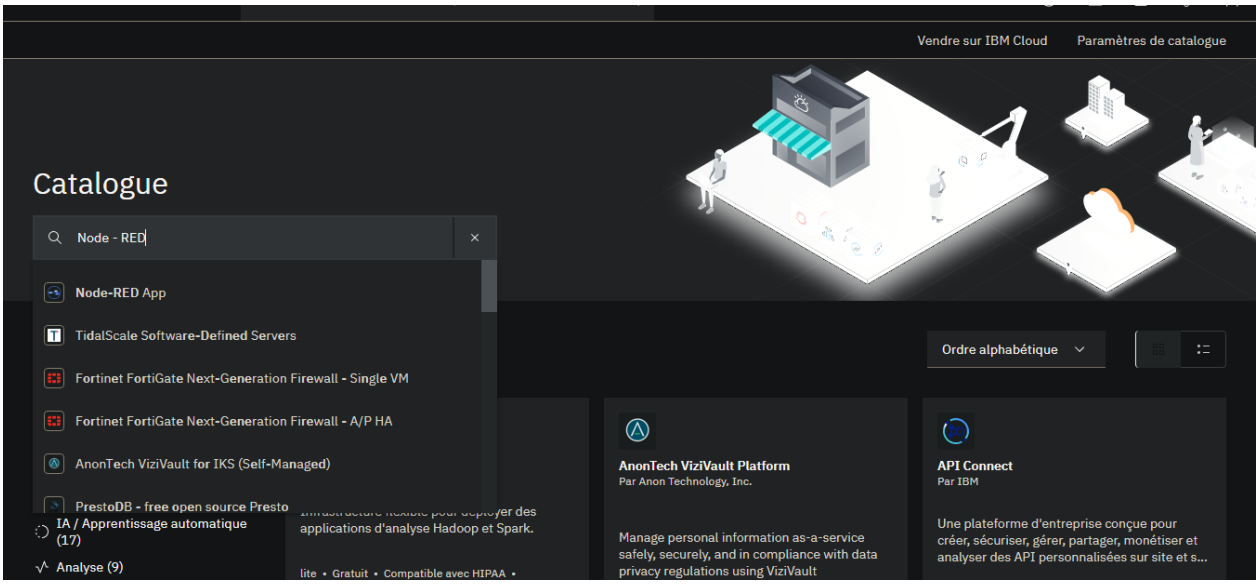


Create and deploy the Node-RED app.

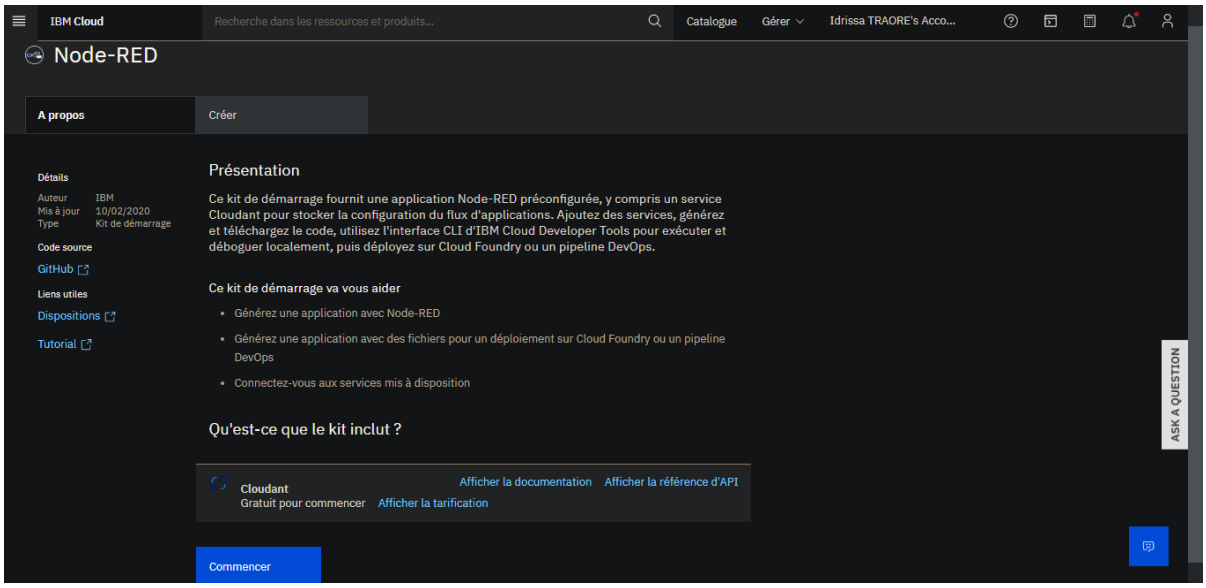
1. I click on create a resource again

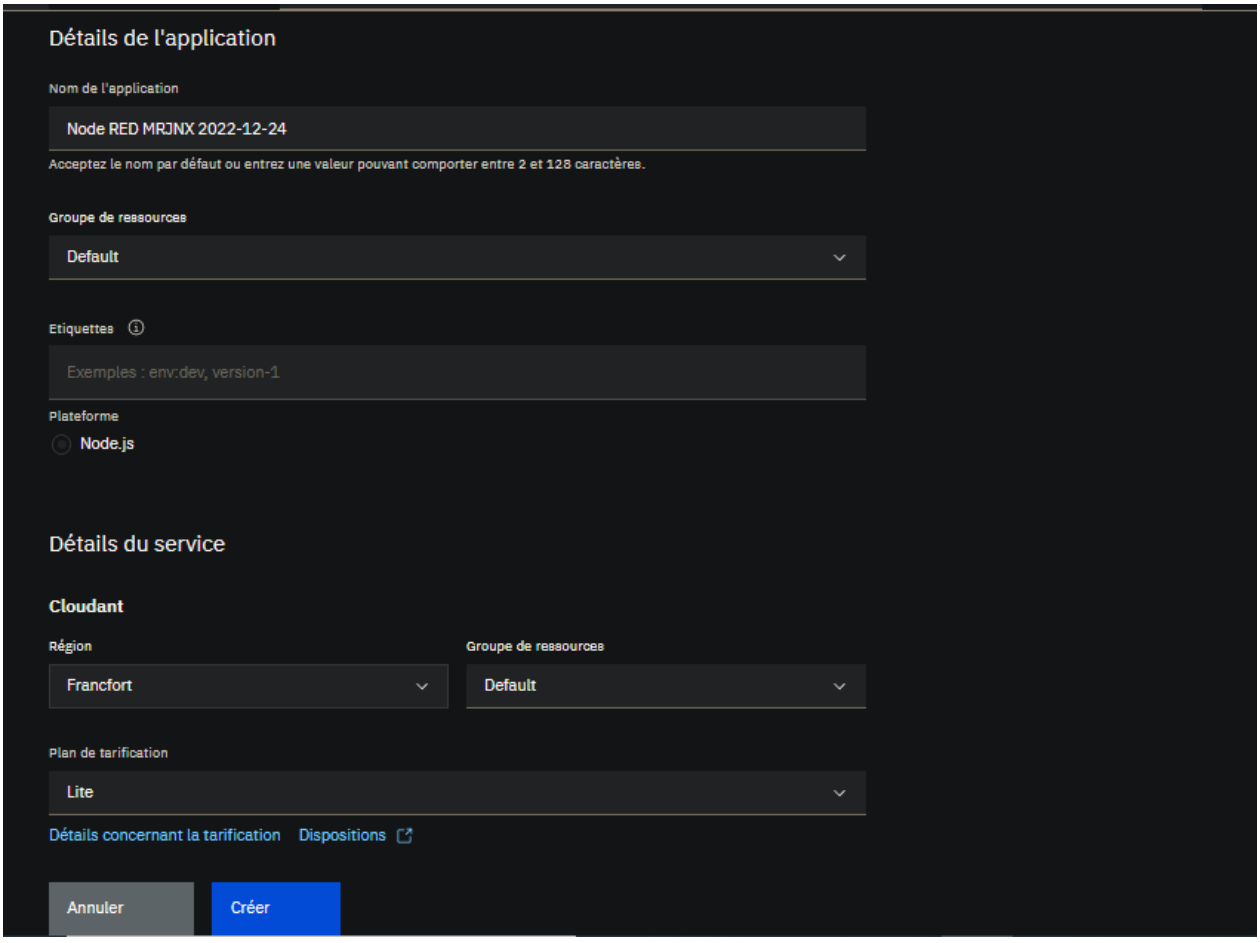


2. I search the catalog for “Node-RED”, then click Node-RED App.

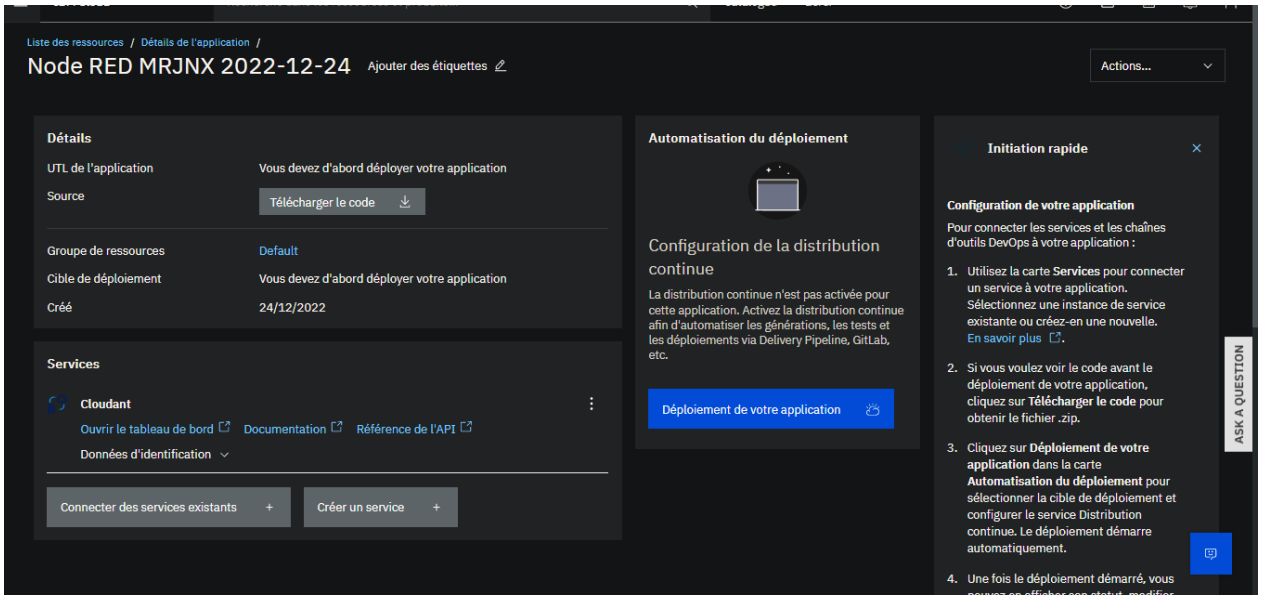


3. I click the Create tab, leave the default values for all fields, and then click Create.

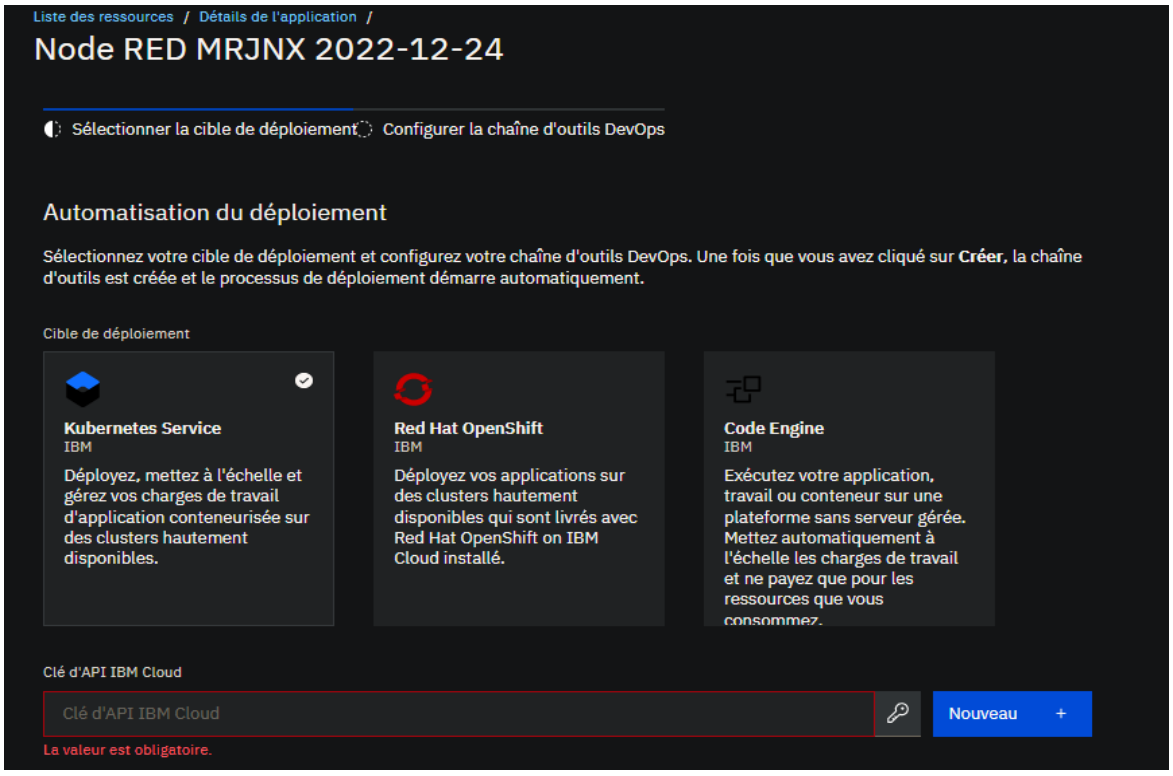

IBM Cloud
Recherche dans les ressources et produits...
Catalogue
Gérer
Idrissa TRAORE's Acco...
Node-RED
A propos
Créer
Détails
Auteur IBM
Mis à jour 10/02/2020
Type Kit de démarrage
Code source
GitHub
Liens utiles
Dispositions
Tutorial
Présentation
Ce kit de démarrage fournit une application Node-RED préconfigurée, y compris un service Cloudant pour stocker la configuration du flux d'applications. Ajoutez des services, générez et téléchargez le code, utilisez l'interface CLI d'IBM Cloud Developer Tools pour exécuter et déboguer localement, puis déployez sur Cloud Foundry ou un pipeline DevOps.
Ce kit de démarrage va vous aider
Générez une application avec Node-RED
Générez une application avec des fichiers pour un déploiement sur Cloud Foundry ou un pipeline DevOps
Connectez-vous aux services mis à disposition
Qu'est-ce que le kit inclut ?
Cloudant
Gratuit pour commencer
Afficher la documentation
Afficher la référence d'API
Afficher la tarification
Commencer
ASK A QUESTION


Détails de l'application
Nom de l'application
Node RED MRJNX 2022-12-24
Acceptez le nom par défaut ou entrez une valeur pouvant comporter entre 2 et 128 caractères.
Groupe de ressources
Default
Etiquettes
Exemples : env:dev, version-1
Plateforme
Node.js
Détails du service
Cloudant
Région
Francfort
Groupe de ressources
Default
Plan de tarification
Lite
Détails concernant la tarification
Dispositions
Annuler
Créer

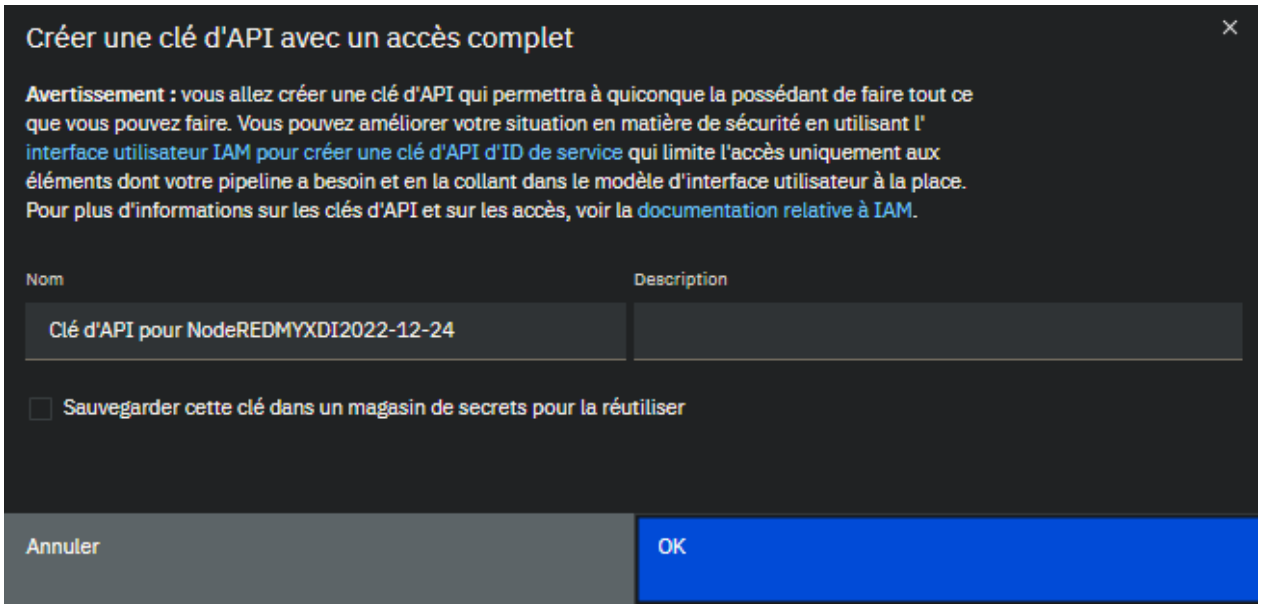
4. The Node-RED App administration page opens. So I Click on Deploy your app.



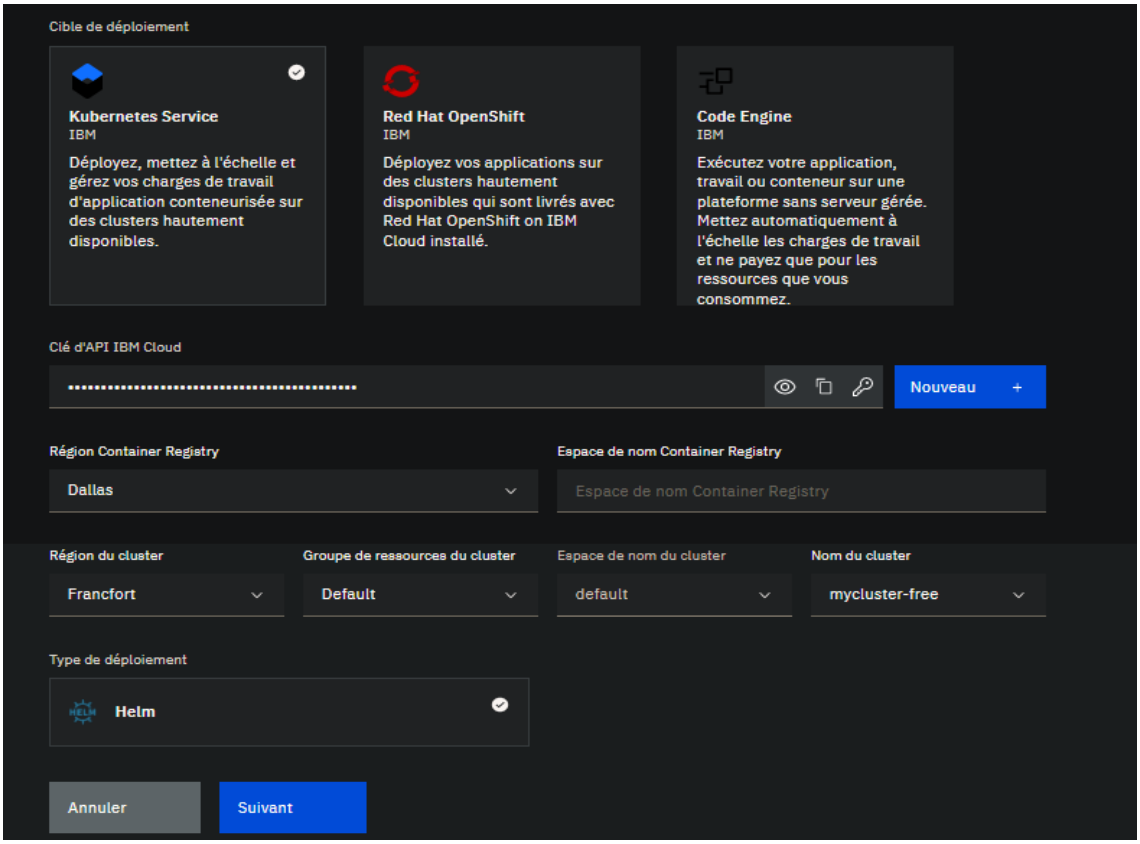
5. I select Kubernetes Service for deployment target.



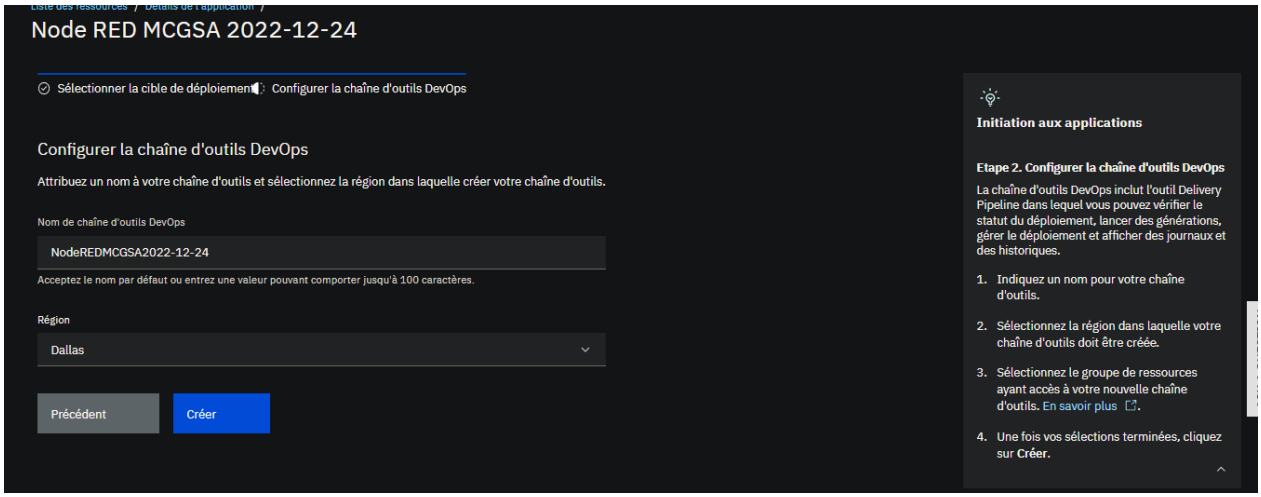
6. Click New + to generate an IBM Cloud API key, then click OK.



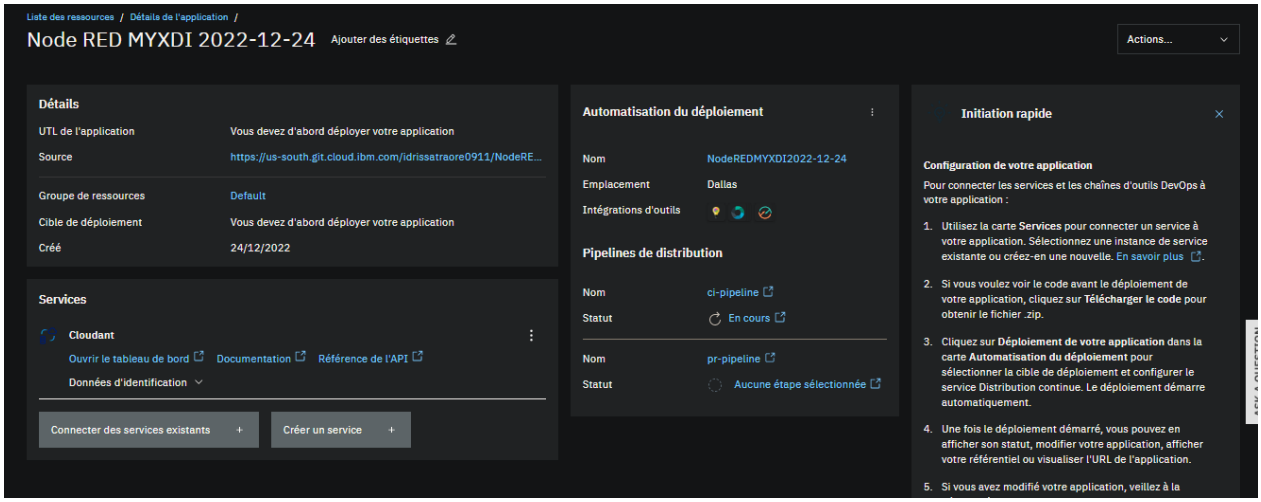
7. Leave the default values and then click Next.



