# Hands-On Lab

## Fraudulent Storm Claim Detection

*Weather ChatBot Demo*



IBM

## DISCLAIMER

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results like those stated here.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts. In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply."
**Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.**

Performance data contained herein was generally obtained in controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products about this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.



# Table of Contents

# 1 Demo Business Scenario

## 1.1 Storms can cause chaos – for insurance customers *and* insurance companies!

Sadly, the chaos after a serious storm can be devastating. Many people who are badly affected are likely to contact their insurance company, to find basic information about how to make an insurance claim and to claim compensation for damage caused by the storm. With this sudden load, the supporting systems are often overwhelmed, and communication lines get congested with claim requests and questions.

*What do I do if my home's been flooded or damaged in a storm? How to make an insurance claim? What do I need to submit for an insurance claim? Who will assess the damage? When should I make an insurance claim?...*

In times of crisis, fraudulent requests also surge and which in turn forces insurance companies to apply tedious processes for claim evaluation. Multiple layers and checks to discriminate between fraud and genuine claims are time-consuming for both parties which usually creates frustration among genuine victims and loyal customers.

## 1.2 Avoid the chaos with integrated cognitive technology

A chatbot can play a critical role in helping insurance companies to accelerate the process of claim evaluation checks and quickly gather information about storm damage and offering smoother user experience.

IBM Watson Assistant helps you build, train, and deploy conversational interactions that takes place as a first step in claiming a storm damage, assisting the victim about the process as well as validating the data to filter fraudulent claims to a certain degree without frustrating the victim. With IBM Cloud Pak for Integration you can elevate your chatbot experience from intelligent conversational interactions to real, cognitive system interactions by integrating with multiple applications. IBM Cloud Pak for Integration allows you to orchestrate your integration flows;

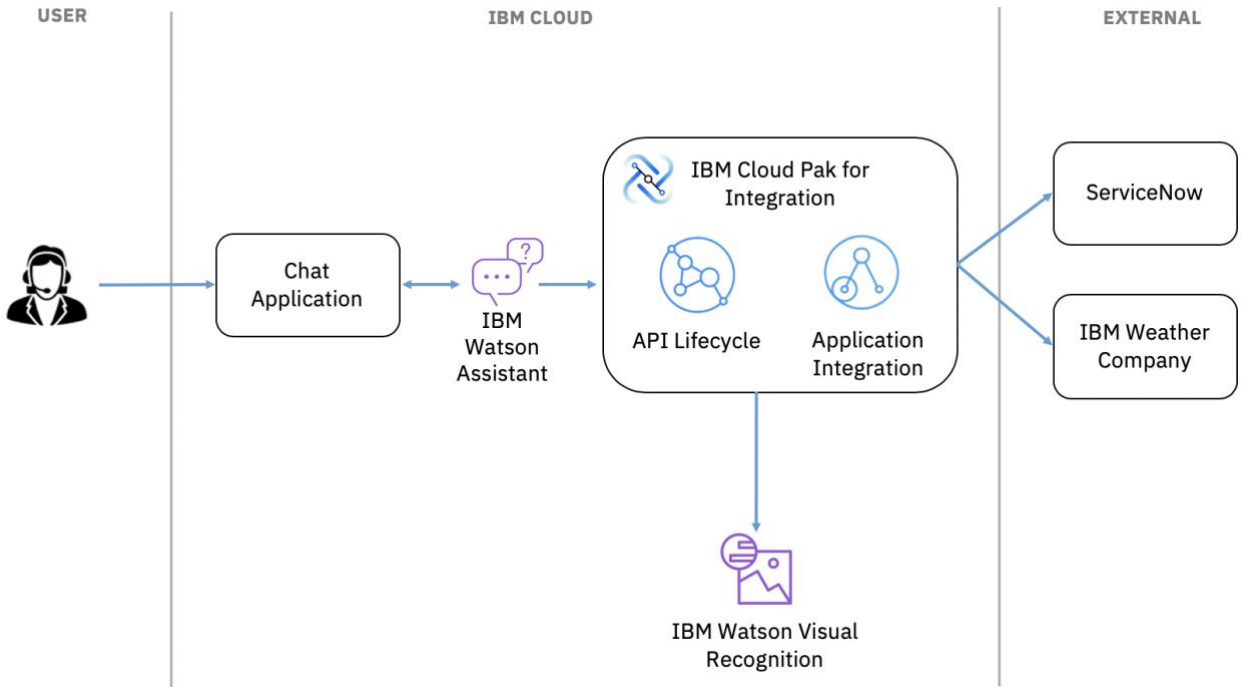
- to verify that there was a storm on the date and location provided by the victim with IBM's Weather service
- to create an online insurance claim
- to identify the items claimed for by analyzing the photographs uploaded by the victim using IBM Watson Visual Recognition and using this information to provide an estimate of value of the damage

## 1.3 The integrated solution architecture

An insurance company that specializes in 'Storm insurance' wants to streamline its claims process. They have a chatbot that verifies that there was a storm on the date and location provided by the victim and provides an initial estimate for property damage, by analyzing images uploaded by the customer and raises a provisional claim ticket for follow-up by a specialist. This will dramatically increase the efficiency of the specialist, by eliminating obviously fraudulent claims, gathering key information, and providing an initial claim estimate.

This solution shows how IBM Watson Assistant on IBM cloud can be used together with IBM Cloud Pak for Integration to create an engaging chatbot experience implemented in Node.js which allows users to make online insurance claims and also upload photographs of the items for which they wish to claim.

Three separate APIs which are independently deployable and scalable are brought together to support chatbot application. These APIs also leverages IBM Watson Visual Recognition and IBM's Weather service using built-in connectors.




1. The user visits the insurance company's site with the storm insurance chatbot application to raise a storm claim.
2. The chat application calls IBM Watson Assistant hosted in IBM Cloud.
3. IBM Watson Assistant uses natural language understanding and machine learning to extract entities and intents of the user question.
4. IBM Watson Assistant collects address and date information from the user to validate the eligibility to raise a storm claim.
5. Watson Assistant invokes the API hosted on IBM App Connect for claim validation.
6. IBM App Connect uses IBM Weather data connector to determine if the path of storm from historical data actually crosses home location and severity would warrant such damage. Based on outcome, the claim is rejected or accepted.
7. If the claim is accepted IBM App Connect uses ServiceNow connector to create a provisional claim and returns back the claim details along with validation message.
8. IBM Watson Assistant replies to the user the validation message and asks the user to upload the photograph of the damage for claim assessment. The chat application displays the chat answer to the user.
9. The user uploads the photograph of the damaged object.
10. IBM Watson Assistant application invokes the API hosted on IBM App Connect for damage assessment.
11. IBM App Connect feeds the photograph into IBM Watson Visual Recognition running in IBM Cloud.
12. IBM Watson Visual Recognition analyzes the photograph and determines the object category and maximum amount user can claim for the item.
13. IBM App Connect returns the analyses result.

IBM Watson Assistant replies to the user the analyses result along with the provisional claim details. The chat application displays the answer to the user.

# 2 Plan of Work

This solution has a number of moving parts, so we'll tackle them in a logical sequence. If you're familiar with how to do any of the steps, feel free to do them in the way you prefer or are familiar with.

If you're familiar with any of the tools we're using, feel free to embellish or change the lab –as long as you make sure it all 'hangs together'. You can build the 'extension' version if you're familiar with the tooling.

We recommend you make one journey through the lab as we describe it and then go back and explore if you have time. You can make changes and redeploy new versions afterwards.

We'll be doing the following:

- **Set up our integration systems and services endpoints**

We are going to integrate with SaaS systems and IBM Watson AI services. We will need to have these endpoints created and create credentials for so that we can integrate to them securely in the lab.
In the 'real world' systems like IBM's Weather service or ServiceNow will be running at customers already – this is a lab task.