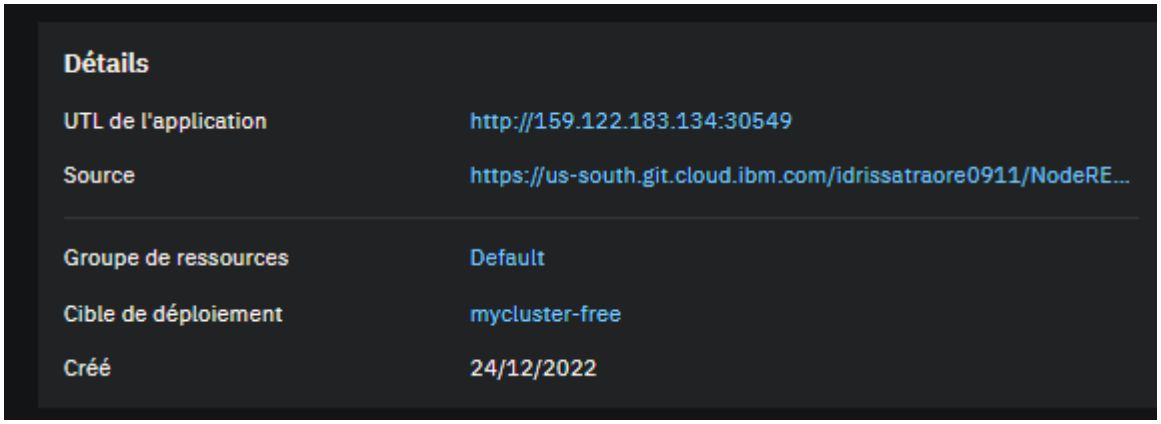
8. I leave the default values and click Create



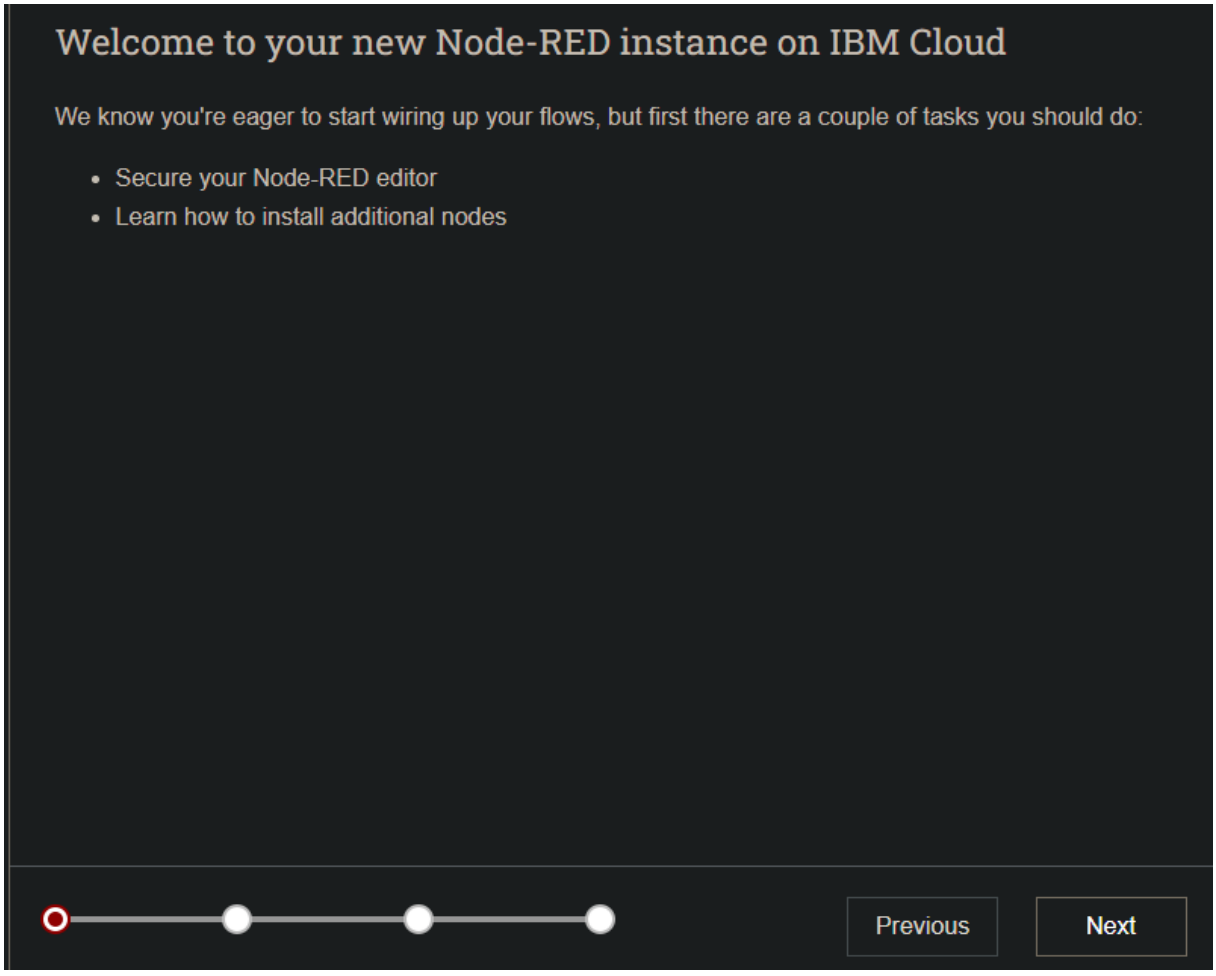
9. I'm waiting for the URL



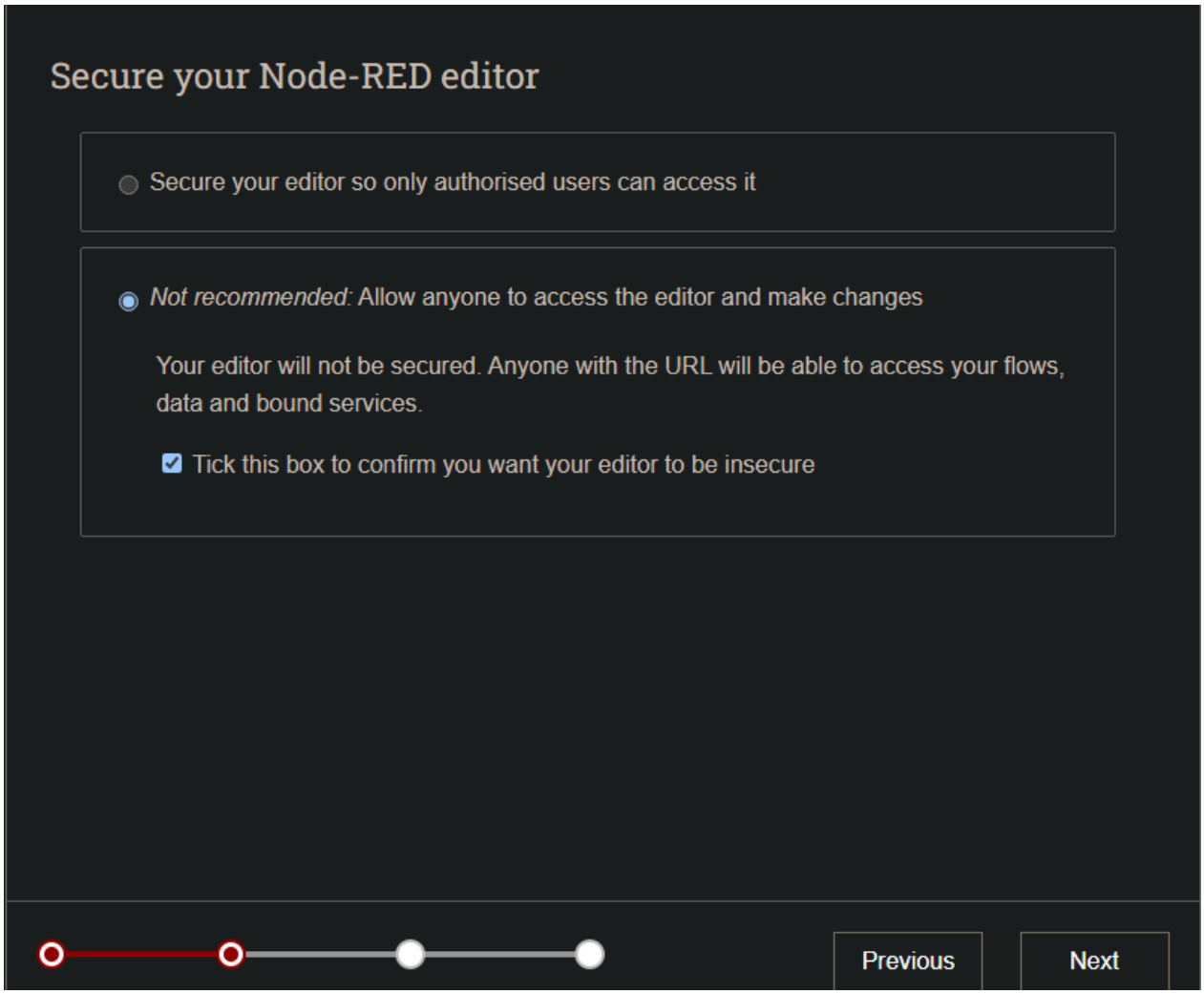
Here it is



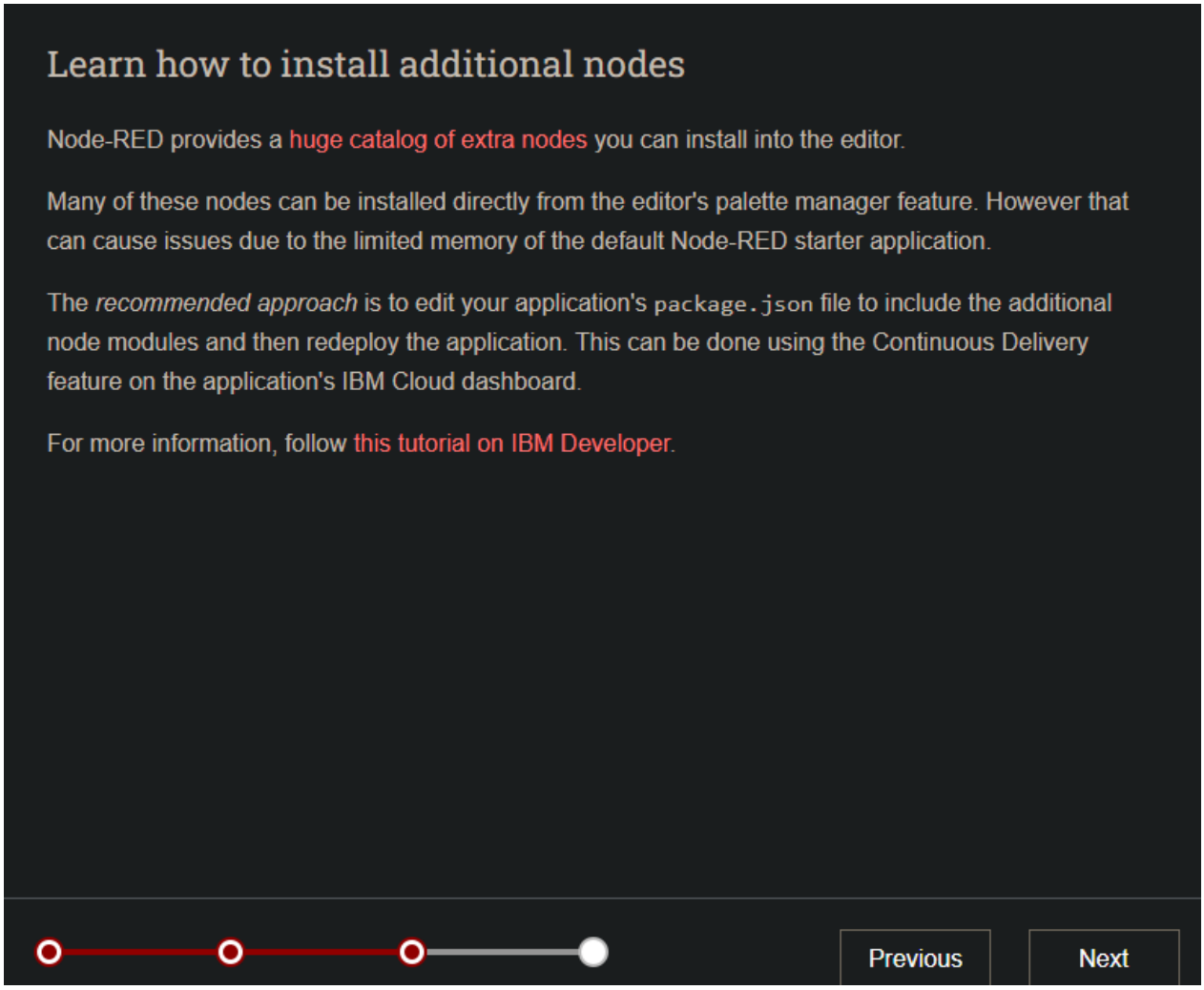
10. To start the Node-RED app configuration, click App URL. The welcome page is displayed. Click Next.



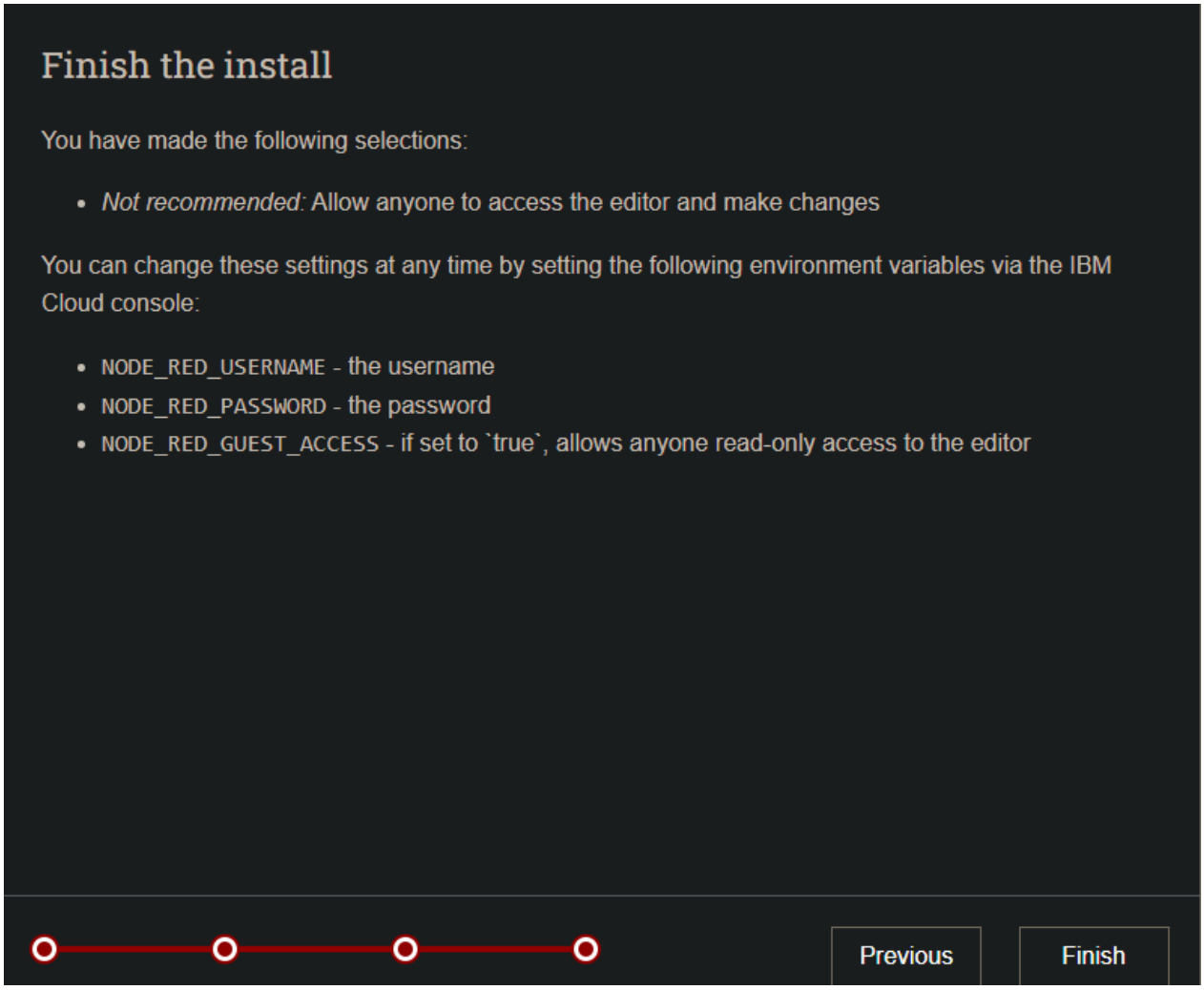
11. I am prompted to secure my application. In this lab, I do not secure the Node-RED editor for easy access. I Click Not recommended: I allow anyone to access the editor and make changes, then check Tick this box to confirm I want my editor to be insecure and I click Next.



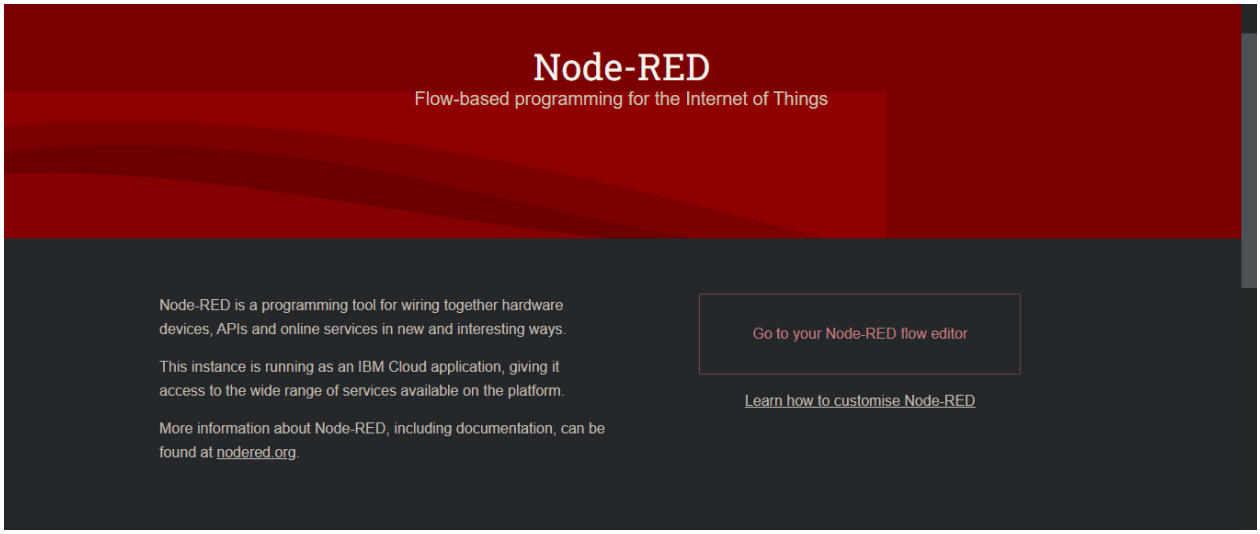
Next steps:



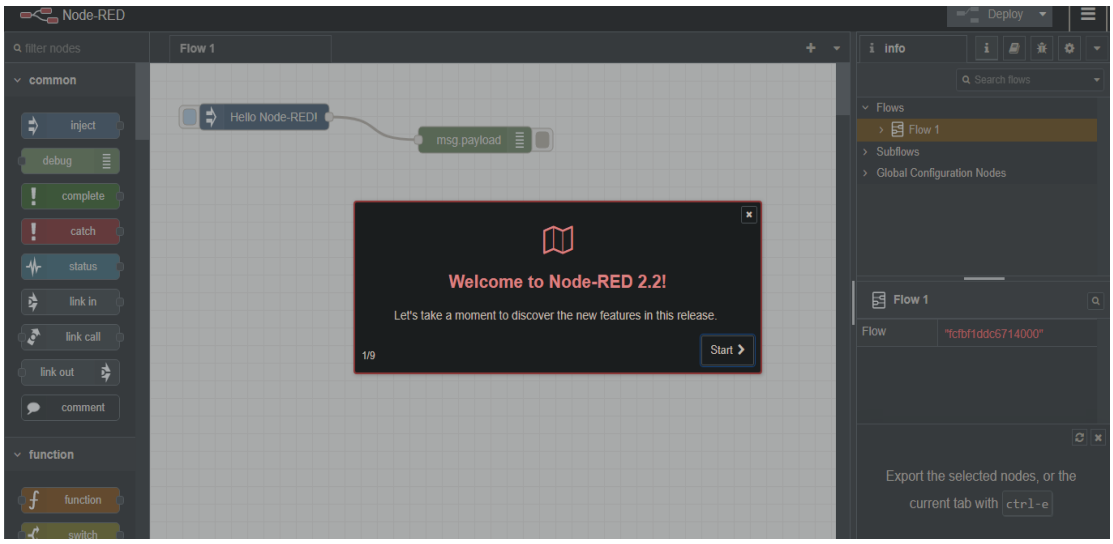
12. Click Next, then Finish. The page that is shown in the following figure is displayed.

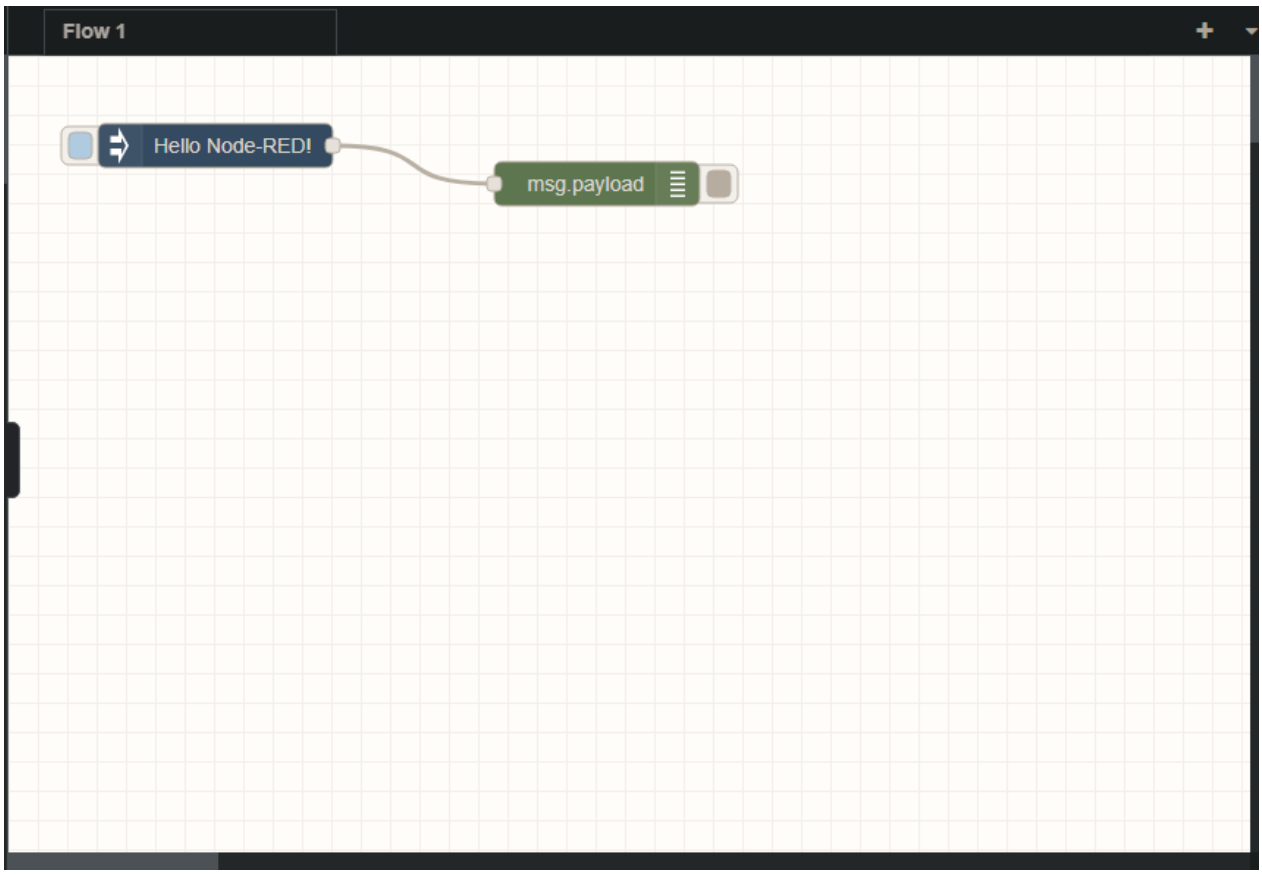


This is what it shows:



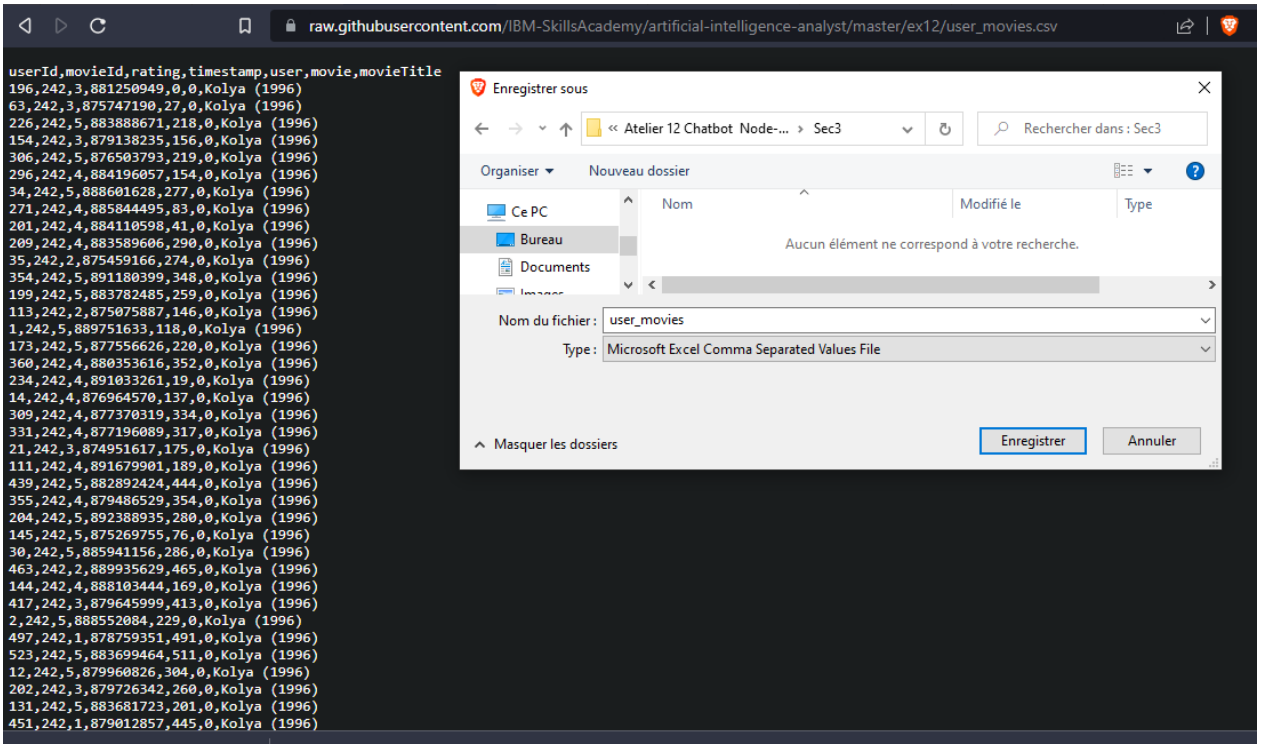
13. Click Go to your Node-RED flow editor and the Node-RED Flow editor is displayed as shown in the following figure.





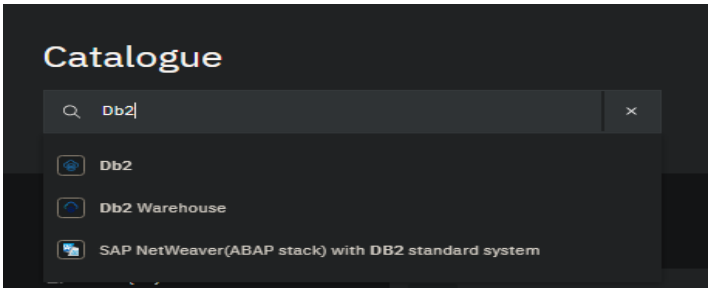
Create a Db2 service and load movies data.

- I download the file user_movies.csv from github

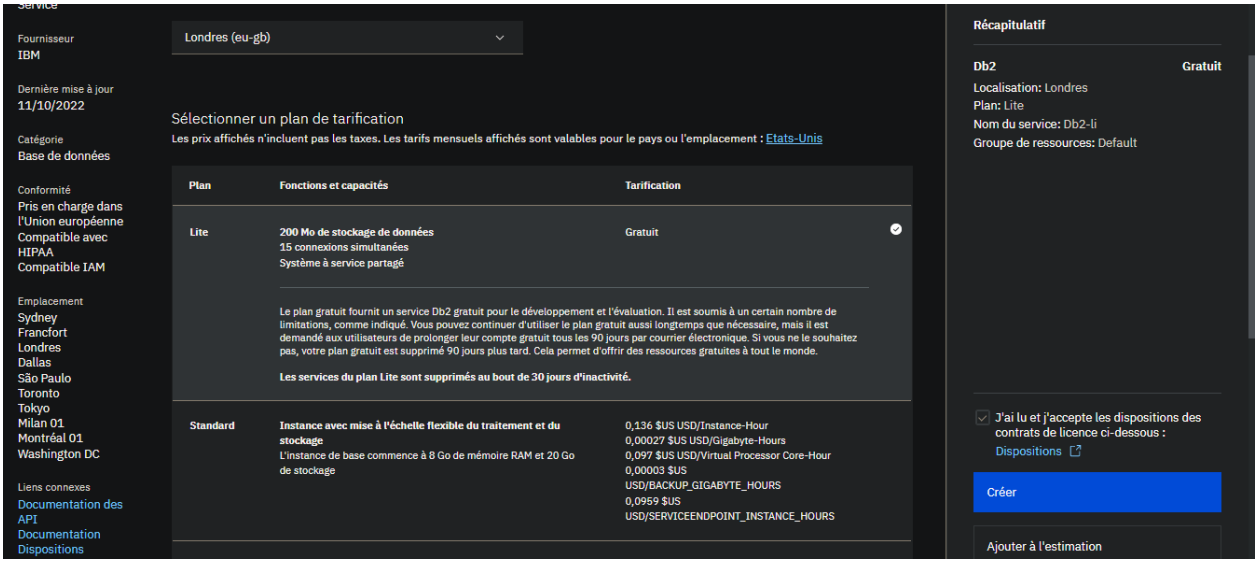


- In the IBM Cloud Dashboard, I click Create resource +

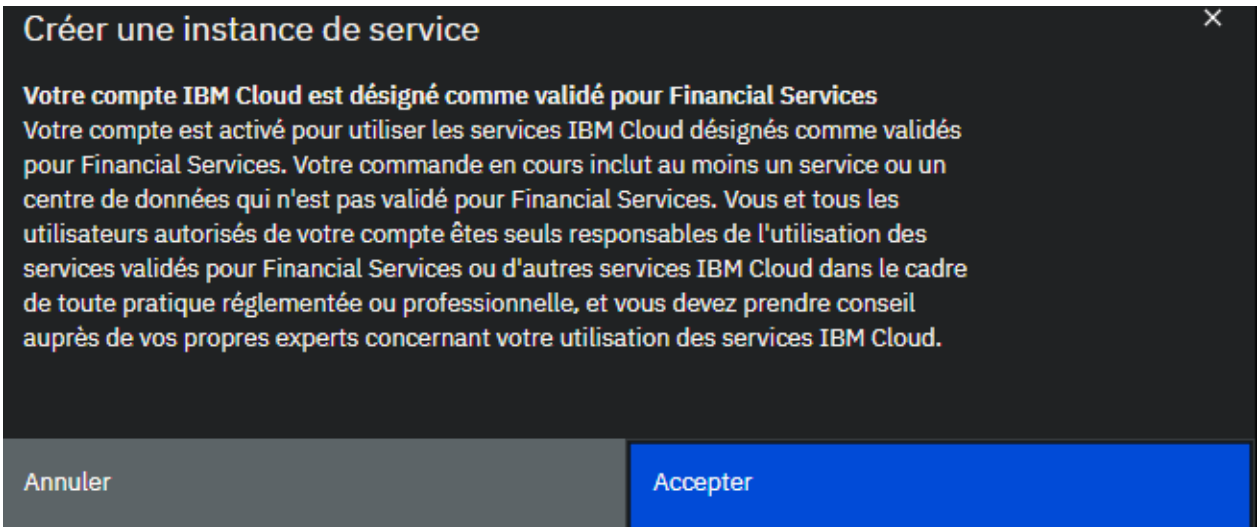
- Search the catalog for “Db2”, then click Db2.



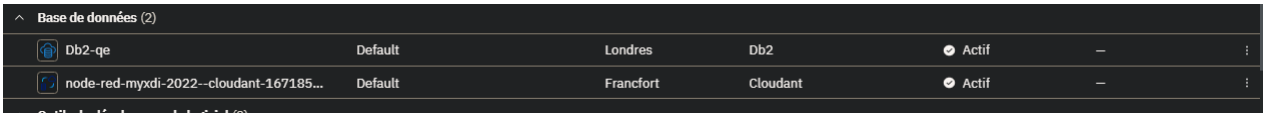
- Ensure that the Lite plan is selected, leave the default values, accept the license agreement, and then click Create.



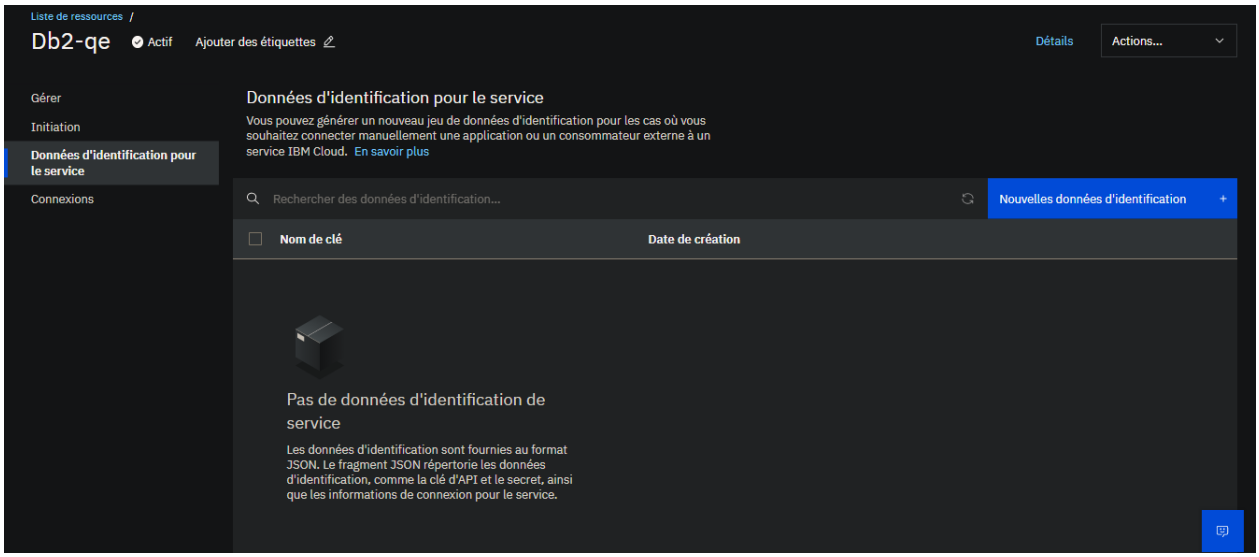
Then, I saw this message and I clicked on “accept”



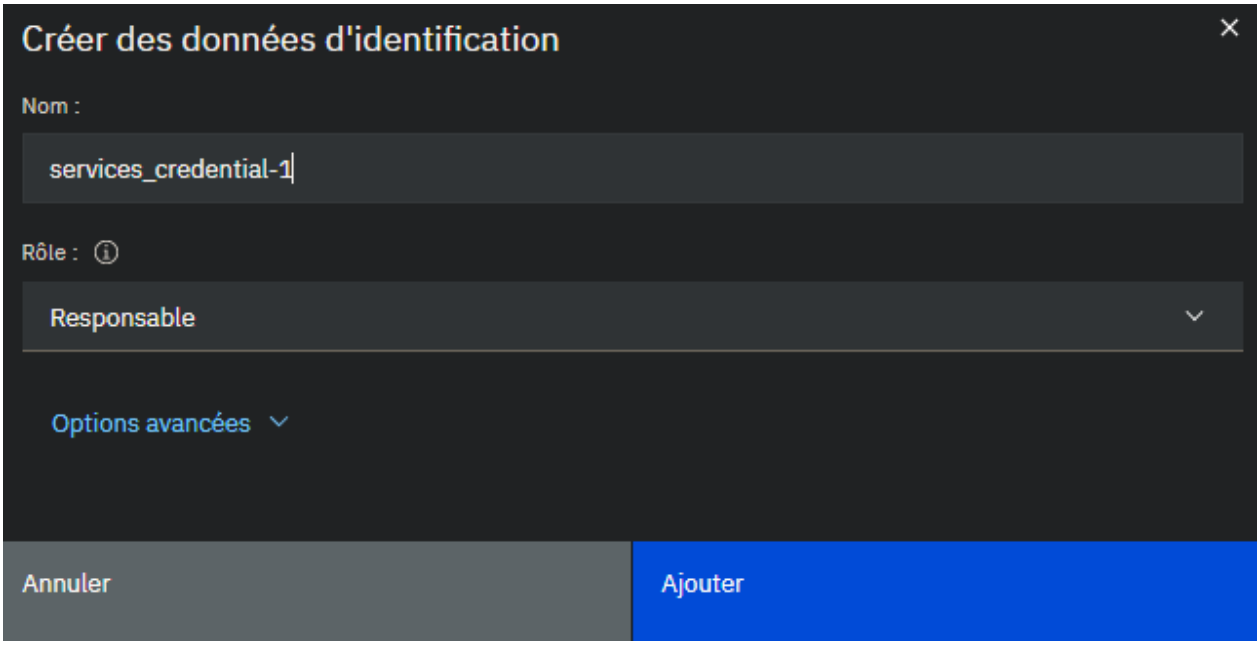
- I am redirected to the Resource list page. So, I wait until the Db2 service status is Active and then clicked your Db2 service.



- Click Service credentials and then click new credential +.



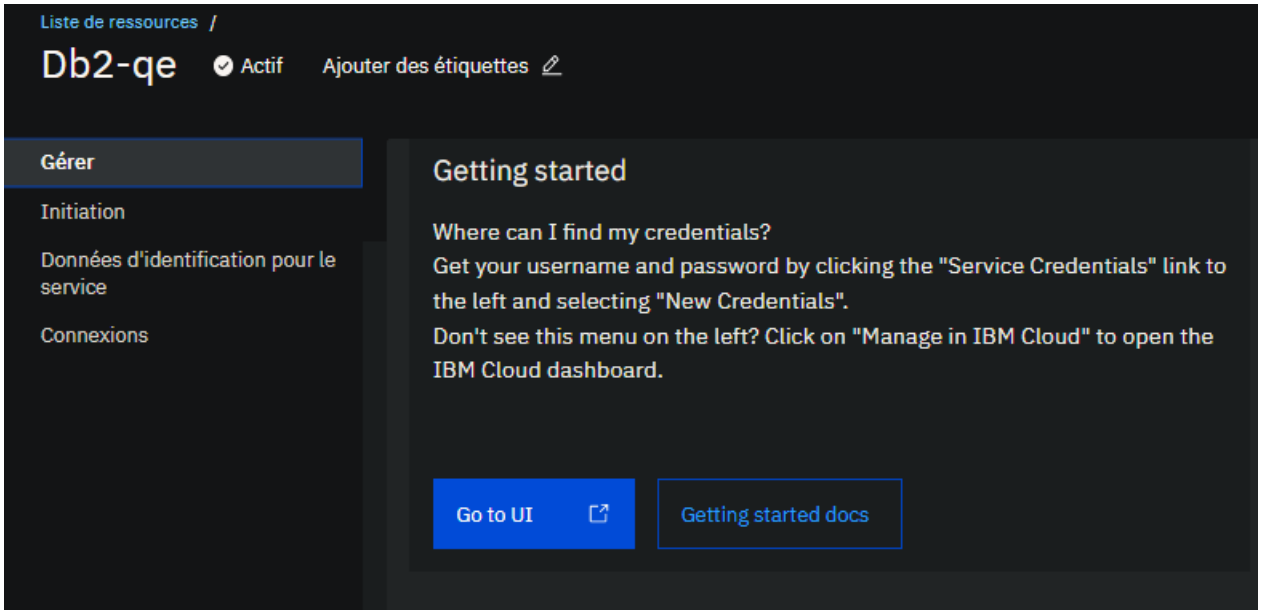
- I clicked on Add.



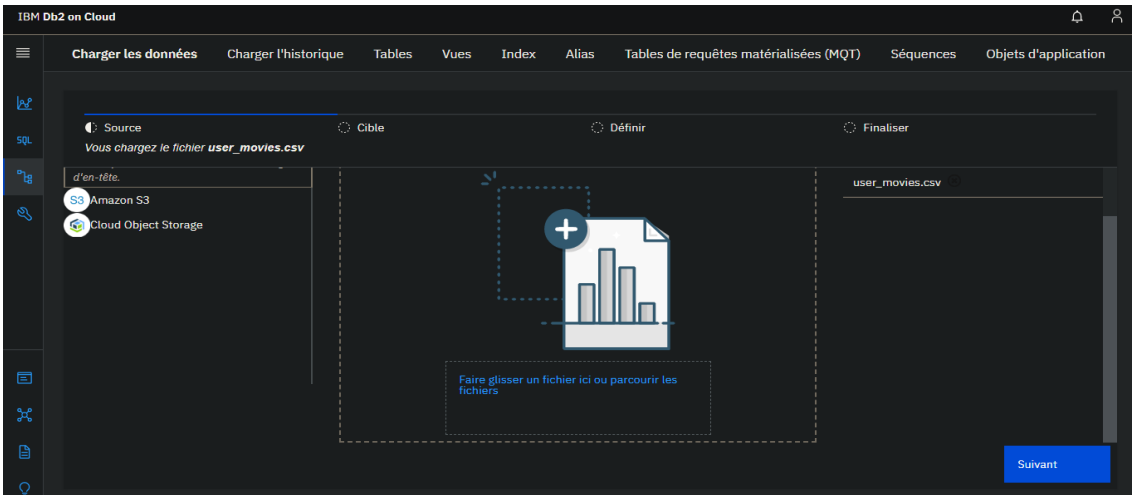
- Expand the service credentials. Copy the following parameters; you use them in “Part 6: Integrating the solution components by using Node-RED”:

```
        "name": "1dd14d0c-1b52-4f63-a606-53ecba28771d"
      },
      "composed": [
        "db2 -u xdq97333 -p d61gs1Q8Vi6mIXxW --ssl --sslCAFile 1dd14d0c-1b52-4f63-a606-53ecba28771d --authenticationDatabase admin --host 815fa4db-dc03-4c70-869a-a9cc13f33084.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:30367"
      ],
      "environment": {},
      "type": "cli"
    },
    "db2": {
      "authentication": {
        "method": "direct",
        "password": "d61gs1Q8Vi6mIXxW",
        "username": "xdq97333"
      },
      "certificate": {
```


- Click Manage, then click Go to UI and start importing movies data.

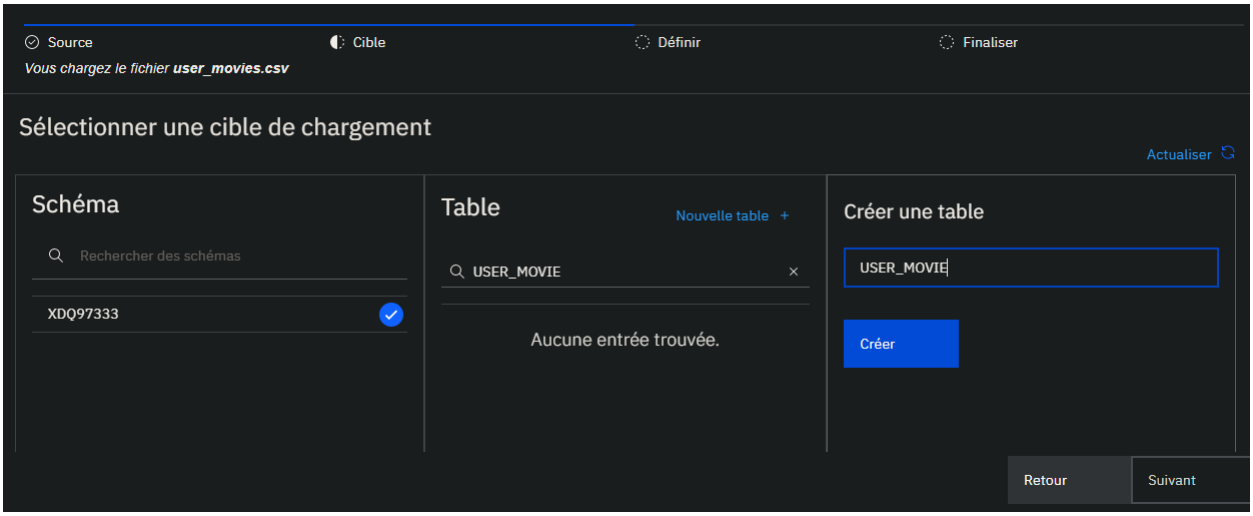


- Click Data from the left menu.

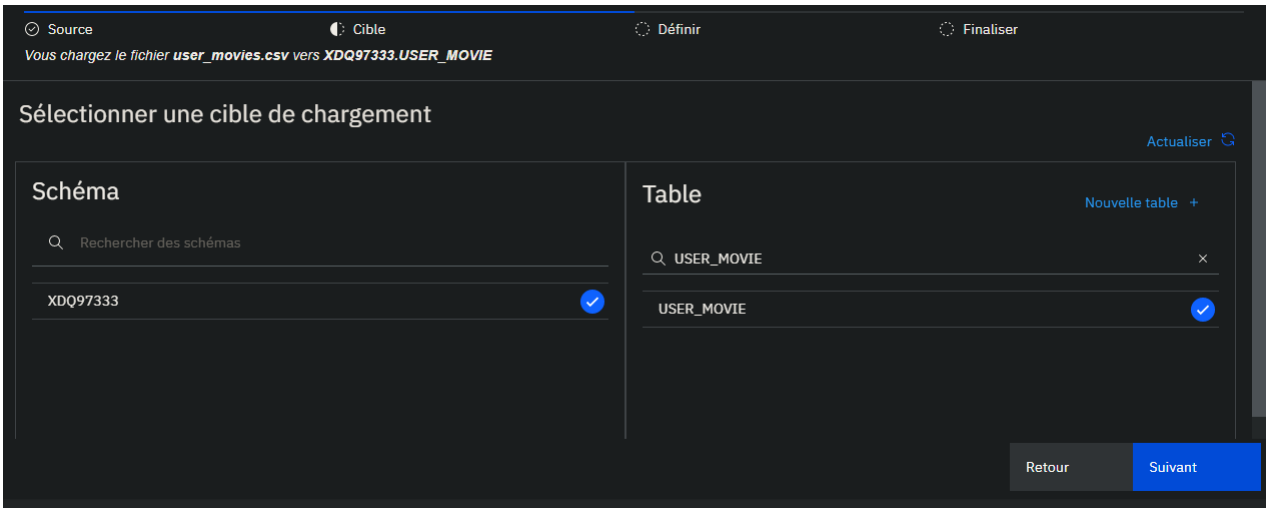


- I click browse files, then select the user_movies.csv file that I downloaded previously from https://github.com/IBM-SkillsAcademy/artificial-intelligence-analyst/blob/master/ex12/user_movies.csv

  Therefore, I select my default schema, then click new table + and name the table USER_MOVIE, then click Create. I take note of the schema name; I use it in “6: Integrating the solution components by using Node-RED”.