- **Create an integration flows for our 'Storm Claim APIs'**

This will create our three separate APIs which are independently deployable and scalable and the integrations to all of our endpoints. We will create an 'integration flow' which takes the API request, calls the endpoints in the correct order, maps the data between them and sends an appropriate API response back to the caller. These APIs will leverage IBM Watson Visual Recognition and IBM's Weather service using built-in connectors.

- **Deploy the API to the Cloud Pak for Integration (CP4I) runtime**

Once we have developed our flows and tested it, we will deploy it to CP4I running on OpenShift. This will create a Kubernetes container/pod deployment with highly available replicas which we can then scale up and down as we wish.

- **Manage the API, applying security and rate-plans and publish it to our Self-Service Portal**

We will create an API which will route across the three 'Storm Claim APIs' and secure it using API keys and rate-limited API plans. This way our consumers can discover it, get the information they need to use it and preferably sign up for access in a self-service and secure manner.

- **Set up our Watson Assistant and Node.js application**

We are going to implement a chatbot using Watson Assistant and use a Node.js client application to embed the chatbot.

- **Create an ‘application’ to consume our API. We’ll use the portal to discover the API and self-service register it.**

We will register to our Watson Assistant chatbot embedded Node.js application to API server. Registering Applications allows us to assign API keys and also to monitor the calls made to the API by that application.

# 3 List of things we will need:

As this is an integration lab, we will need systems and services to integrate to:

## 3.1 List of Systems and Services Endpoints

### Service Now:

Service is an IT services management (ITSM) system provided as a SaaS i.e. it is hosted in the cloud.

In this scenario, we as a Storm Inc insurance company will use Service Now to log an insurance claim incident.

Service allows you to create developer instances/accounts free of charge. You will need a developer account to run this lab so instructions as to how to create them are included and you can set an account up as part of the lab (https://developer.servicenow.com/). If you already have a developer Service Now account, you can use that.

### IBM Weather Data:

IBM Weather Data provides historical, current and future information about Weather parameters of a particular geo location. You could connect to Weather Data using the API Key.

A free 30-day API key can be available here.

https://epwt-www.mybluemix.net/software/support/trial/cst/programwebsite.wss?siteId=443&tabId=774&w=1

### IBM Watson Visual Recognition:

IBM Watson is available on the IBM Cloud and also in the IBM Cloud Pak for Data. IBM Cloud lets you create non-expiring free instances of the IBM Watson services that you can use for this lab (or anything else)

The IBM Watson Image Recognition service lets you send a picture (.jpg, .png) to Watson and returns a list of things that Watson can 'see'.

In this use case we are using Visual Recognition to recognize the images of weather damaged objects, assess the maximum claim possible for the type of the object. You can train your Visual Recognition model with your set of images and classifiers.

### IBM Watson Assistant:

The IBM Watson Assistant service lets you build, train, and deploy conversational interactions into any application, device, or channel. Watson Assistant can be deployed in any cloud or on-premises environment – meaning smarter AI is finally available wherever you need it.

In this use case we are using Watson Assistant to build a chatbot dialog that takes place as a first step in claiming a storm damage, assisting the victim about the process as well as validating the data to filter fraudulent claims to a certain degree without frustrating the victim.

Source code is hosted on our Github repository.

## Node.js Client Application:

This project deploys a react application that connects to a Watson API. It currently runs from localhost. The application deals with text and image responses only from the Watson assistant.

Source code is hosted on our Github repository.