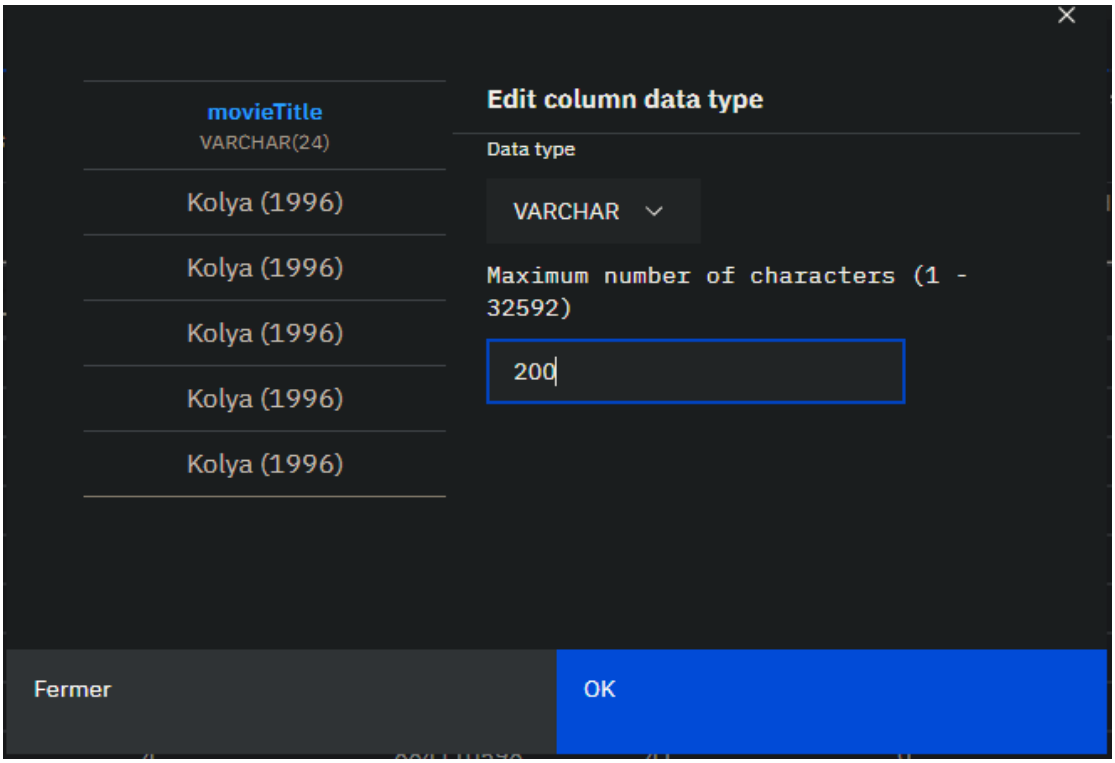


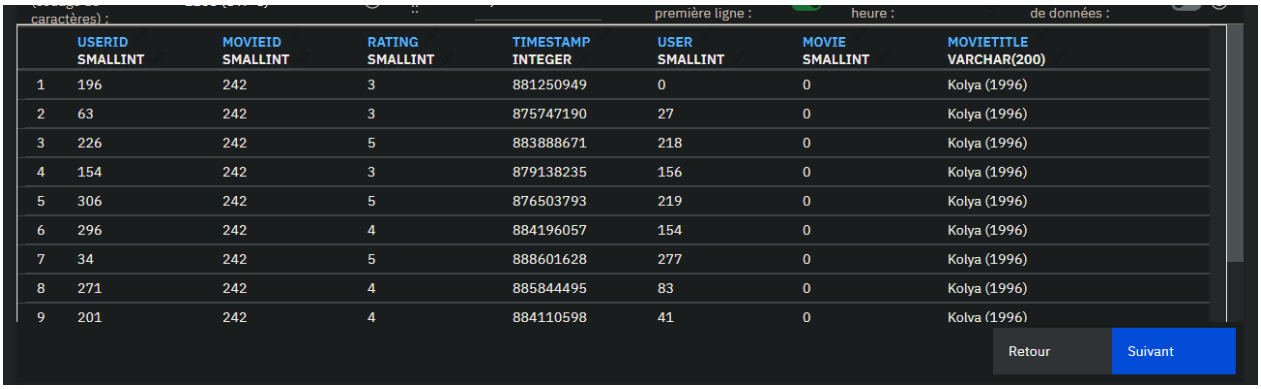
- Click Next to define the table.



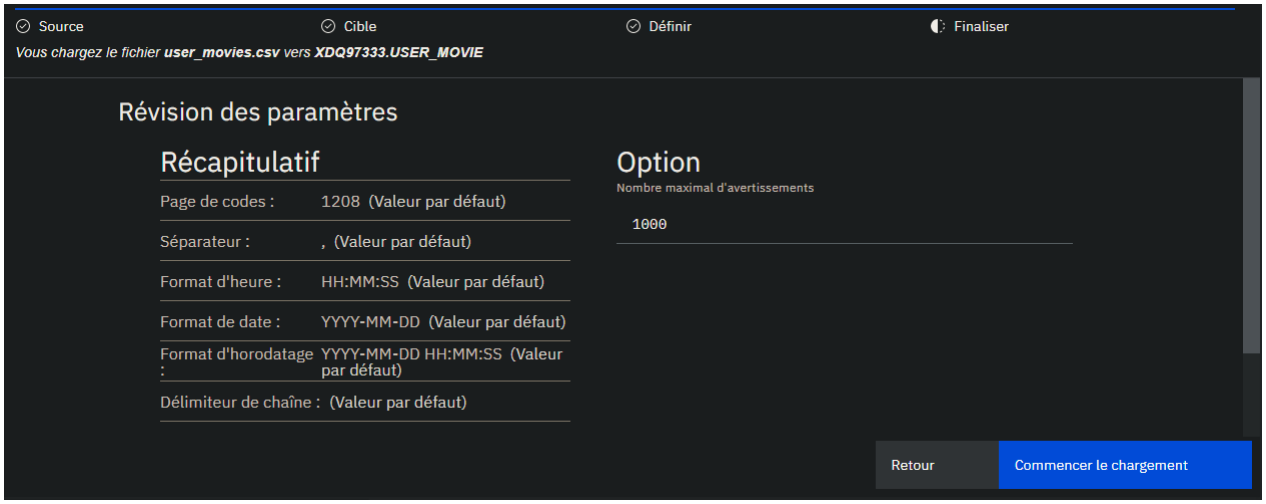
- Change the type of MOVIETITLE column to VARCHAR(200) by clicking the pencil icon next to VARCHAR(24).

  I Change Maximum number of characters to 200. I Click OK and then Next.

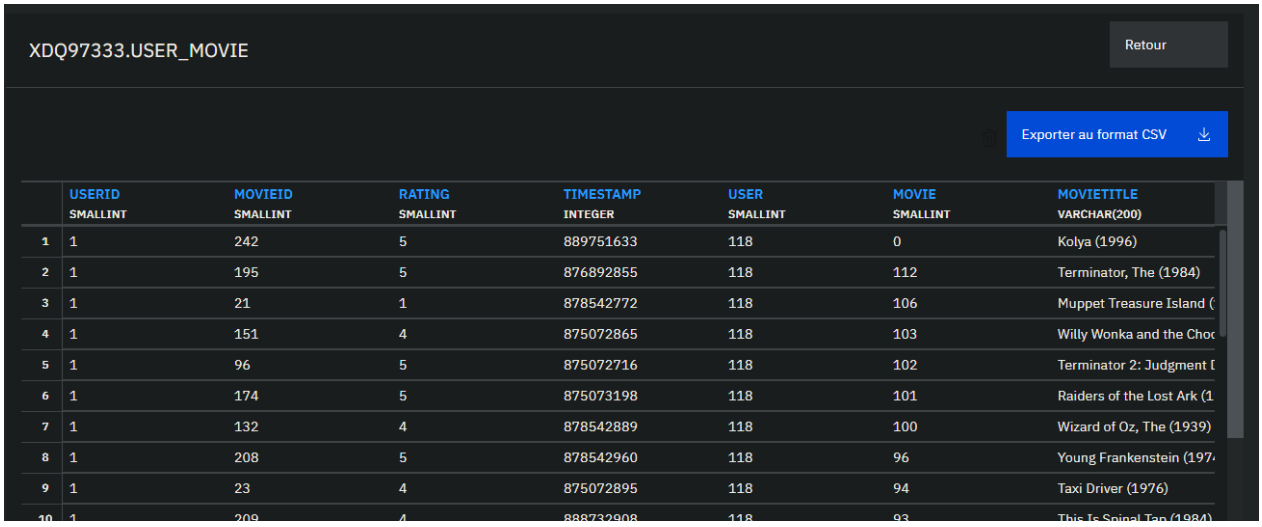




- The Review settings page is displayed. Click Begin Load.

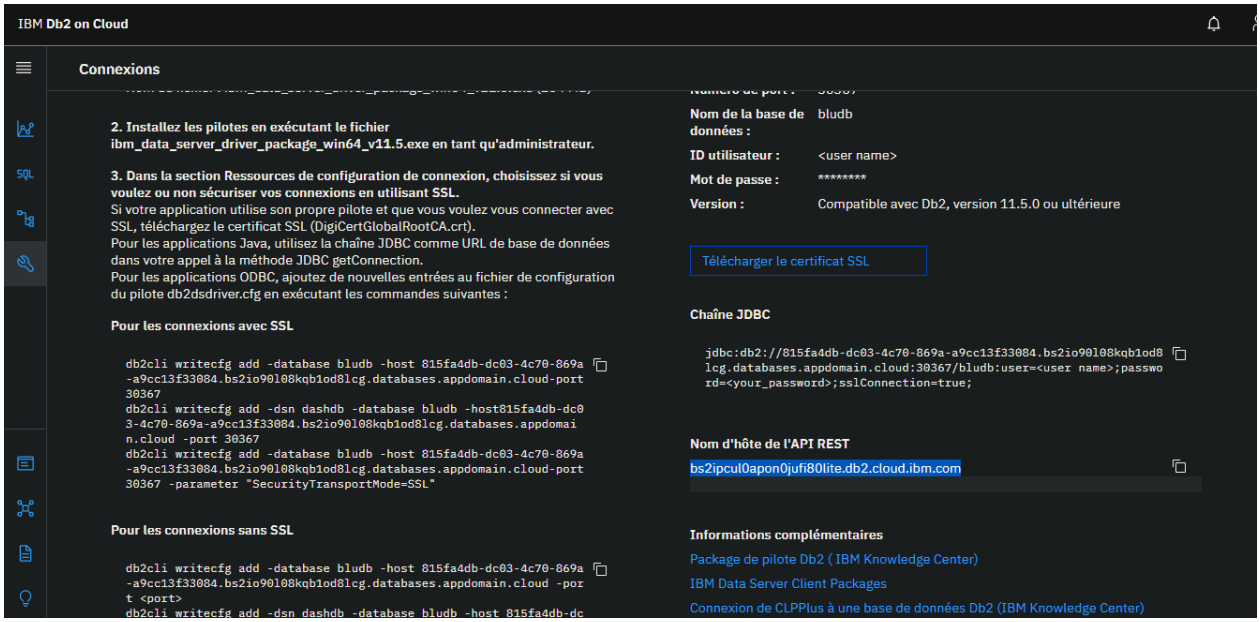


- Wait until the data is loaded successfully. Confirm that no errors occurred while the data was loaded then I click View table. The table looks like the following figure.



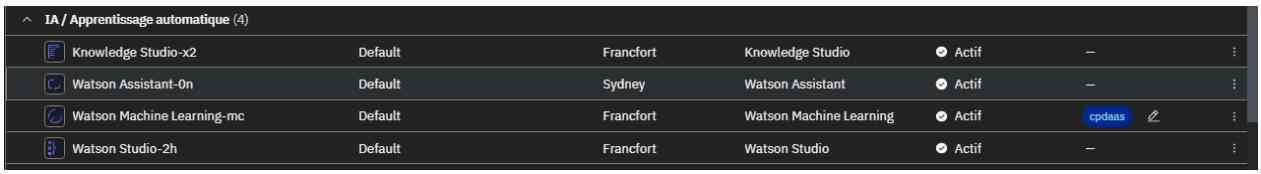
	USERID SMALLINT	MOVIEID SMALLINT	RATING SMALLINT	TIMESTAMP INTEGER	USER SMALLINT	MOVIE SMALLINT	MOVIETITLE VARCHAR(200)
1	1	242	5	889751633	118	0	Kolya (1996)
2	1	195	5	876892855	118	112	Terminator, The (1984)
3	1	21	1	878542772	118	106	Muppet Treasure Island (
4	1	151	4	875072865	118	103	Willy Wonka and the Choc
5	1	96	5	875072716	118	102	Terminator 2: Judgment D
6	1	174	5	875073198	118	101	Raiders of the Lost Ark (1
7	1	132	4	878542889	118	100	Wizard of Oz, The (1939)
8	1	208	5	878542960	118	96	Young Frankenstein (197
9	1	23	4	875072895	118	94	Taxi Driver (1976)
10	1	209	4	888732908	118	93	This Is Spinal Tap (1984)

- Click the Administration icon from the left menu. Copy the REST API host name. You use it in “ 6: Integrating the solution components by using Node-RED.”

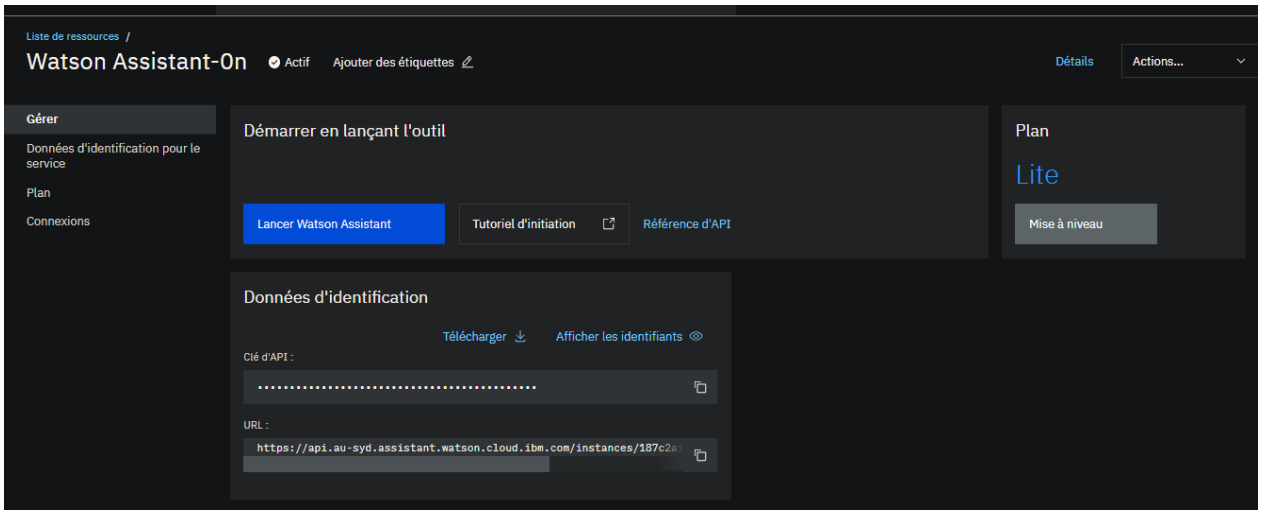


Create a Watson Assistant and configure it.

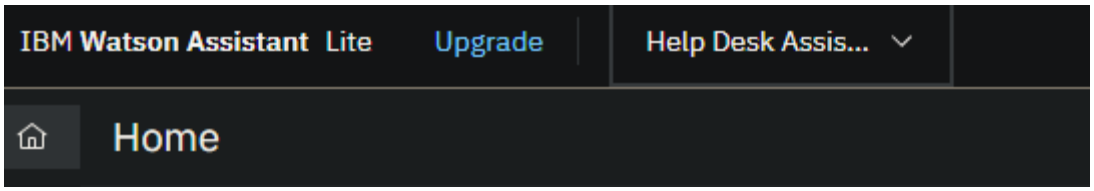
- From the IBM Cloud Dashboard, click Resource list.



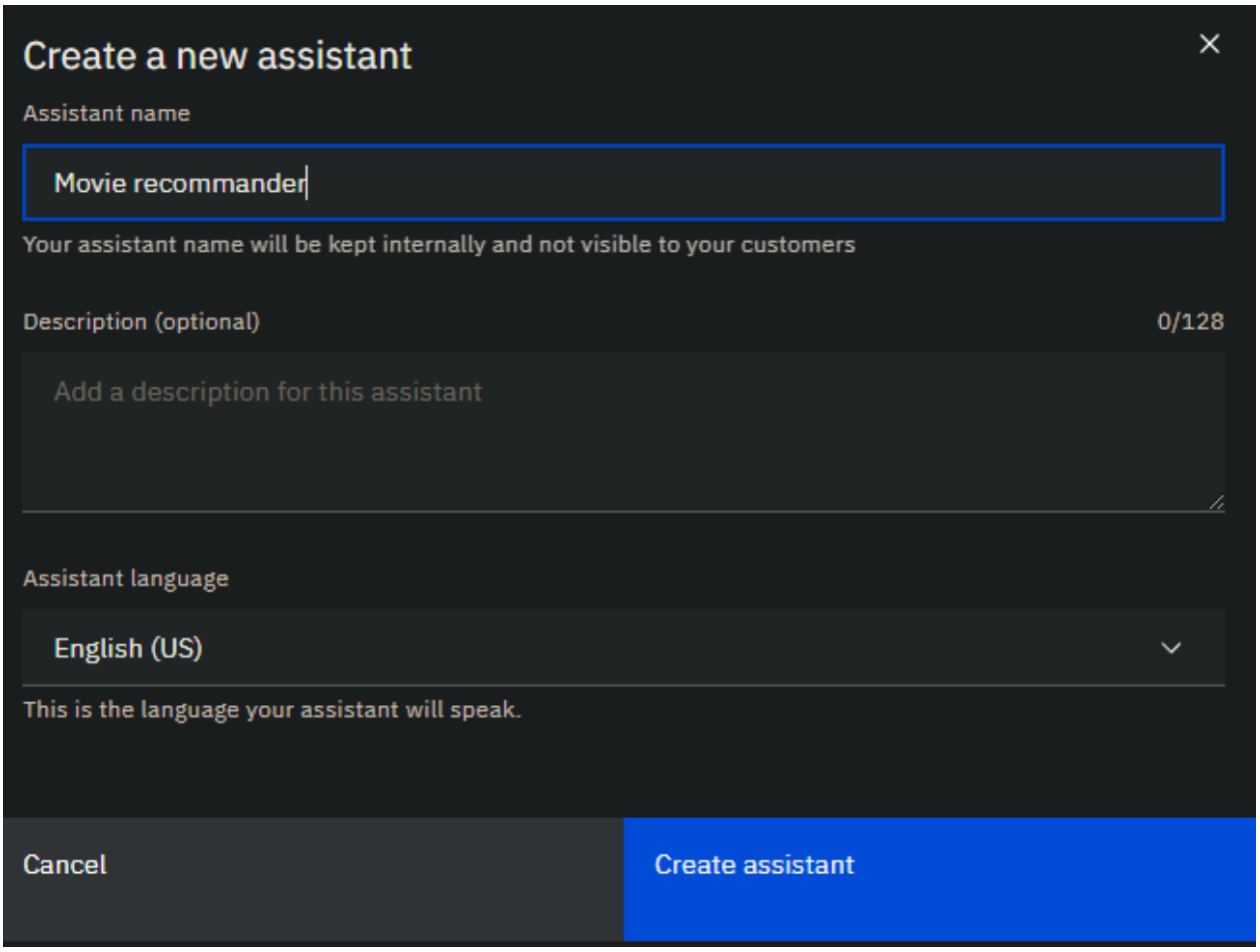
- Click Launch Watson Assistant.



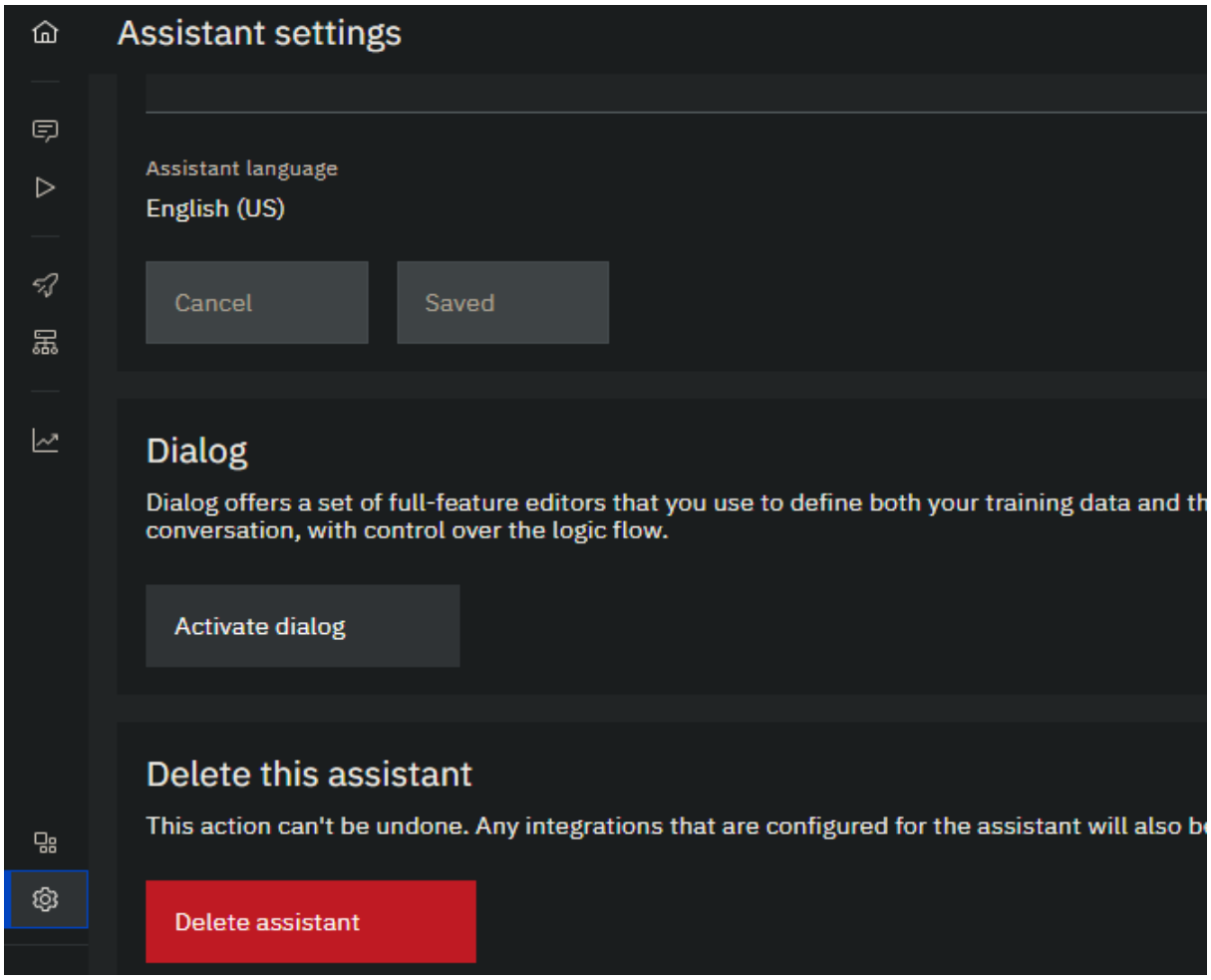
- From the menu at the top, click the assistants' list, then click Create New +.



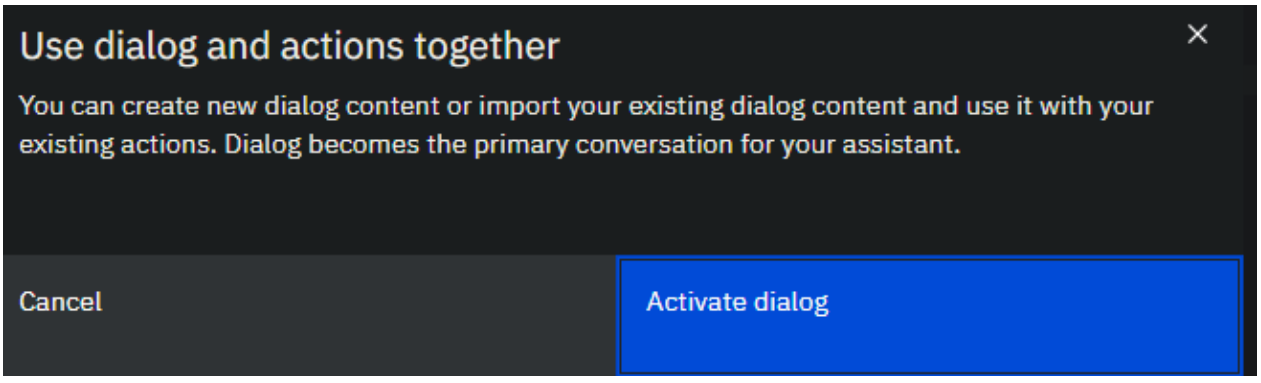
- Name the assistant Movie recommender, then click Create Assistant.