## Some extra things we need for this lab:

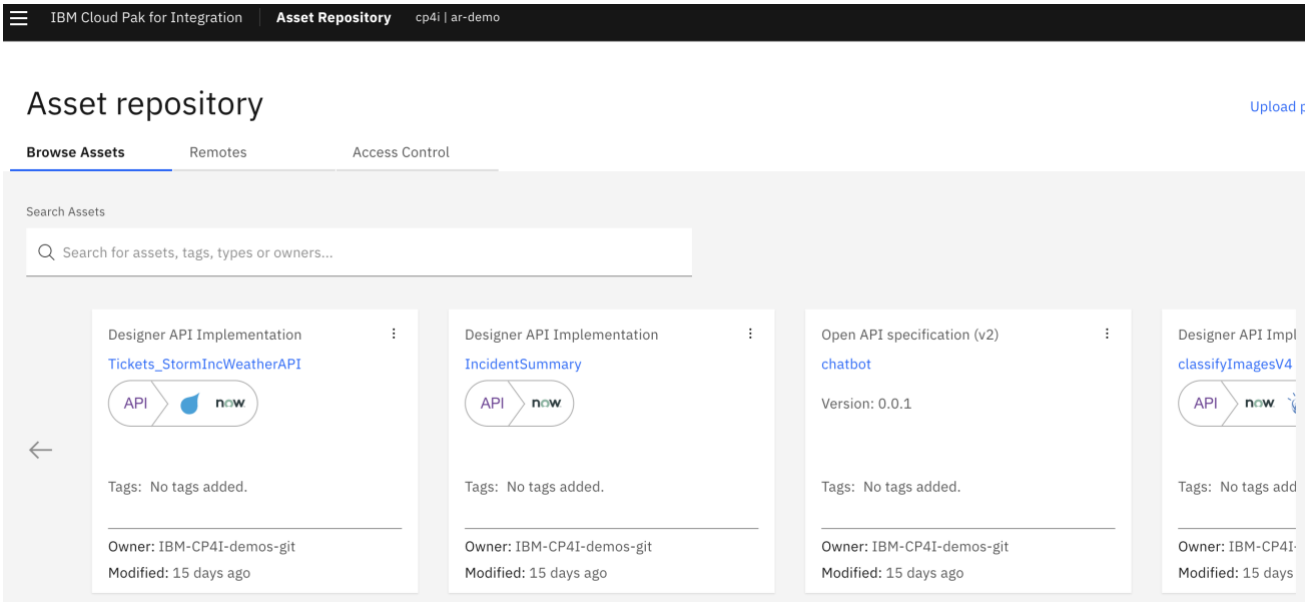
### GitHub

We've hosted our API flow definitions, our test scripts and some other things you might need on a public github repository, so they're easy to download to your lab environment.

The repo is here: https://github.com/IBM/cp4i-demos

In this way, if you want to redo this lab in your own environment you can use the assets that you need.

Also, if you are using one-click install API Connect and App Connect assets will be available to you via Asset Repository.


IBM Cloud Pak for Integration
Asset Repository
cp4i | ar-demo
Asset repository
Upload
Browse Assets
Remotes
Access Control
Search Assets
Search for assets, tags, types or owners...
Designer API Implementation
Tickets_StormIncWeatherAPI
API
now
Tags: No tags added.
Owner: IBM-CP4I-demos-git
Modified: 15 days ago
Designer API Implementation
IncidentSummary
API
now
Tags: No tags added.
Owner: IBM-CP4I-demos-git
Modified: 15 days ago
Open API specification (v2)
chatbot
Version: 0.0.1
Tags: No tags added.
Owner: IBM-CP4I-demos-git
Modified: 15 days ago
Designer API Impl
classifyImagesV4
API
now
Tags: No tags add
Owner: IBM-CP4I-
Modified: 15 days

# 4 Getting Started – Setting up the endpoints

Before we can build our API integration, we need to set up the endpoints that we need will integrate to.

This lab doesn't use emulators, shims or stubs – we're going to connect to real endpoints in the real-world cloud! You can use these endpoints after this lab to explore more with CP4I and to do other demos – they're your endpoints to 'take home and keep'!

All of these endpoints have been deliberately chosen for this lab as they have free versions that we can use – don't worry, you won't need a credit card to create them.

We will set up endpoints to connect to:
• IBM Watson Image Recognition
• ServiceNow
• IBM Weather Company

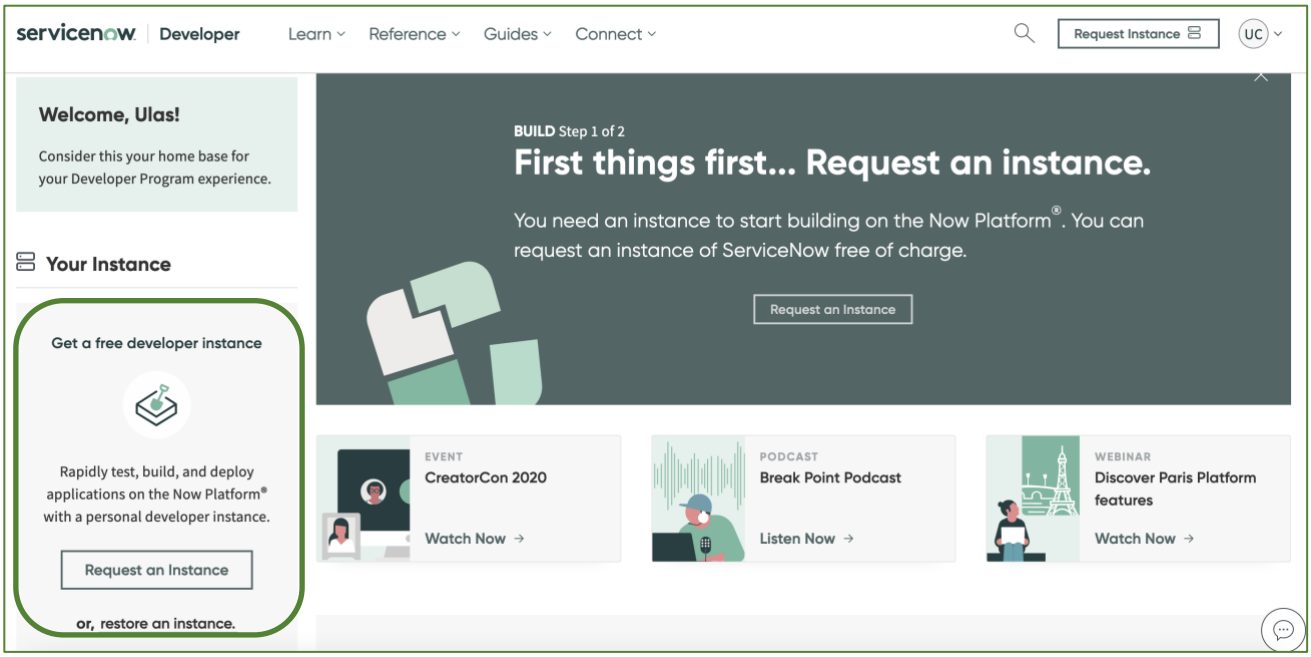
(If you already have Watson Services and a ServiceNow Developer Account e.g., you can skip this section)

Later, we will connect to the following endpoints.