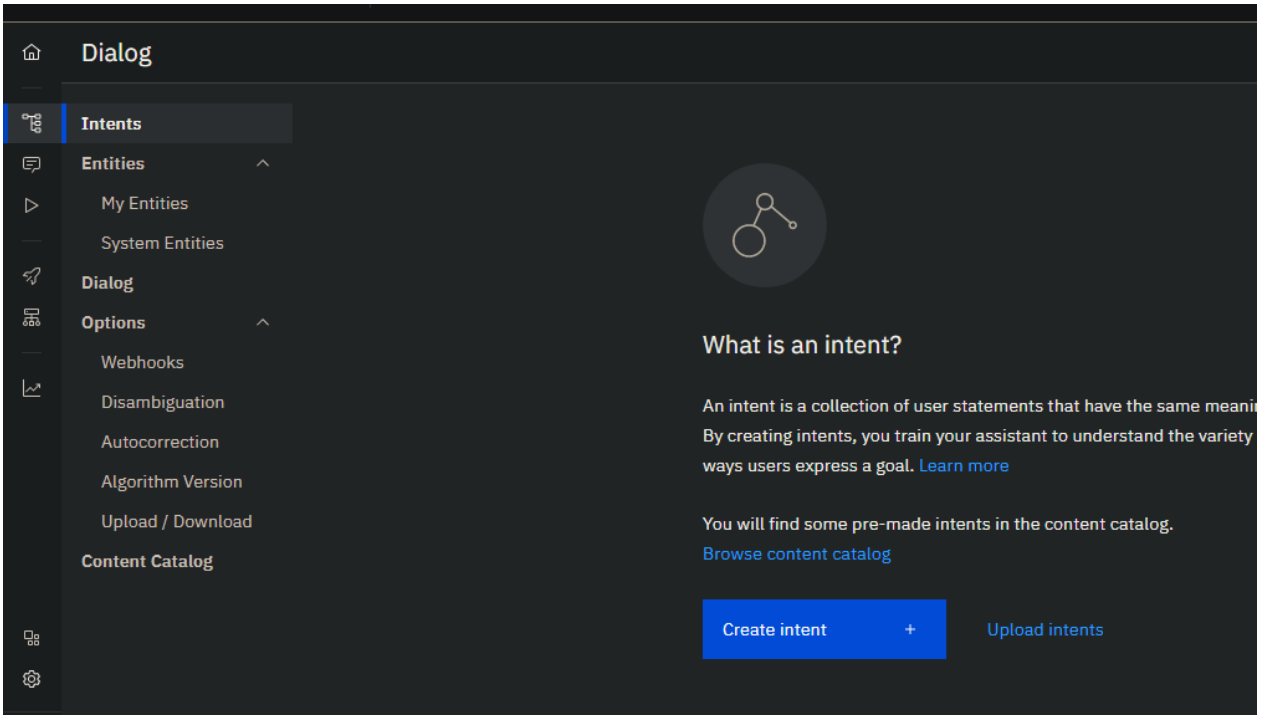
In the Assistant settings page, click Activate dialog.



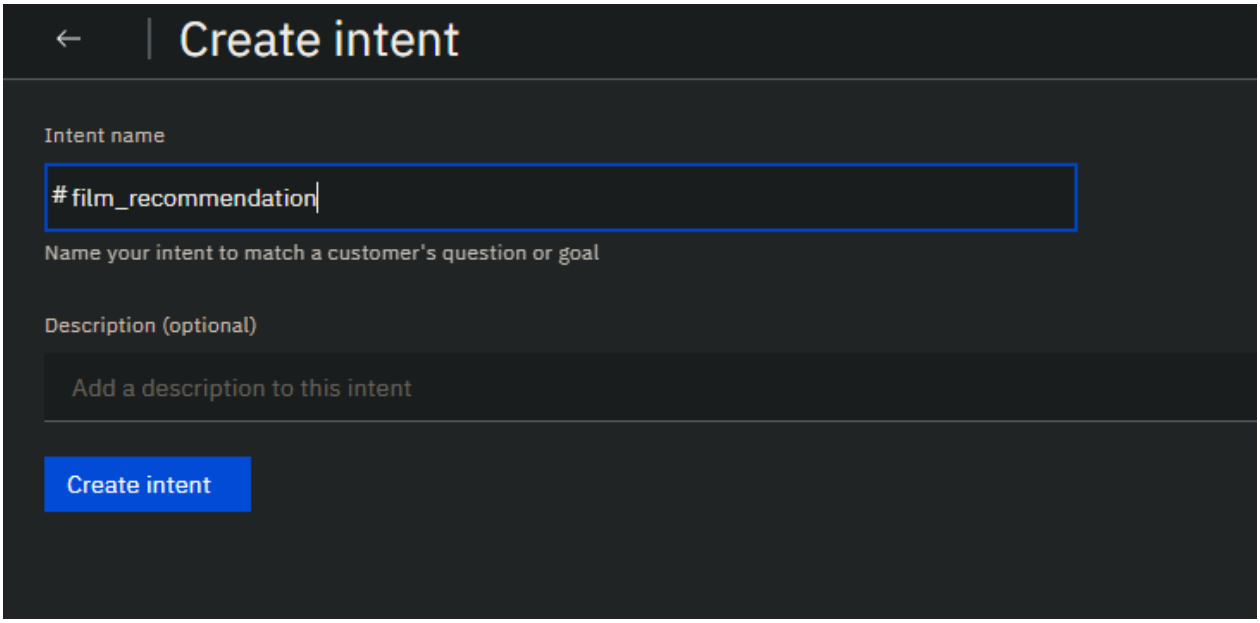
5. At the window “Use dialog and actions together” click Activate dialog.



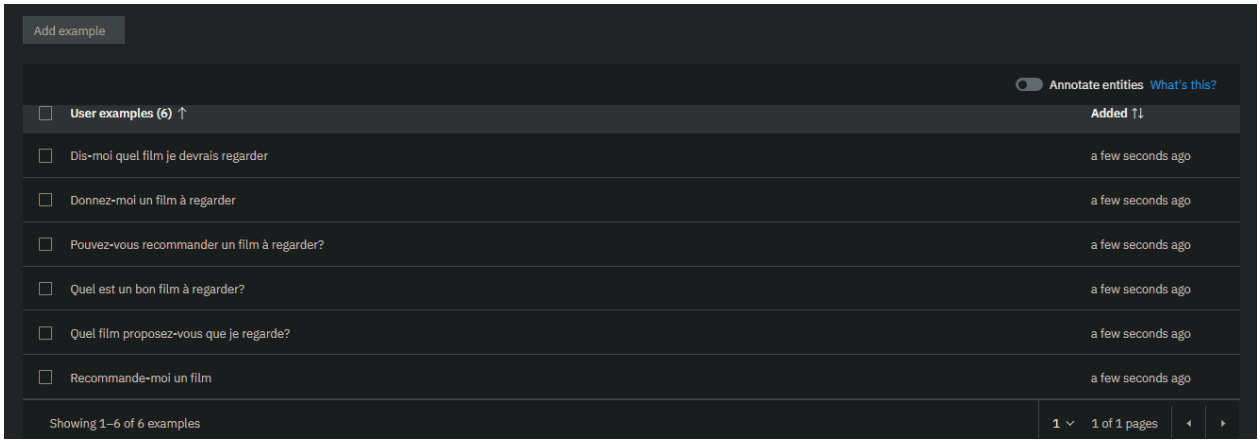
6. From the left menu click Dialog, then click Create Intent +.



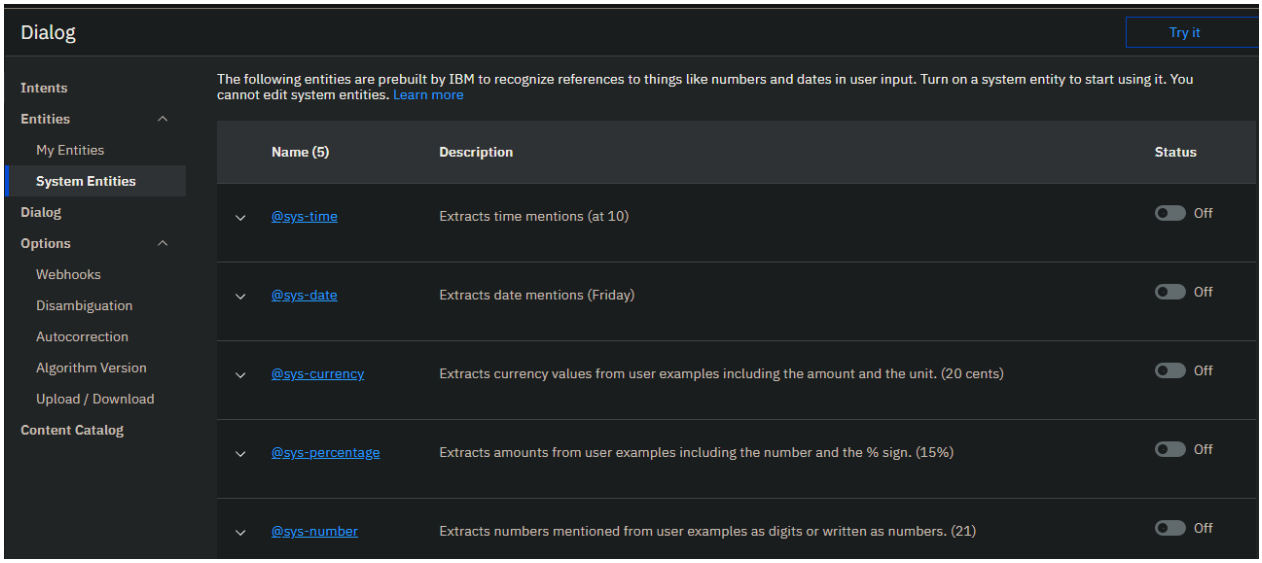
7. Name the intent movie_recommendation, then click Create intent.

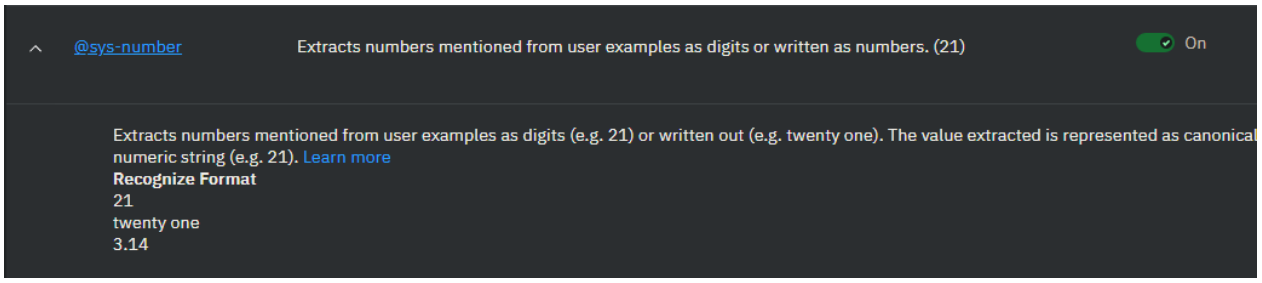


8. Type user examples as follows

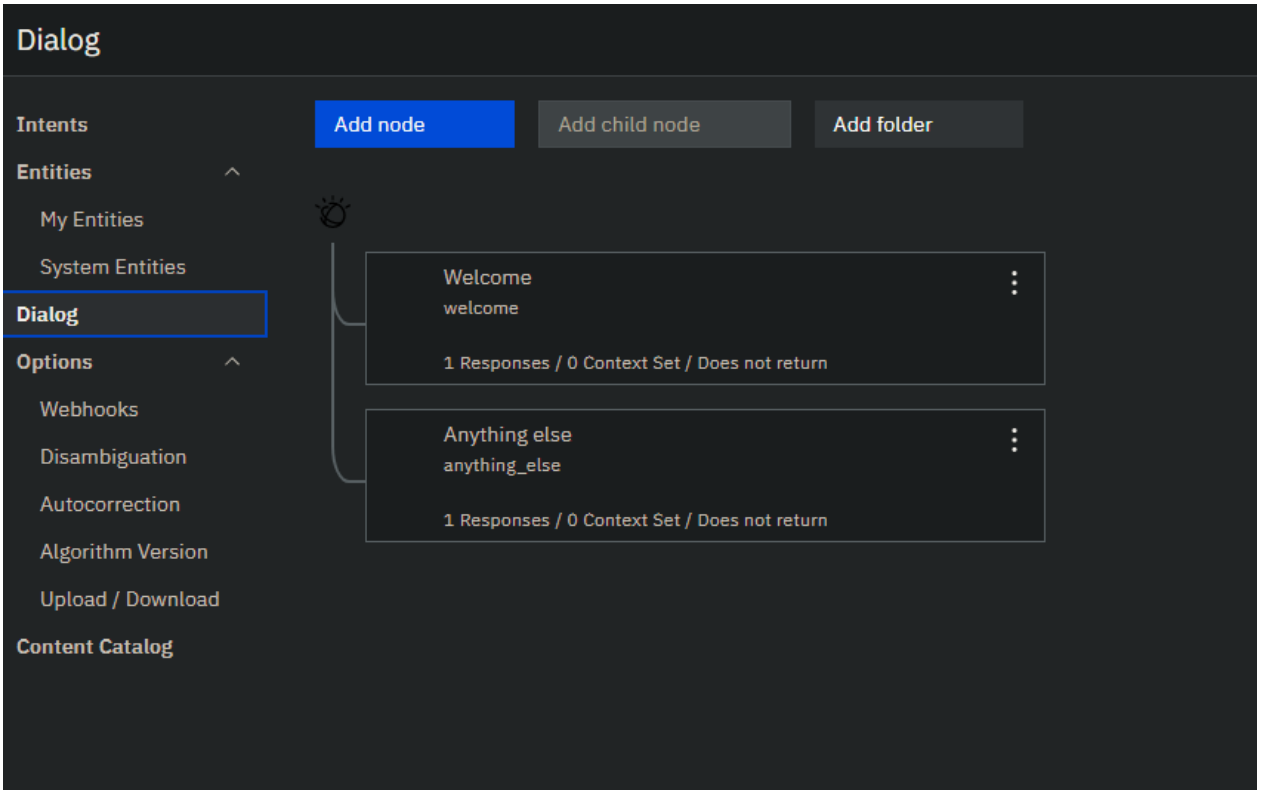


9. Return to the main assistant page. Click System Entities, then enable @sys-number. To personalize the movie recommendations, you use user IDs.

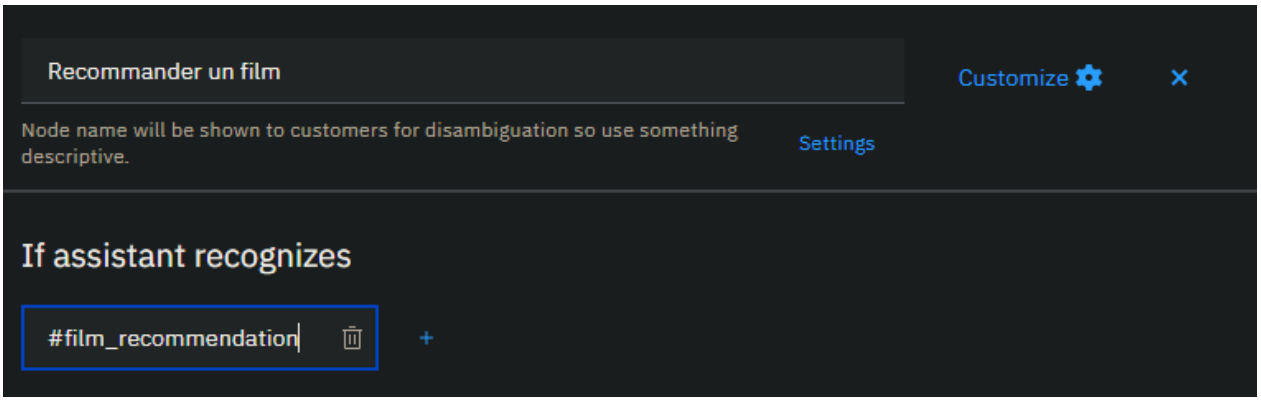




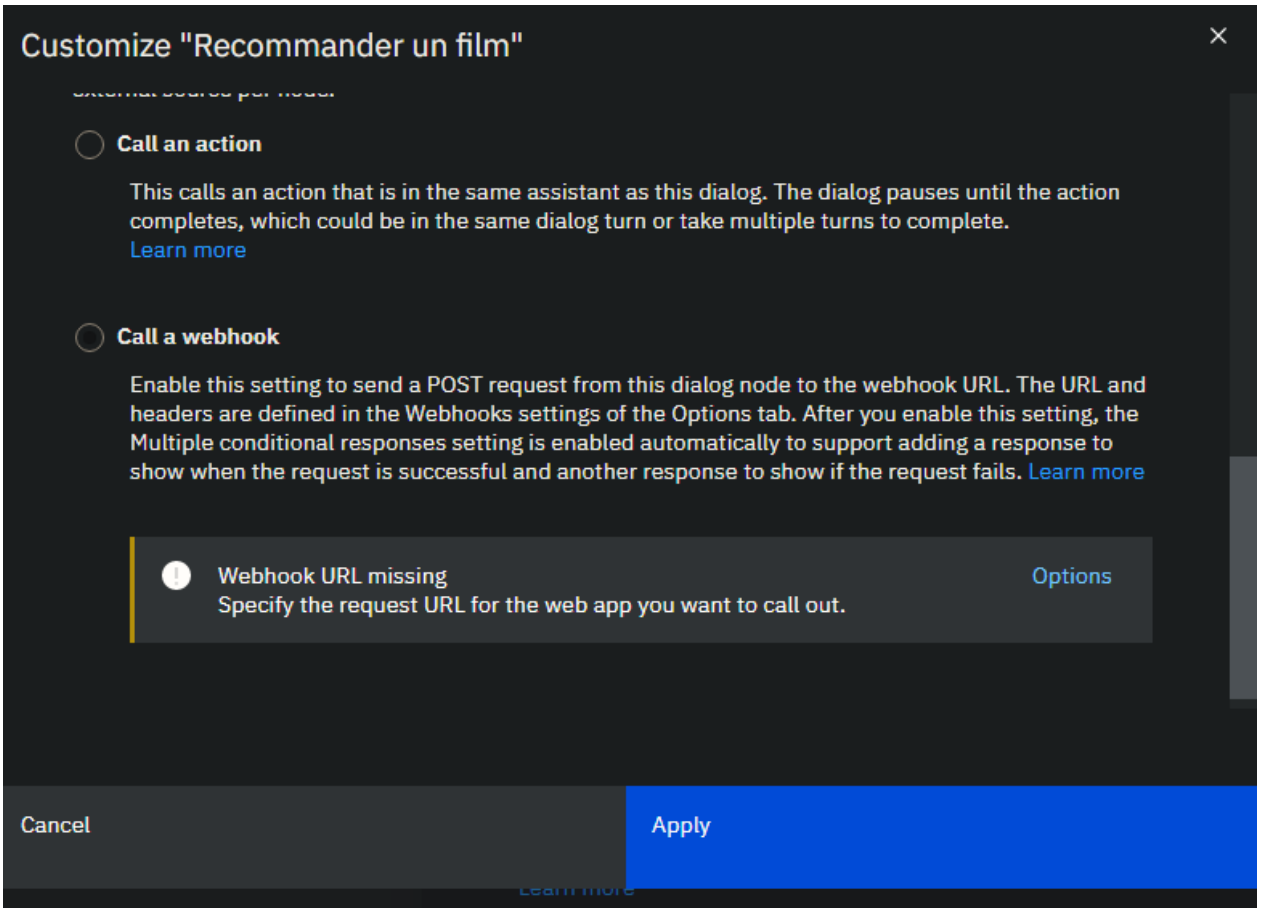
10. Select Dialog from the menu pane on the left.



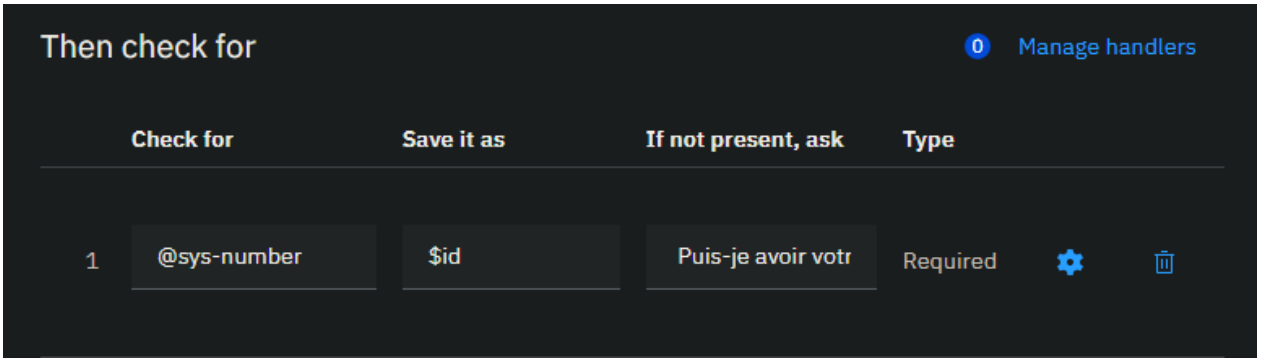
11. Enable customizations for Slots and Callout to Webhooks.



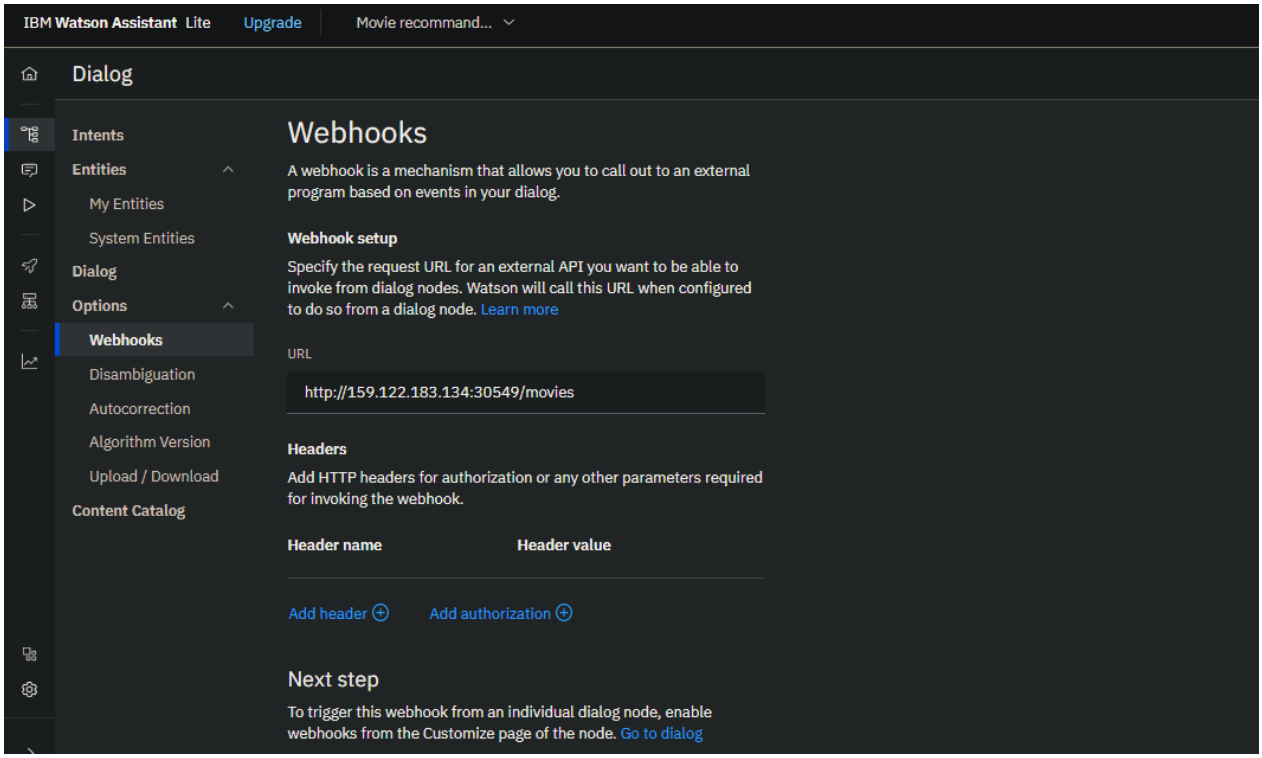
12. Enable Slots and Callout to Webhooks as shown in the following figure and then click Apply.