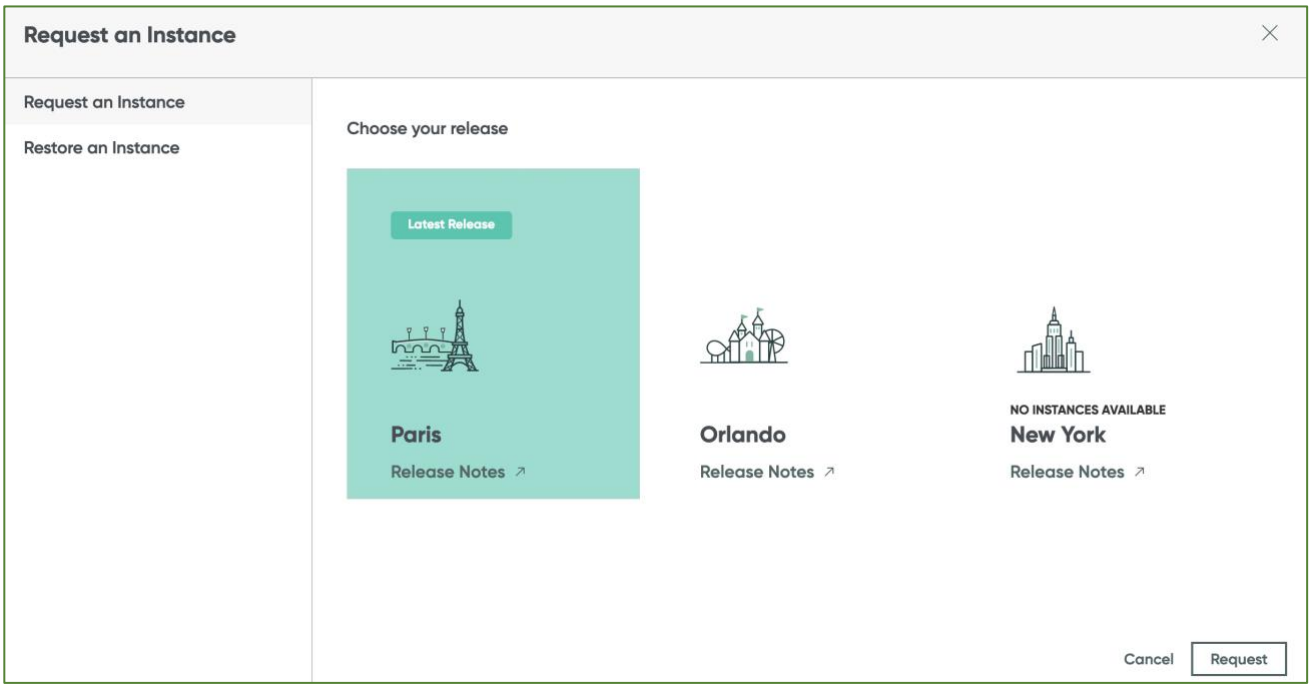
## 4.1 Setting up the Service Now

To get started you will require admin level access to your ServiceNow account. If you want to create a free ServiceNow account to test out App Connect, you'll have to register (https://developer.servicenow.com/) for a ServiceNow Account. Once your account is activated, you can request a ServiceNow personal developer instance.

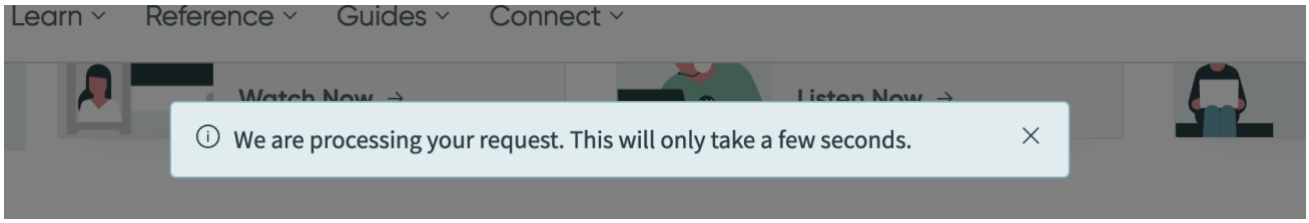
Once your account is activated, you can request a ServiceNow free developer instance.



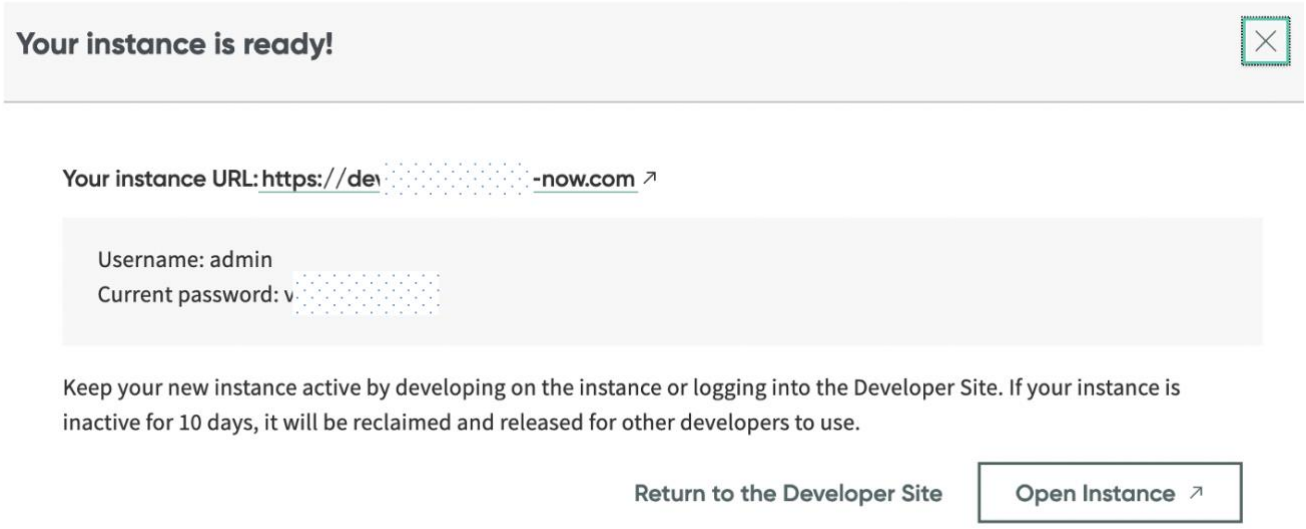

Click on ‘Request an Instance’ and select the location of your instance.



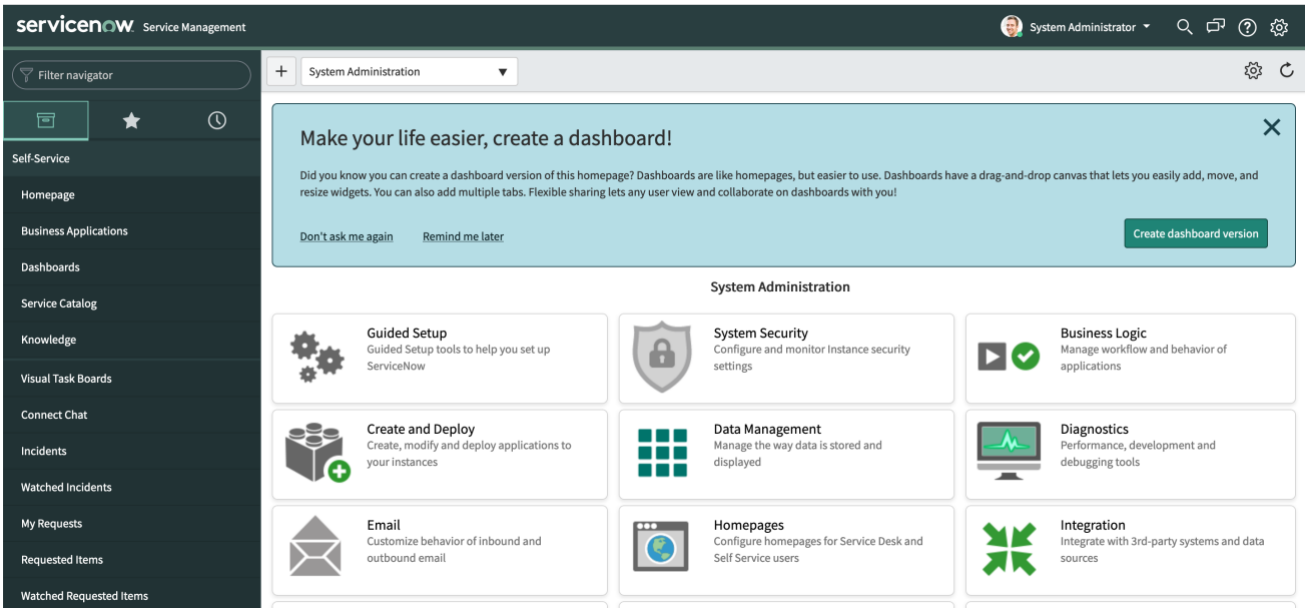

Click on ‘Request’. This will only take a few seconds.