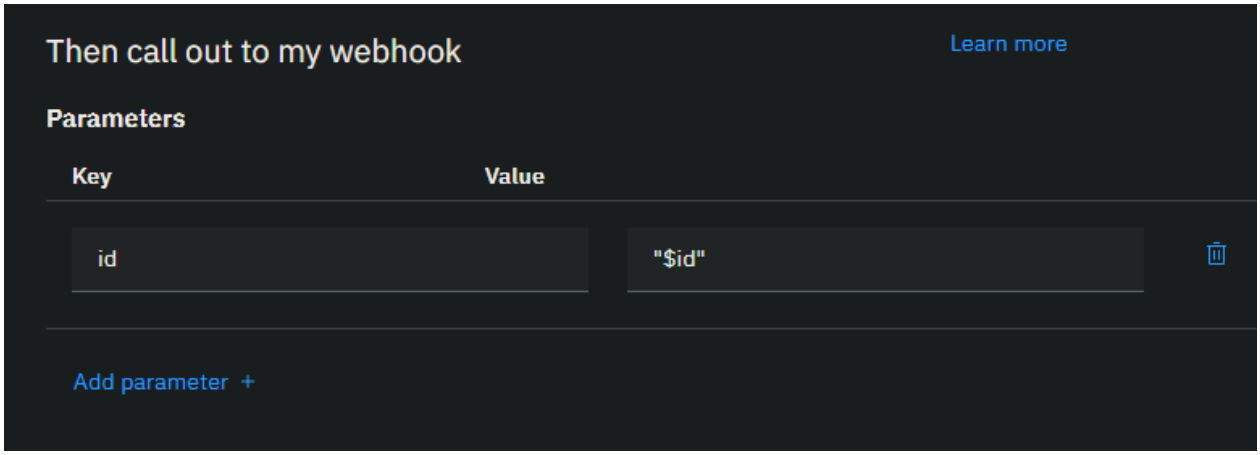
13. Check for the presence of a user ID. Add @sys-number in the “Check for” field, save it as $id, then add Can I please have your ID? in the “If not present, ask” field.



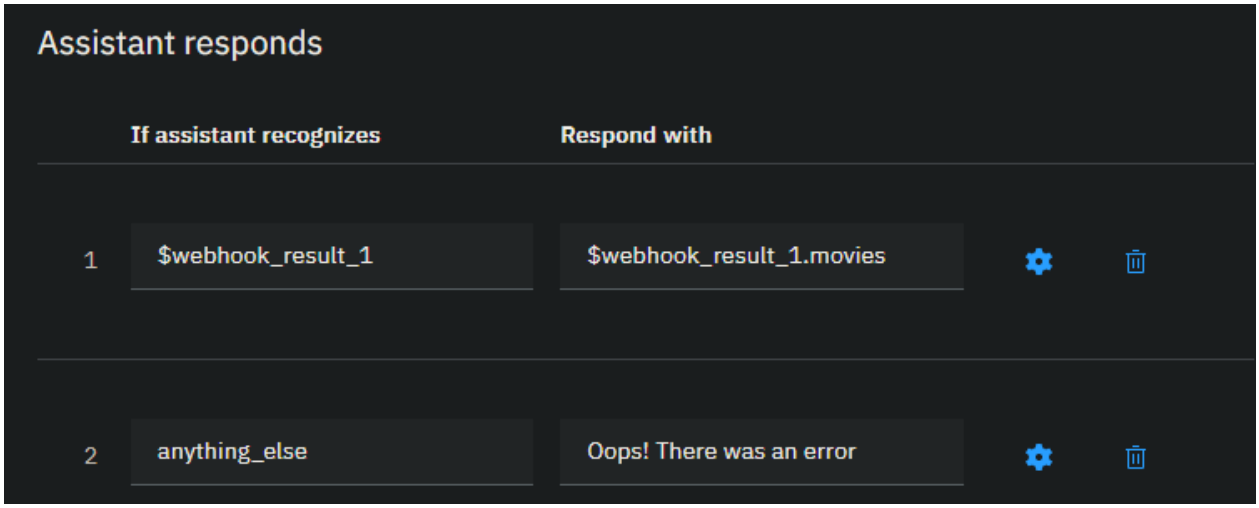
14. Set the webhook URL. Enter the APP URL that you copied in “Part 2 Creating and deploying the Node-RED App”:



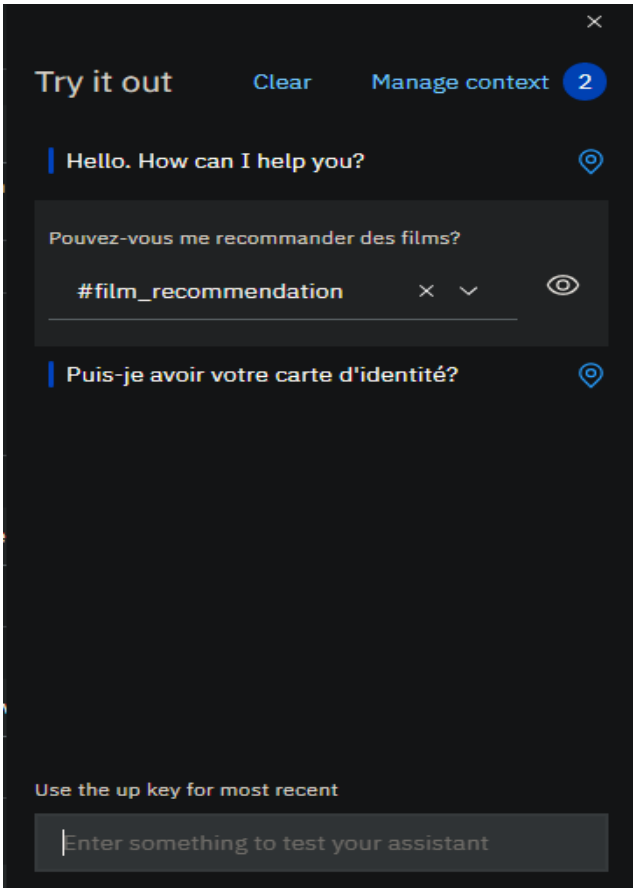
15. Configure the call out to the webhook. Enter Key and Value as shown in the following figure.



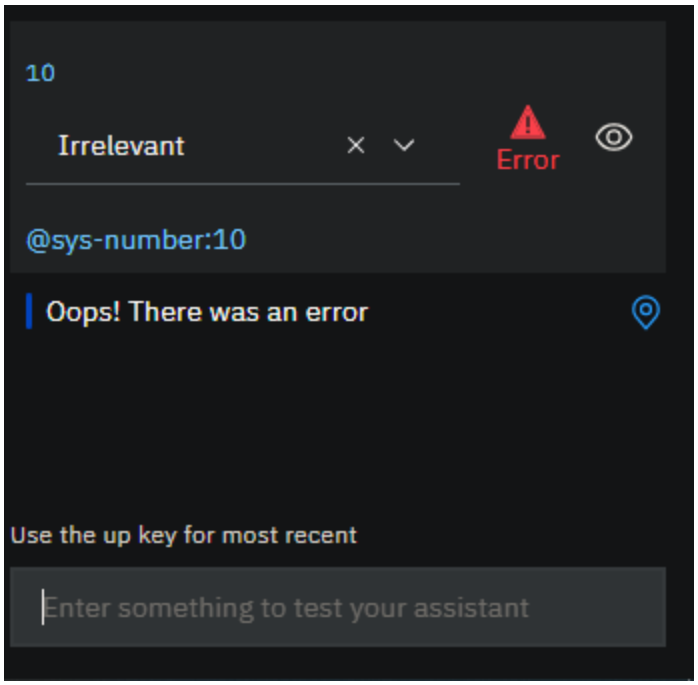
16. Leave the webhook return variable as is. The webhook responds to your request with JSON data. Leave the response as webhook_result_1, which is a variable that stores the data.
Click Try it to test the assistant up to this point.



17. Type Can you recommend some movies for me? The assistant detects the #movie_recommendation intent, then it asks you for your ID.



18. Type “10” as your ID. The assistant detects “10” as @sys-number, and for now it returns an error message “Oops! There was an error” because you did not implement the webhook in Node-RED yet.



Train and deploy the recommender model.

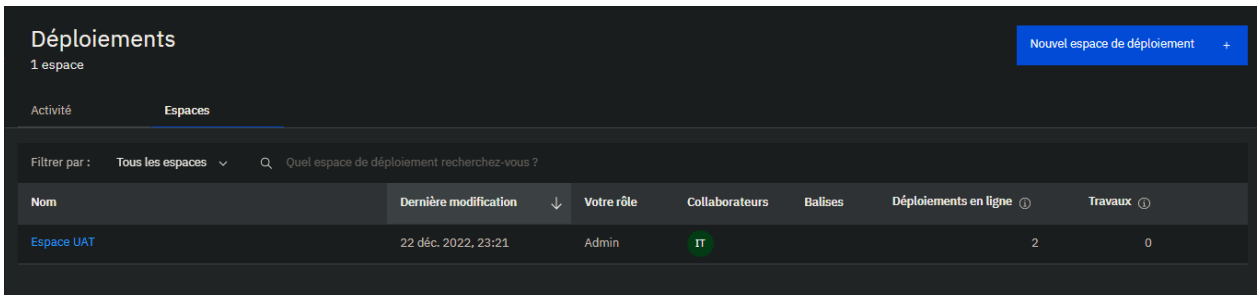
1. Click IBM Cloud Shell. Run the following command to get API Key:

```
idrissatraore0911@cloudshell:~$ ibmcloud iam api-key-create API_KEY_NAME
Creating API key API_KEY_NAME under 90cb73d331b64dab99340a1241473dfe as idrissatraore0911@gmail.com...
OK
API key API_KEY_NAME was created

Please preserve the API key! It cannot be retrieved after it's created.

ID            ApiKey-08cda5f4-1192-48a9-9bae-9c0fd5d5d3b5
Name          API_KEY_NAME
Description
Created At    2022-12-24T07:02+0000
API Key       G6L7aFqaawkOmIp2q-G41LR4sPypRZSBA4cj9K--KmdM
Locked        false
idrissatraore0911@cloudshell:~$
```

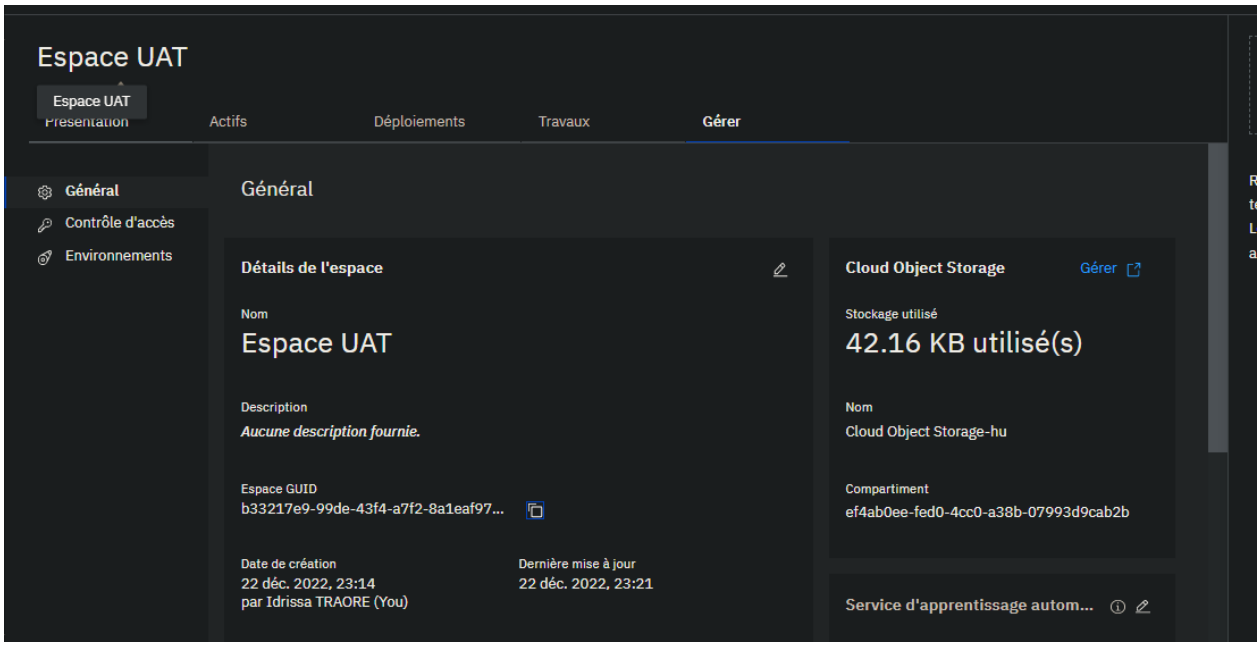
2. To get the location and the corresponding url, run the following command:

```
idrissatraore0911@cloudshell:~$ ibmcloud resource service-instance Watson\ Machine\ Learning-mc
Retrieving service instance Watson Machine Learning-mc in all resource groups under account Idrissa TRAORE's Account as idrissatraore0911@gmail.com...
OK

Name:                  Watson Machine Learning-mc
ID:                    crn:v1:bluemix:public:pm-20:eu-de:a/90cb73d331b64dab99340a1241473dfe:1b83a1ee-f759-4cc7-b60b-5a1eefde0f86::
GUID:                  1b83a1ee-f759-4cc7-b60b-5a1eefde0f86
Location:              eu-de
Service Name:          pm-20
Service Plan Name:     lite
Resource Group Name:   Default
State:                 active
Type:                  service_instance
Sub Type:
Locked:                false
Created at:            2022-12-22T22:05:02Z
Created by:            idrissatraore0911@gmail.com
Updated at:            2022-12-22T22:05:05Z
Last Operation:
                       Status    create succeeded
                       Message   Completed create instance operation

idrissatraore0911@cloudshell:~$
```

3. To get the space id, from IBM Cloud Pak for Data, click the hamburger menu > Deployments.



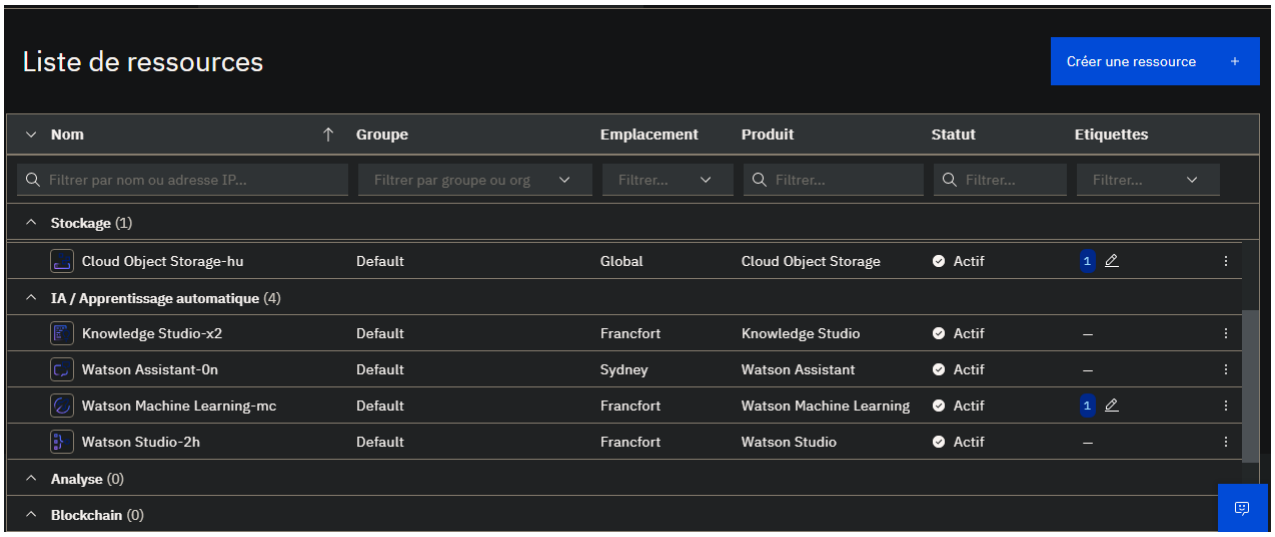
4. Click the Manage tab. Copy Space GUID. Use this value for space_id.



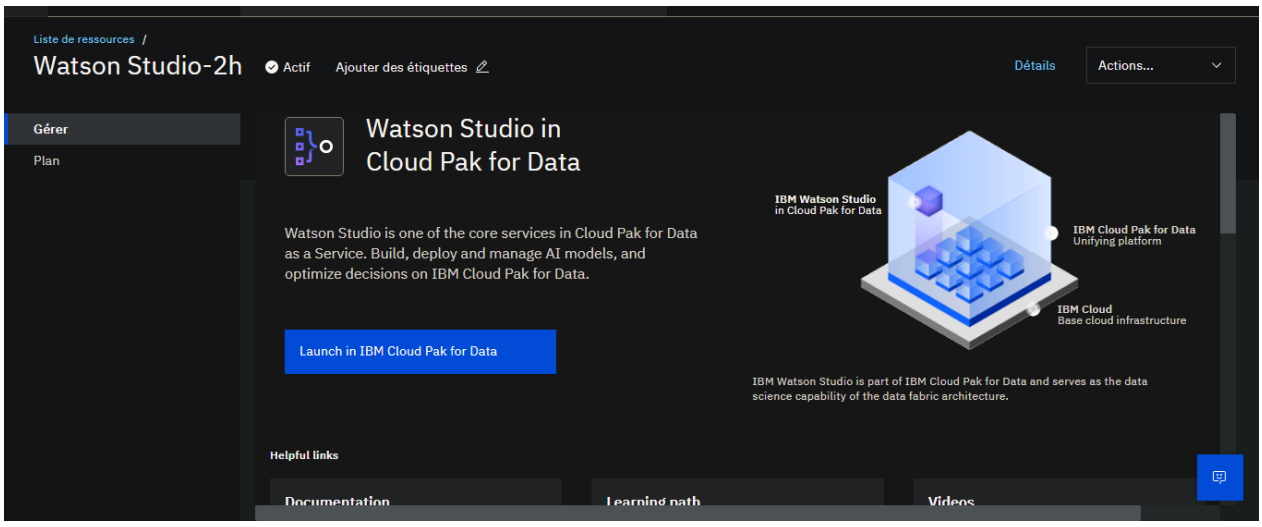
Download the notebook file collaborative_filtering_movielens_with_model_dep.ipynb from https://github.com/IBM-SkillsAcademy/artificial-intelligence-analyst/blob/master/ex12/collaborative_filtering_movielens_with_model_dep.ipynb

5. From the IBM Cloud Dashboard, click Resource list.

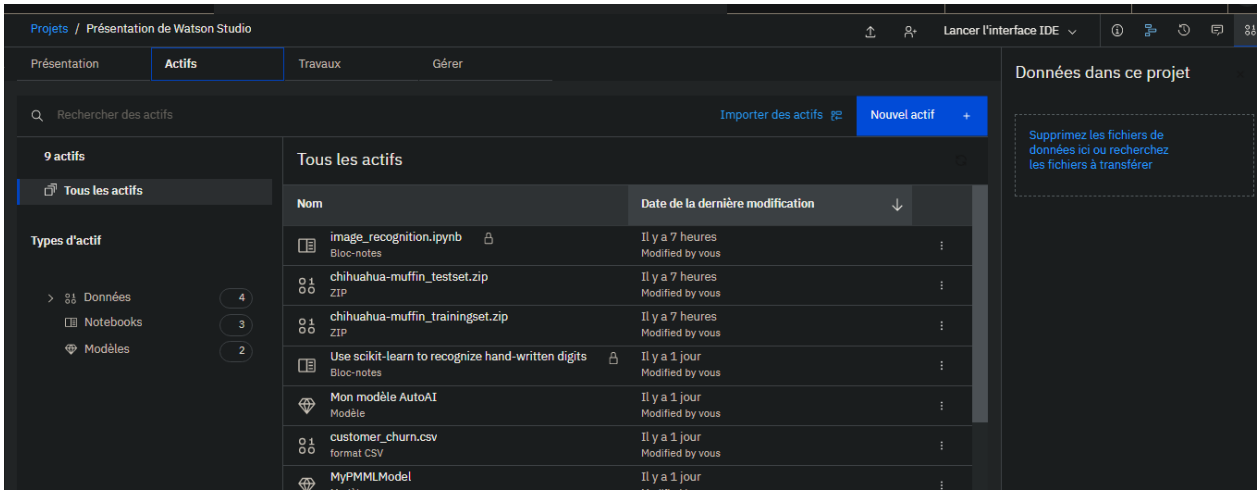
Expand Services and software, then click your Watson Studio service.



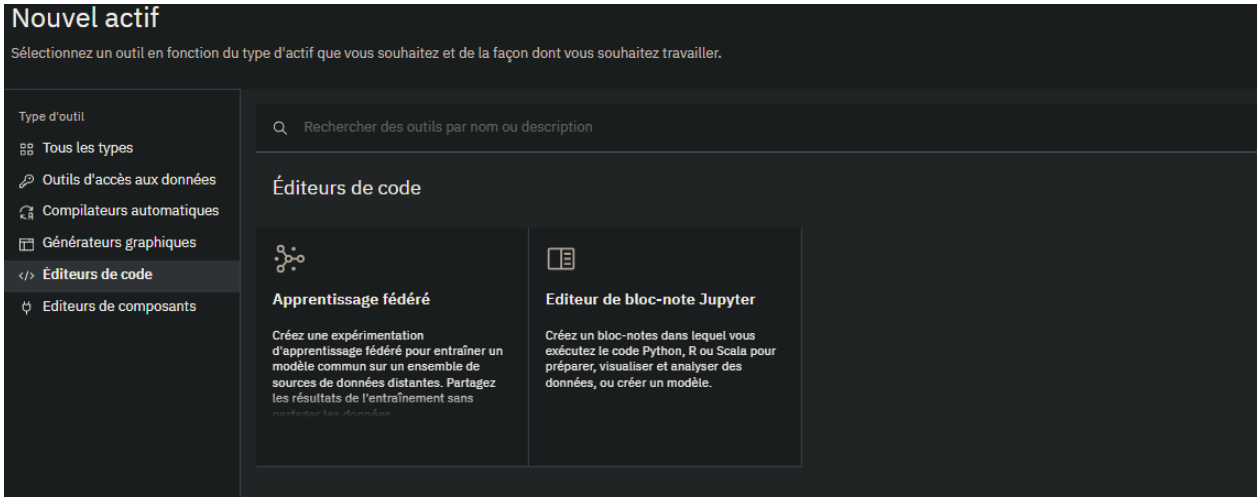
6. Expand Services and software, then click your Watson Studio service.



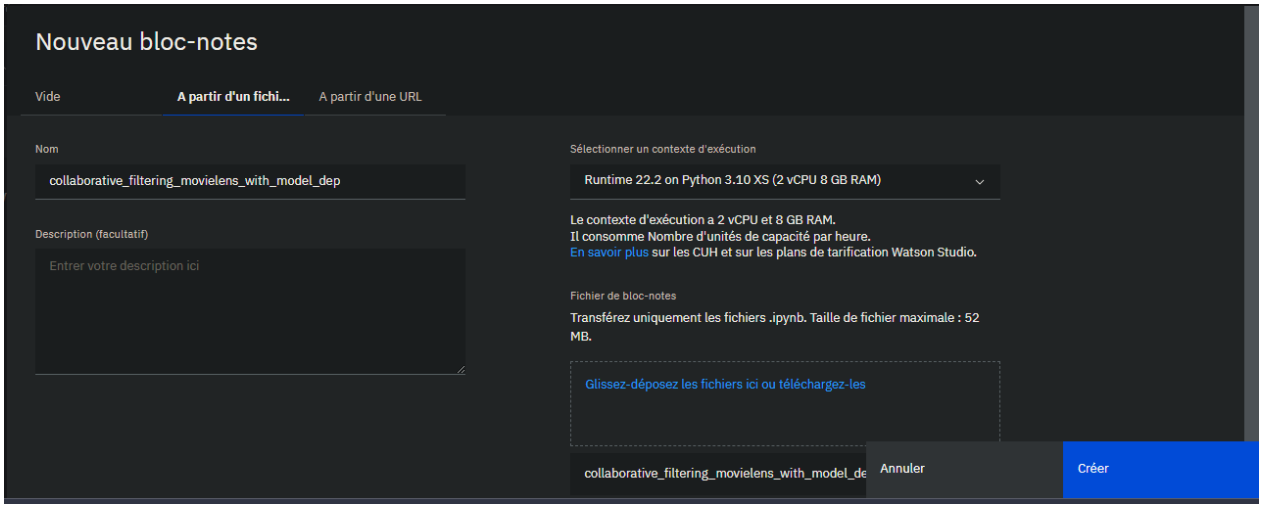
7. Open your Watson Studio project. Click the Assets tab and then click New asset +



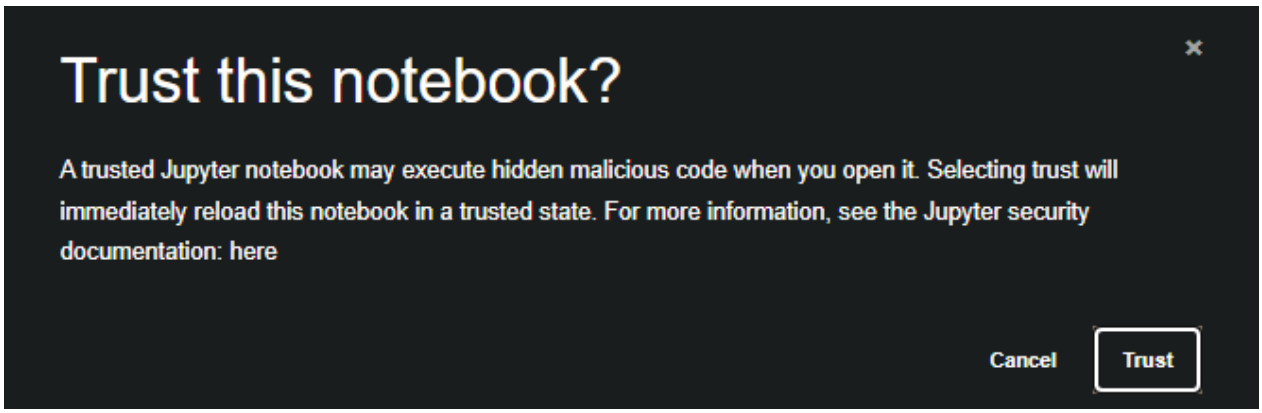
8. From the left pane choose Code editors, then click Jupyter notebook editor.



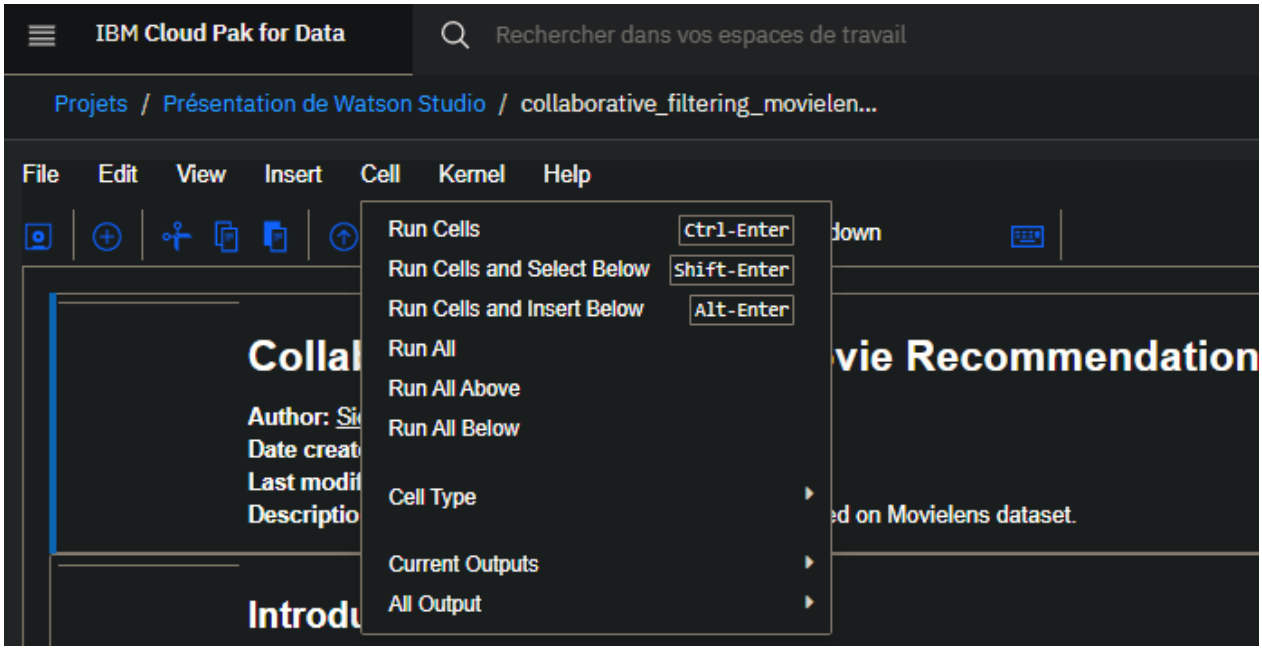
9. Configure the new asset.



10. Wait until the notebook is opened, then click Not trusted then Trust.



11. Click Cell > All Output > Clear to remove any output from previous runs.



12. Read the notebook comments and run the cells one by one until you reach the cell to save the model. Compare your output with the following figures.

```
In [1]: import pandas as pd
        import numpy as np
        from zipfile import ZipFile
        import tensorflow as tf
        from tensorflow import keras
        from tensorflow.keras import layers
        from pathlib import Path
        import matplotlib.pyplot as plt
```

First, load the data and apply preprocessing

In [2]:
```
# Download the actual data from http://files.grouplens.org/datasets/movielens/ml-latest-small.zip"
# Use the ratings.csv file
movielens_data_file_url = (
    "http://files.grouplens.org/datasets/movielens/ml-latest-small.zip"
)
movielens_zipped_file = keras.utils.get_file(
    "ml-latest-small.zip", movielens_data_file_url, extract=False
)
keras_datasets_path = Path(movielens_zipped_file).parents[0]
movielens_dir = keras_datasets_path / "ml-latest-small"

# Only extract the data the first time the script is run.
if not movielens_dir.exists():
    with ZipFile(movielens_zipped_file, "r") as zip:
        # Extract files
        print("Extracting all the files now...")
        zip.extractall(path=keras_datasets_path)
        print("Done!")

ratings_file = movielens_dir / "ratings.csv"
df = pd.read_csv(ratings_file)
```

```
Downloading data from http://files.grouplens.org/datasets/movielens/ml-latest-small.zip
978202/978202 [==============================] - 1s 1us/step
Extracting all the files now...
Done!
```

In [3]:
```
user_ids = df["userId"].unique().tolist()
user2user_encoded = {x: i for i, x in enumerate(user_ids)}
userencoded2user = {i: x for i, x in enumerate(user_ids)}
movie_ids = df["movieId"].unique().tolist()
movie2movie_encoded = {x: i for i, x in enumerate(movie_ids)}
movie_encoded2movie = {i: x for i, x in enumerate(movie_ids)}
df["user"] = df["userId"].map(user2user_encoded)
df["movie"] = df["movieId"].map(movie2movie_encoded)

num_users = len(user2user_encoded)
num_movies = len(movie_encoded2movie)
df["rating"] = df["rating"].values.astype(np.float32)
# min and max ratings will be used to normalize the ratings later
min_rating = min(df["rating"])
max_rating = max(df["rating"])

print(
    "Number of users: {}, Number of Movies: {}, Min rating: {}, Max rating: {}".format(
        num_users, num_movies, min_rating, max_rating
    )
)
```

```
Number of users: 610, Number of Movies: 9724, Min rating: 0.5, Max rating: 5.0
```

Prepare training and validation data

In [4]:
```
df = df.sample(frac=1, random_state=42)
x = df[["user", "movie"]].values
# Normalize the targets between 0 and 1. Makes it easy to train.
y = df["rating"].apply(lambda x: (x - min_rating) / (max_rating - min_rating)).values
# Assuming training on 90% of the data and validating on 10%.
train_indices = int(0.9 * df.shape[0])
x_train, x_val, y_train, y_val = (
    x[:train_indices],
    x[train_indices:],
    y[:train_indices],
    y[train_indices:],
)
```

Create the model

We embed both users and movies in to 50-dimensional vectors.

The model computes a match score between user and movie embeddings via a dot product, and adds a per-movie and per-user bias. The match score is scaled to the `[0, 1]` interval via a sigmoid (since our ratings are normalized to this range).

In [5]:
```
EMBEDDING_SIZE = 50


class RecommenderNet(keras.Model):
    def __init__(self, num_users, num_movies, embedding_size, **kwargs):
        super(RecommenderNet, self).__init__(**kwargs)
        self.num_users = num_users
        self.num_movies = num_movies
        self.embedding_size = embedding_size
        self.user_embedding = layers.Embedding(
            num_users,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(1e-6),
        )
        self.user_bias = layers.Embedding(num_users, 1)
        self.movie_embedding = layers.Embedding(
            num_movies,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(1e-6),
        )
        self.movie_bias = layers.Embedding(num_movies, 1)

    def call(self, inputs):
        user_vector = self.user_embedding(inputs[:, 0])
        user_bias = self.user_bias(inputs[:, 0])
        movie_vector = self.movie_embedding(inputs[:, 1])
        movie_bias = self.movie_bias(inputs[:, 1])
        dot_user_movie = tf.tensordot(user_vector, movie_vector, 2)
        # Add all the components (including bias)
        x = dot_user_movie + user_bias + movie_bias
        # The sigmoid activation forces the rating to between 0 and 1
        return tf.nn.sigmoid(x)


model = RecommenderNet(num_users, num_movies, EMBEDDING_SIZE)
model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(), optimizer=keras.optimizers.Adam(learning_rate=0.001)
)
```

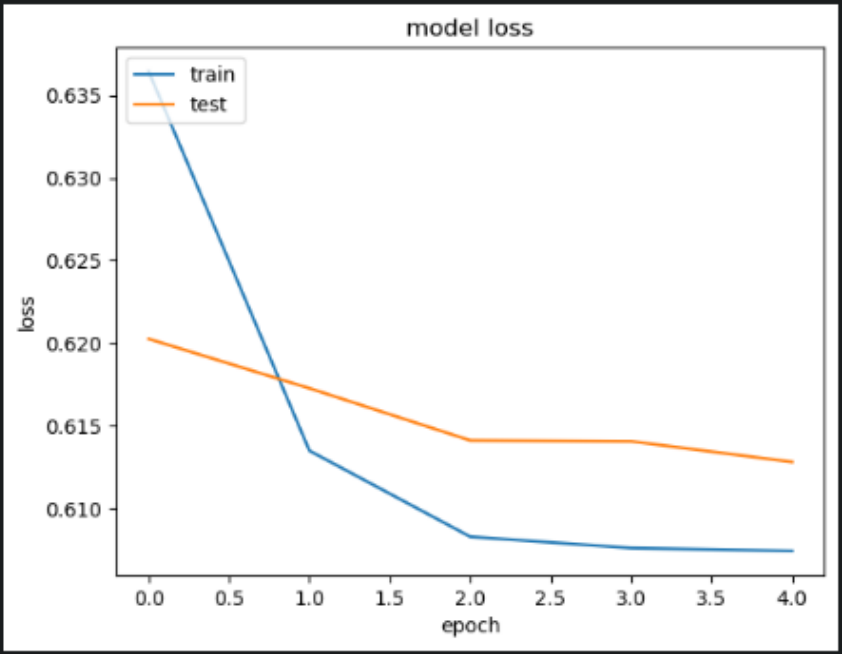
Train the model based on the data split

In [14]:
```
history = model.fit(
    x=x_train,
    y=y_train,
    batch_size=64,
    epochs=5,
    verbose=1,
    validation_data=(x_val, y_val),
)
```

```
Epoch 1/5
1418/1418 [==============================] - 10s 7ms/step - loss: 0.6365 - val_loss: 0.6203
Epoch 2/5
1418/1418 [==============================] - 10s 7ms/step - loss: 0.6135 - val_loss: 0.6173
Epoch 3/5
1418/1418 [==============================] - 10s 7ms/step - loss: 0.6083 - val_loss: 0.6141
Epoch 4/5
1418/1418 [==============================] - 9s 7ms/step - loss: 0.6076 - val_loss: 0.6140
Epoch 5/5
1418/1418 [==============================] - 10s 7ms/step - loss: 0.6074 - val_loss: 0.6128
```

Plot training and validation loss

In [15]:
```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("model loss")
plt.ylabel("loss")
plt.xlabel("epoch")
plt.legend(["train", "test"], loc="upper left")
plt.show()
```




```
298/298 [==============================] - 1s 1ms/step
Showing recommendations for user: 119
====================================
Movies with high ratings from user
--------------------------------
Lion King, The (1994) : Adventure|Animation|Children|Drama|Musical|IMAX
Green Mile, The (1999) : Crime|Drama
Up (2009) : Adventure|Animation|Children|Drama
Avatar (2009) : Action|Adventure|Sci-Fi|IMAX
The Imitation Game (2014) : Drama|Thriller|War
--------------------------------
Top 10 movie recommendations
--------------------------------
Usual Suspects, The (1995) : Crime|Mystery|Thriller
Godfather, The (1972) : Crime|Drama
Casablanca (1942) : Drama|Romance
Star Wars: Episode V - The Empire Strikes Back (1980) : Action|Adventure|Sci-Fi
Goodfellas (1990) : Crime|Drama
Godfather: Part II, The (1974) : Crime|Drama
Cool Hand Luke (1967) : Drama
Jaws (1975) : Action|Horror
Raising Arizona (1987) : Comedy
Fight Club (1999) : Action|Crime|Drama|Thriller
```

```
In [17]: model.save('movies_model',save_format='tf')
         !tar -C movies_model -zcvf movies_model.tar.gz ./

         INFO:tensorflow:Assets written to: movies_model/assets
         ./
         ./saved_model.pb
         ./keras_metadata.pb
         ./variables/
         ./variables/variables.index
         ./variables/variables.data-00000-of-00001
         ./assets/

In [18]: # Model deployment:
         # Add your region and apikey
         # Add space guid
```

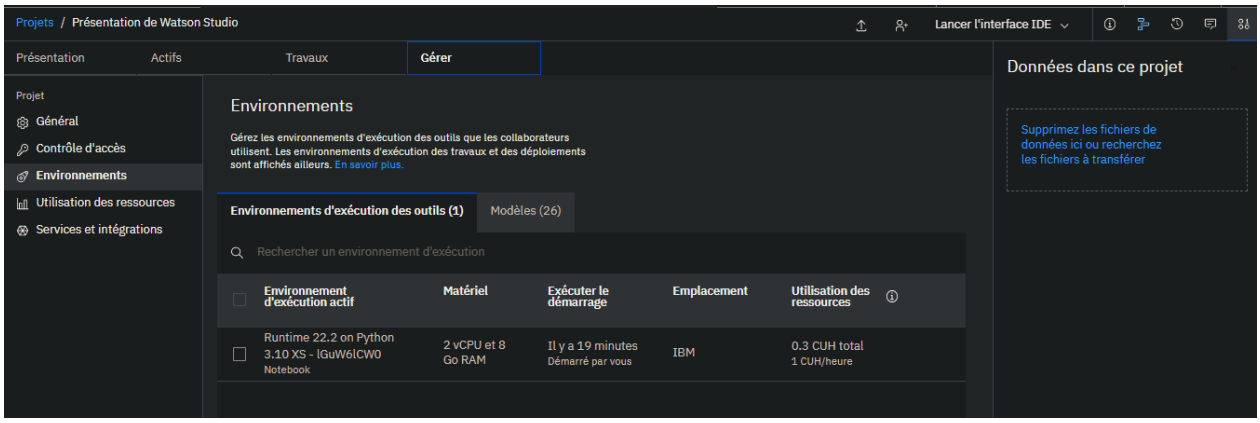
13. To deploy the model, replace the url, apikey, and space id with your values that you obtained in the section “Getting the values for “apikey”, “url”, and ”space_id””. Then, run the following cells.

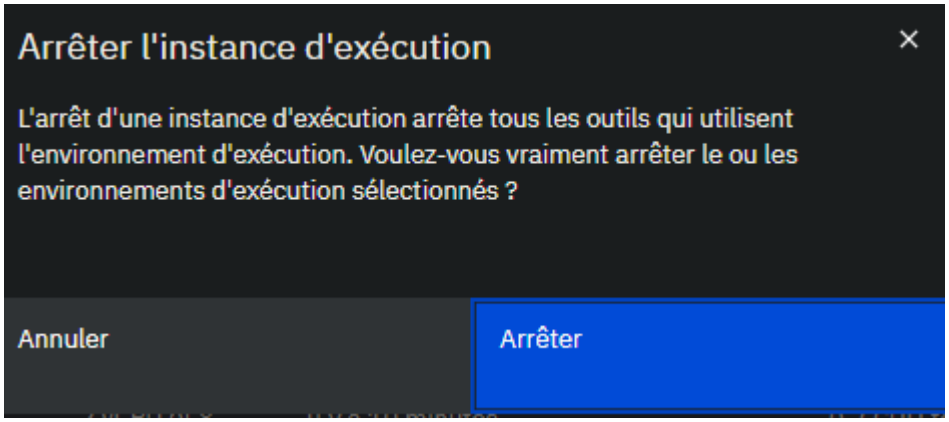
```
In [19]: from ibm_watson_machine_learning import APIClient

         wml_credentials = {
                            "url": "https://eu-de.ml.cloud.ibm.com",
                            "apikey":"lHz1UvO1gNfxxzI7MaBBkiQg5g622tRAXuzW2xwYMH_a"
                           }

         client = APIClient(wml_credentials)

         space_id = 'b33217e9-99de-43f4-a7f2-8a1eaf97b98a'
         client.set.default_space(space_id)

            'SUCCESS'

In [20]: sample_saved_model_filename = "movies_model.tar.gz"
         sofware_spec_uid = client.software_specifications.get_id_by_name("tensorflow_rt22.1-py3.9")

         metadata = {
                     client.repository.ModelMetaNames.NAME: 'keras movie model',
                     client.repository.ModelMetaNames.TYPE: 'tensorflow_rt22.1',
                     client.repository.ModelMetaNames.SOFTWARE_SPEC_UID: sofware_spec_uid
         }
         model_details = client.repository.store_model(
              model=sample_saved_model_filename,
              meta_props=metadata
         )
```

```
In [21]: published_model_uid = client.repository.get_model_id(model_details)
         # Deploy the model
         metadata = {
              client.deployments.ConfigurationMetaNames.NAME: "Keras movie model deployment",
              client.deployments.ConfigurationMetaNames.ONLINE: {}
         }

         model_deployment_details = client.deployments.create(published_model_uid, meta_props=metadata)


         #######################################################################################

         Synchronous deployment creation for uid: '4108b143-d334-4443-806a-8b82f297ca76' started

         #######################################################################################


         initializing
         Note: online_url is deprecated and will be removed in a future release. Use serving_urls instead.
         ...
         ready


         ------------------------------------------------------------------------------------------------
         Successfully finished deployment creation, deployment_uid='8d66e588-8359-4ce2-9aee-1b22e06ca279'
         ------------------------------------------------------------------------------------------------
```

```
In [22]: # copy deployment_uid from the previous cell and replace it here

In [23]: model_deployment_id = "8d66e588-8359-4ce2-9aee-1b22e06ca279"
         payload = {"input_data": [{"values":[[23,123]]}]}

         scores = client.deployments.score(model_deployment_id, payload)

         print(scores)

         {'predictions': [{'id': 'output_1', 'values': [[0.49973264336586]]}]}

In [ ]:
```

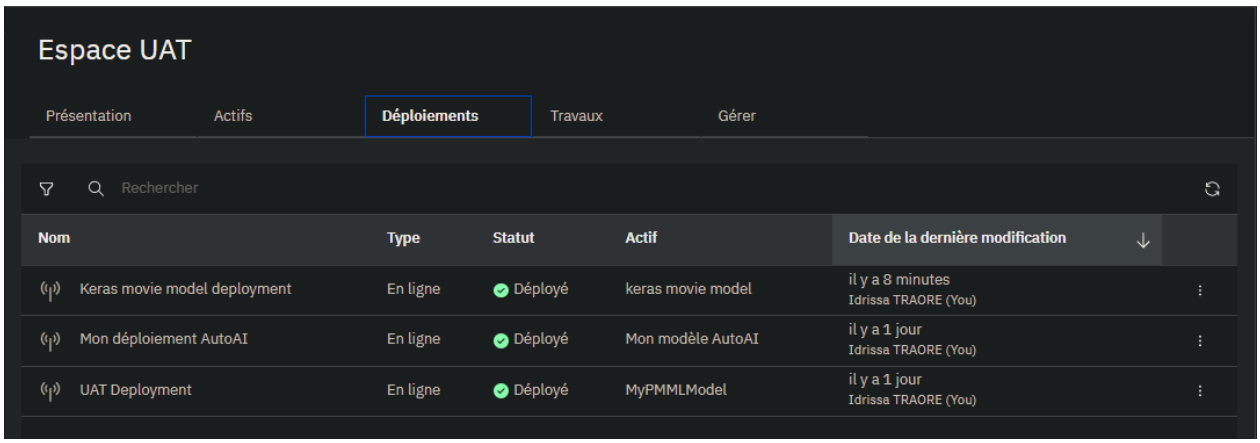
14. Stop the notebook runtime environment to limit your consumption of capacity unit hours (CUH).



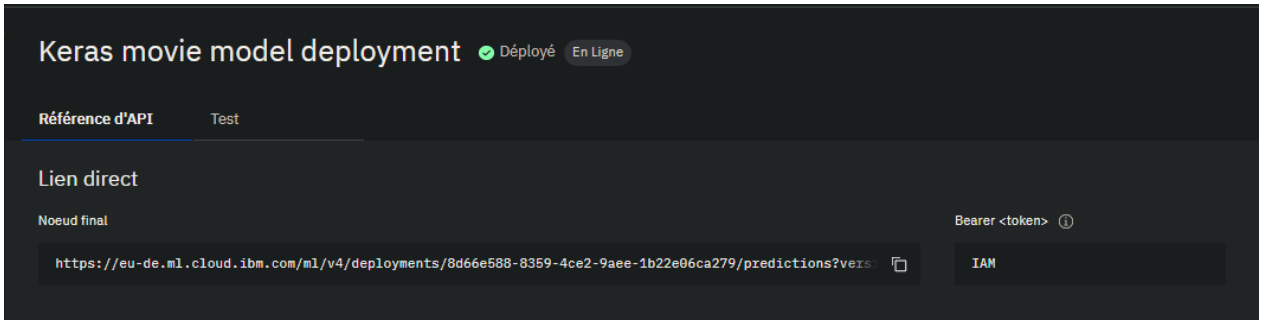


15. Click the hamburger menu, then click Deployments.

Click your space, click the Deployment tab, then click the deployed model.

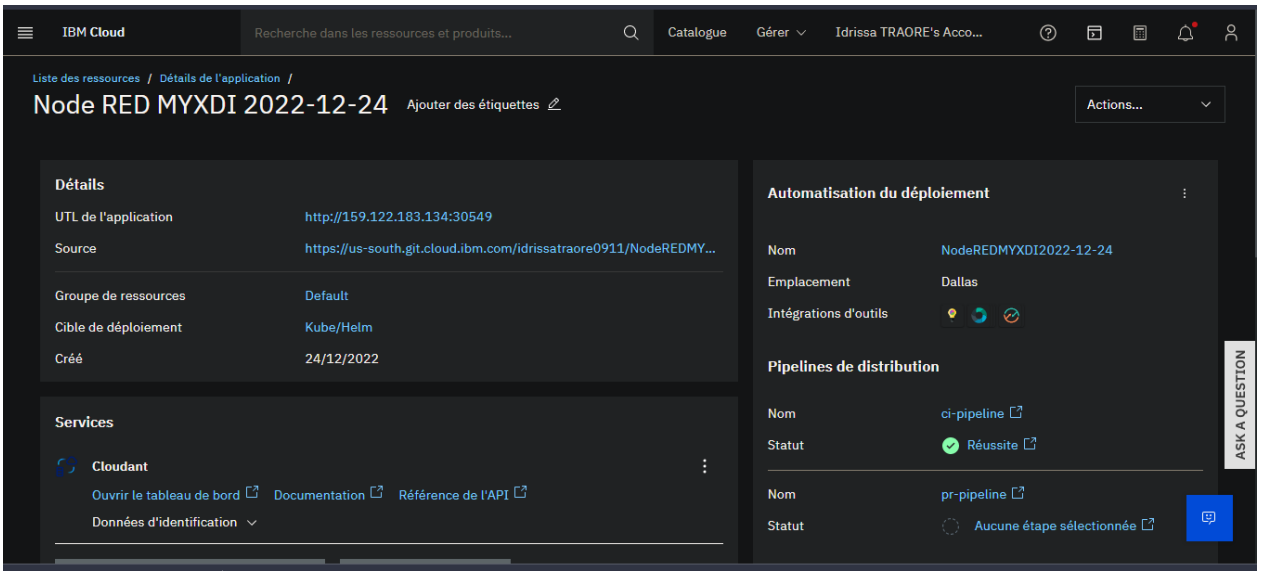


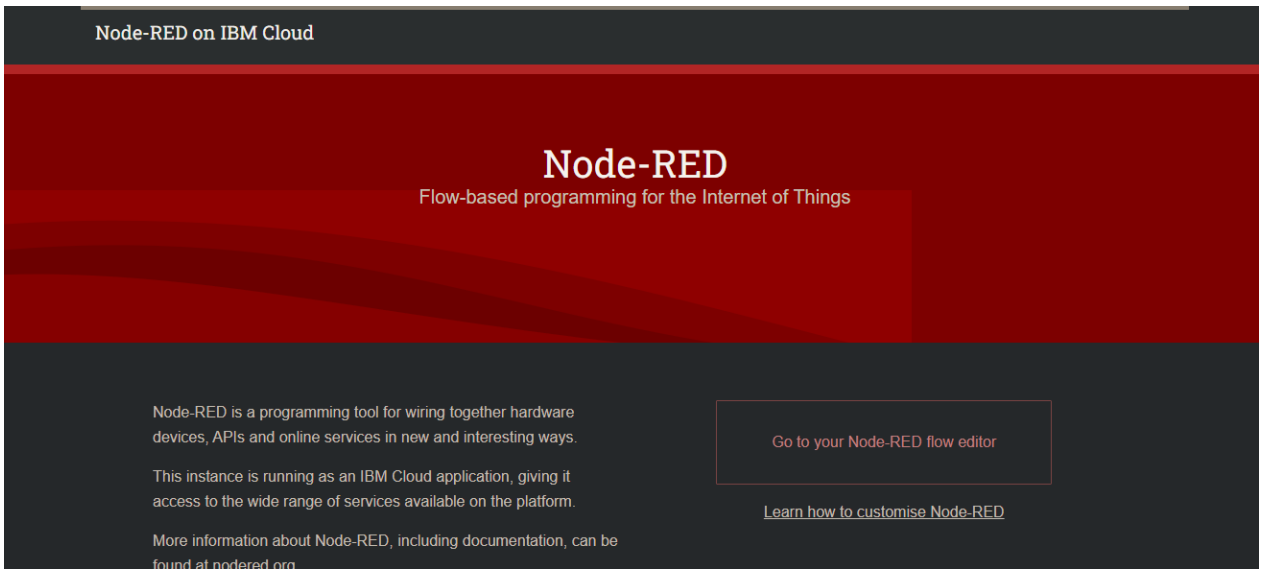
16. Copy the Endpoint. You use it as model_url in “Section 6: Integrating the solution components by using Node-RED”



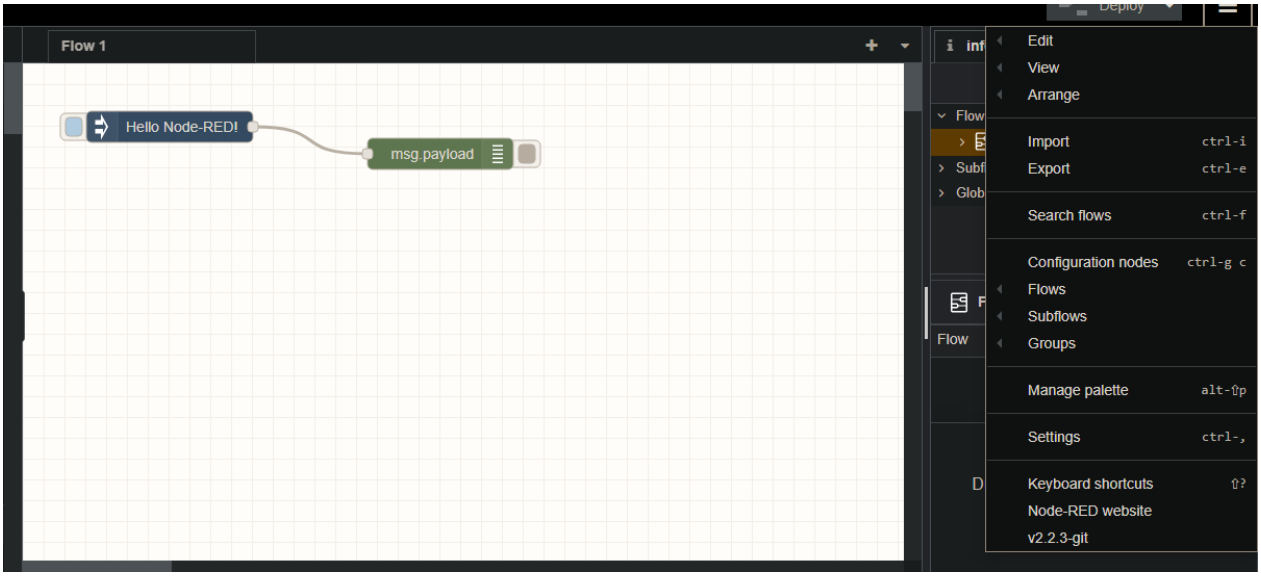
Integrate the solution components by using node-RED.

1. Download the file movie_recommender.json from https://github.com/IBM-SkillsAcademy/artificial-intelligence-analyst/blob/master/ex12/movie_recommender.json
   From IBM Cloud Dashboard, click Resource list.
   Expand Apps, then click your Node-RED app.
   Click the App URL, then click Go to your Node-RED flow editor.

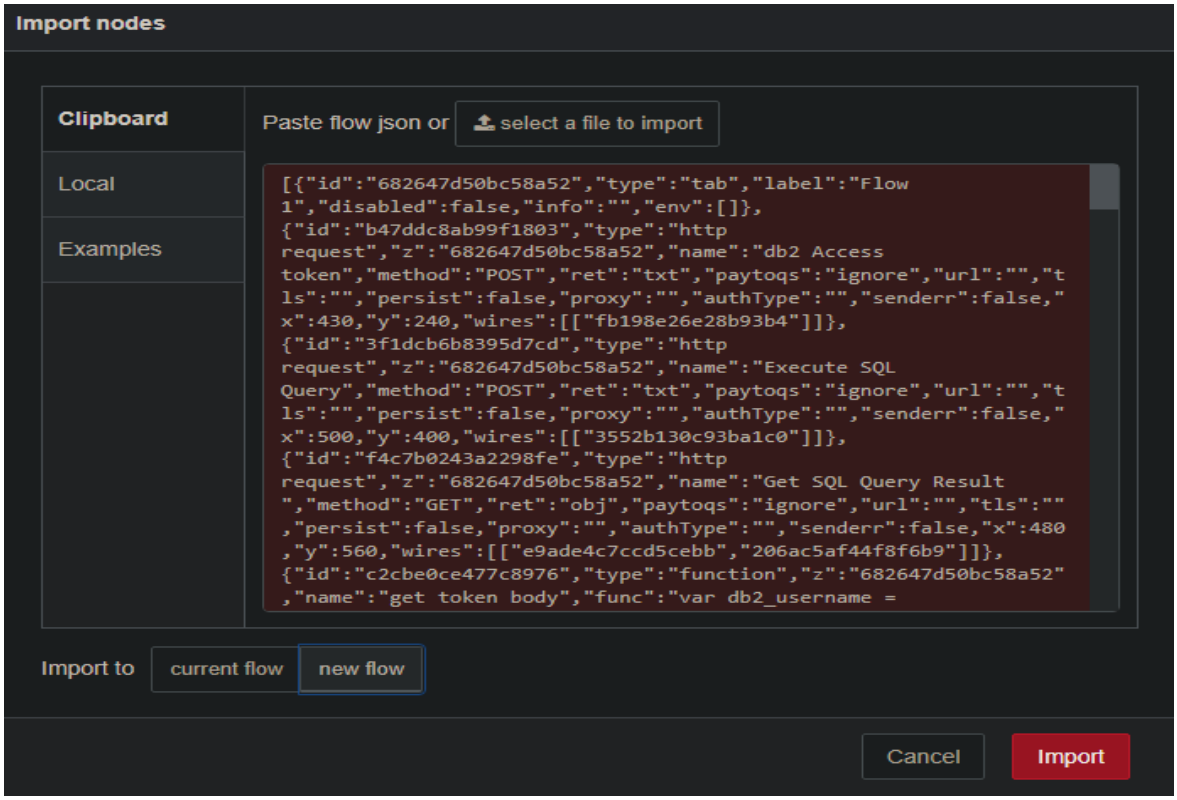




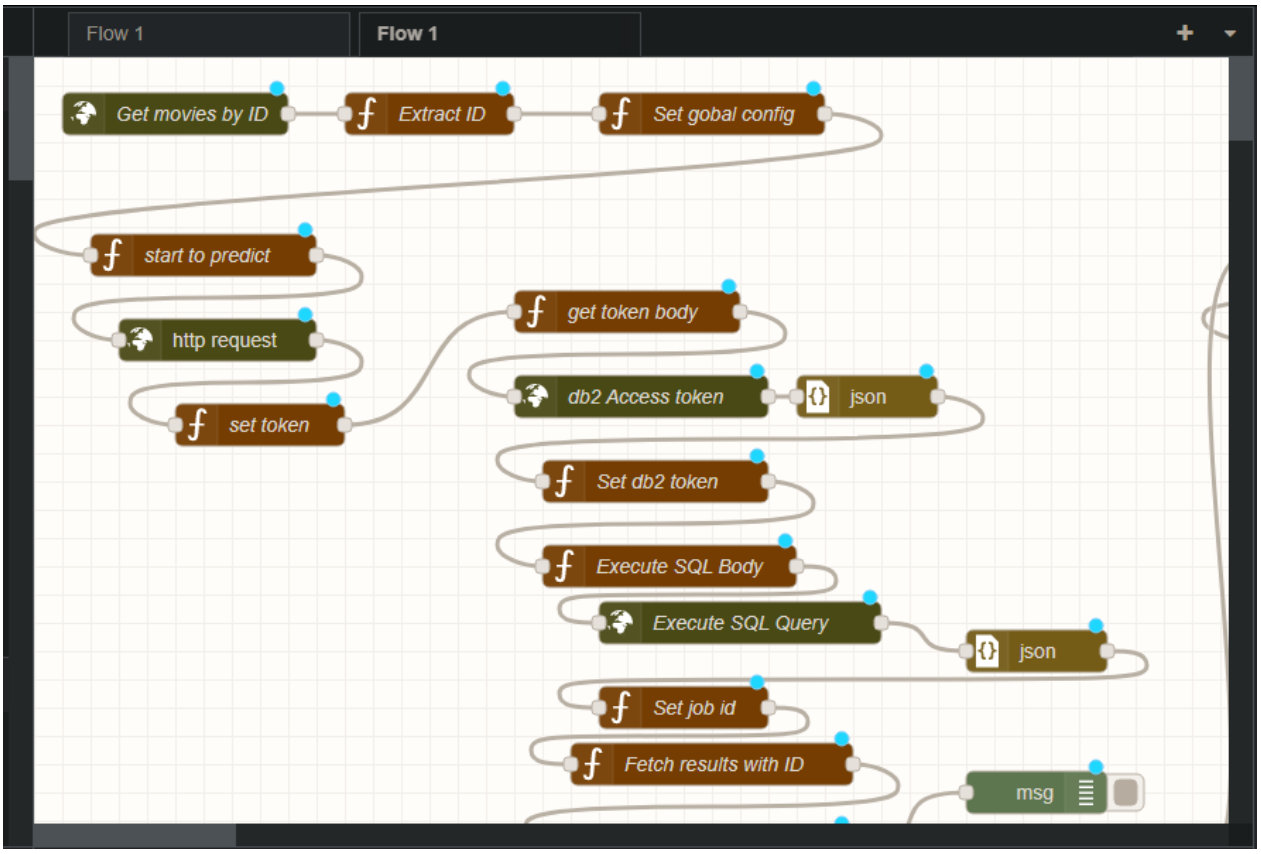
2. The flow editor opens. Click the hamburger menu at the upper right and then click Import to import the integration flow.



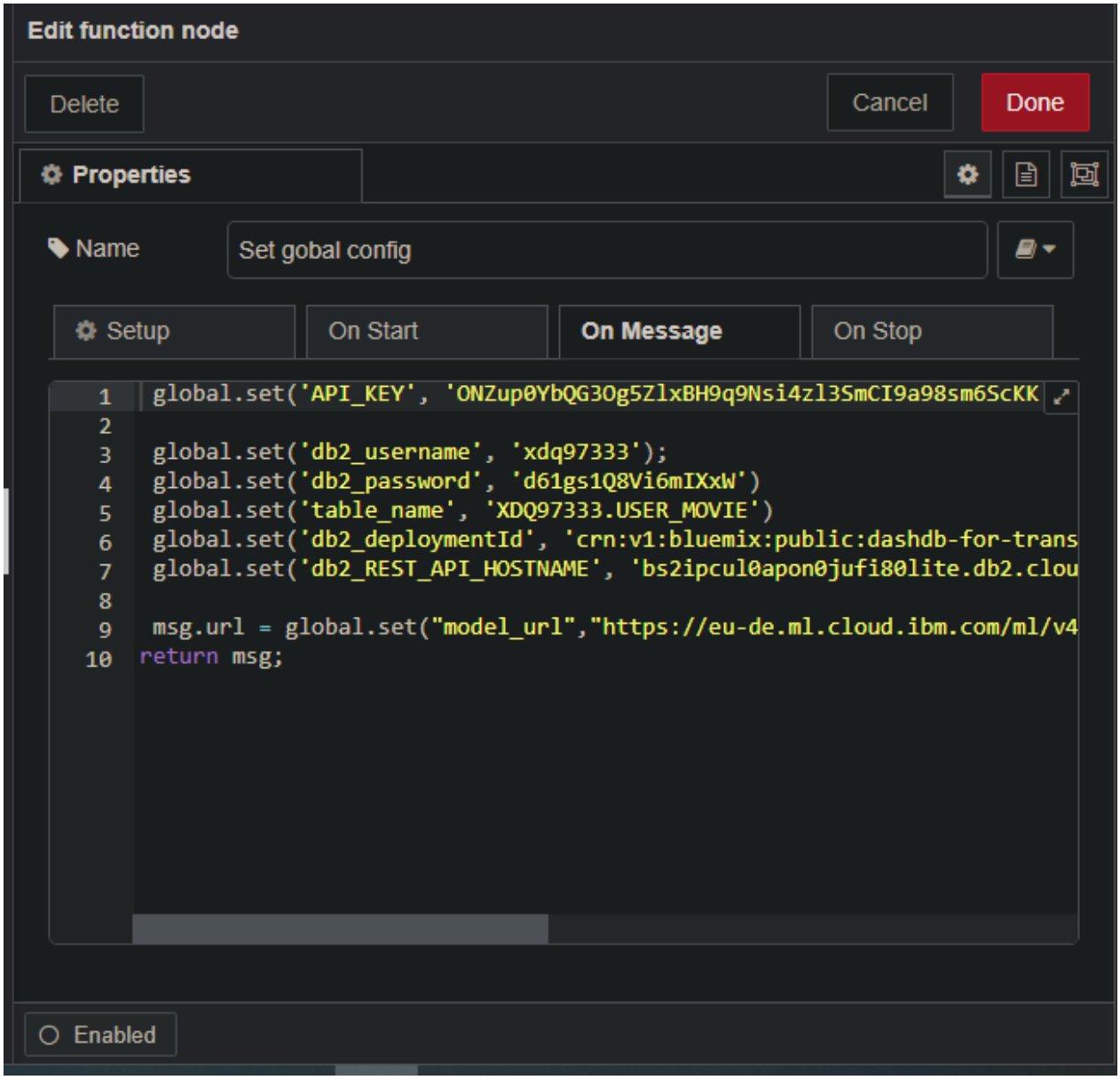
3. Click select a file to import and browse to the file movie_recommender.json that you downloaded earlier from https://github.com/IBM-SkillsAcademy/artificial-intelligence-analyst/blob/master/ex12/movie_recommender.json



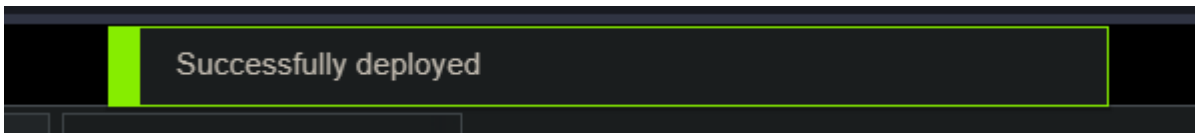
4. Select new flow for the field “Import to” and then click Import.



5. Double-click Set global config node and replace the following parameters with your values, then click Done.
   I insert my apikey, db2_username, db2_password, db2_deploymentId, model_url, table_name

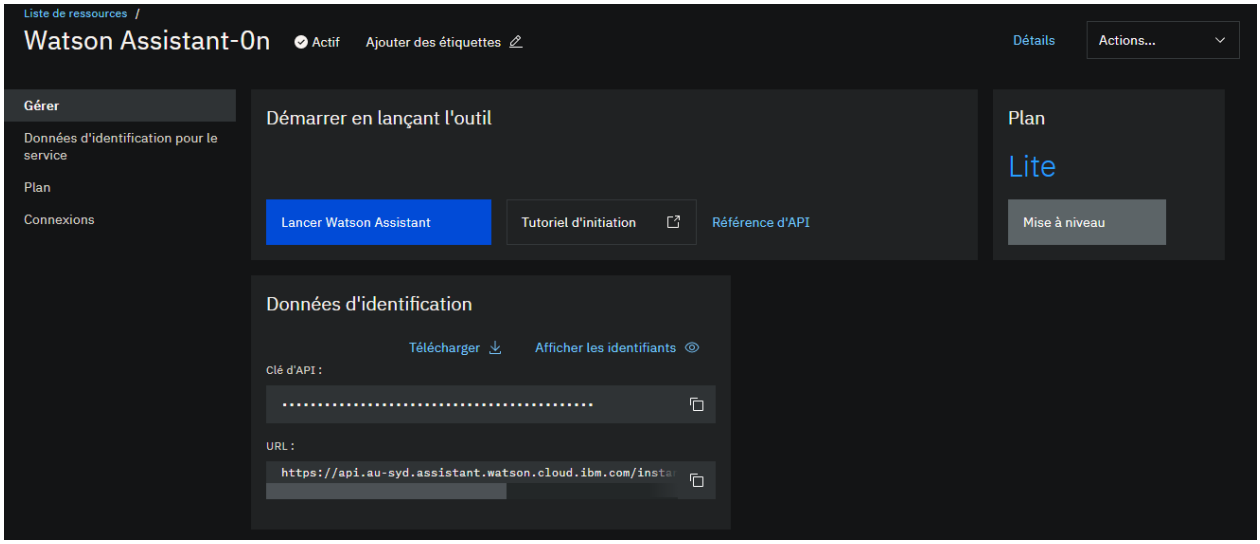


6. Click Deploy to deploy the flow.

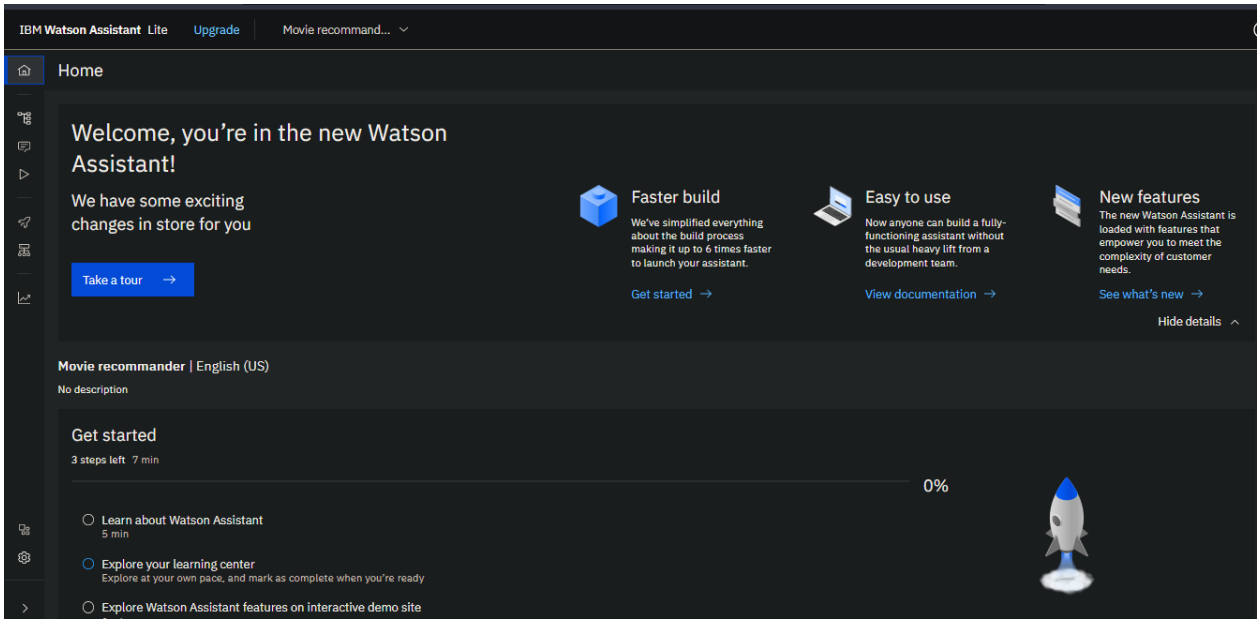


Test the entire movie recommender system.

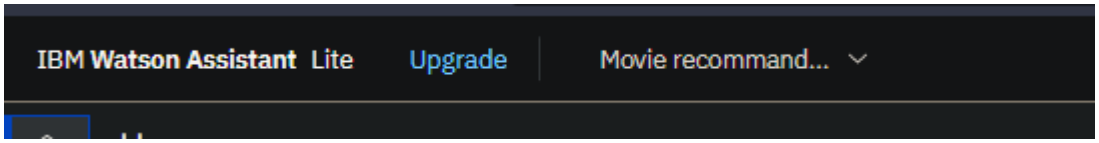
1. From the IBM Cloud Dashboard, click Resource list. Expand Services and software, then click your Watson Assistant service.



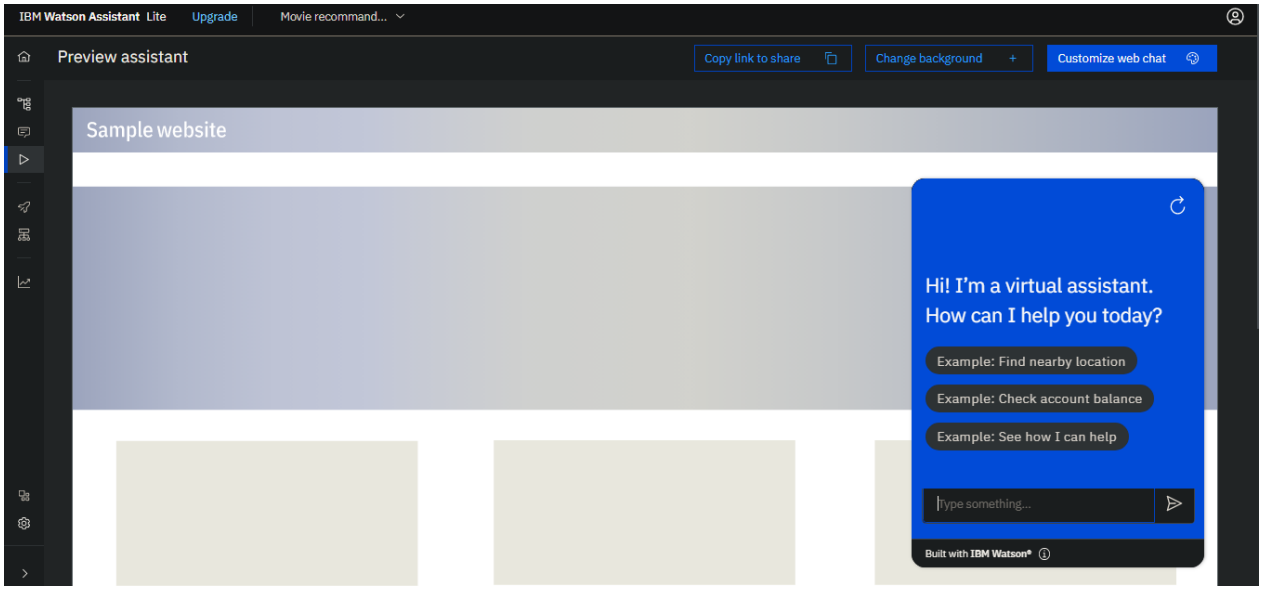
2. Click Launch Watson Assistant.



3. Select the Movie recommender assistant.



4. Click Preview from the left pane.



5. Ask the assistant “Can you recommend a movie?”. The assistant asks for your user id. Enter “10”, for example. Then, you get the list of movies recommendations.