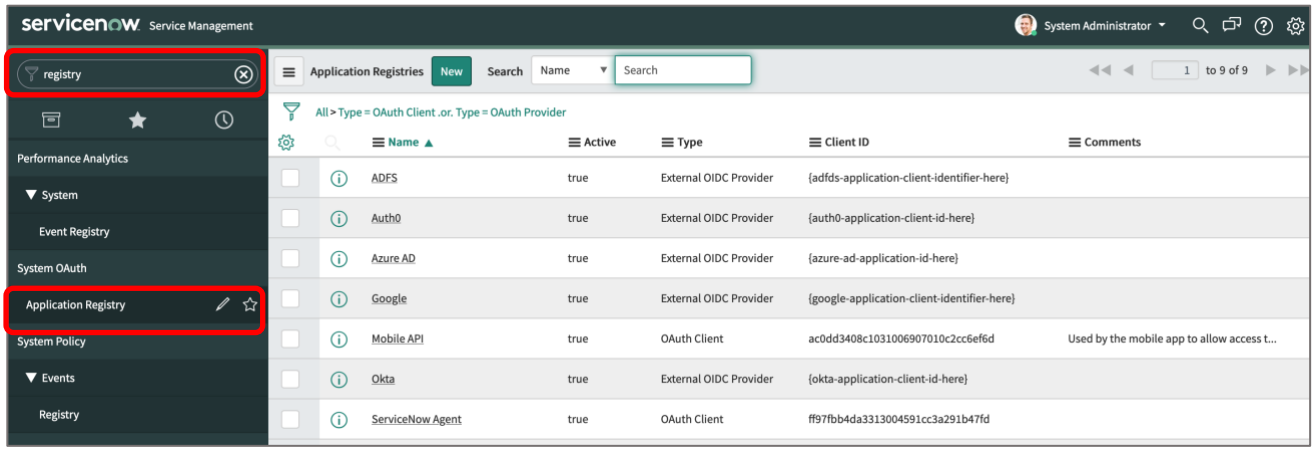
Please note your instance URL and credentials. You will need these later when we configure the ServiceNow connector.



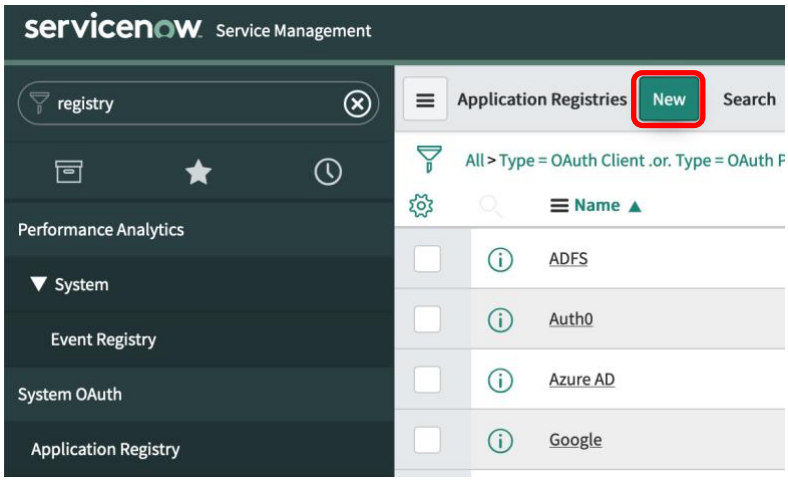

Click on ‘Open Instance’



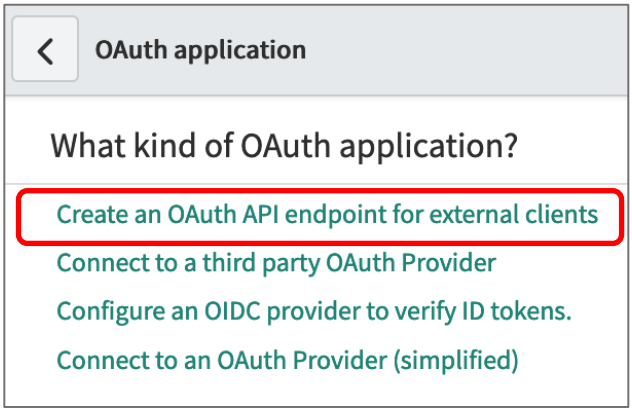

Search for Registry in ‘Filter Navigator’ search bar and then select ‘Application Registry’.



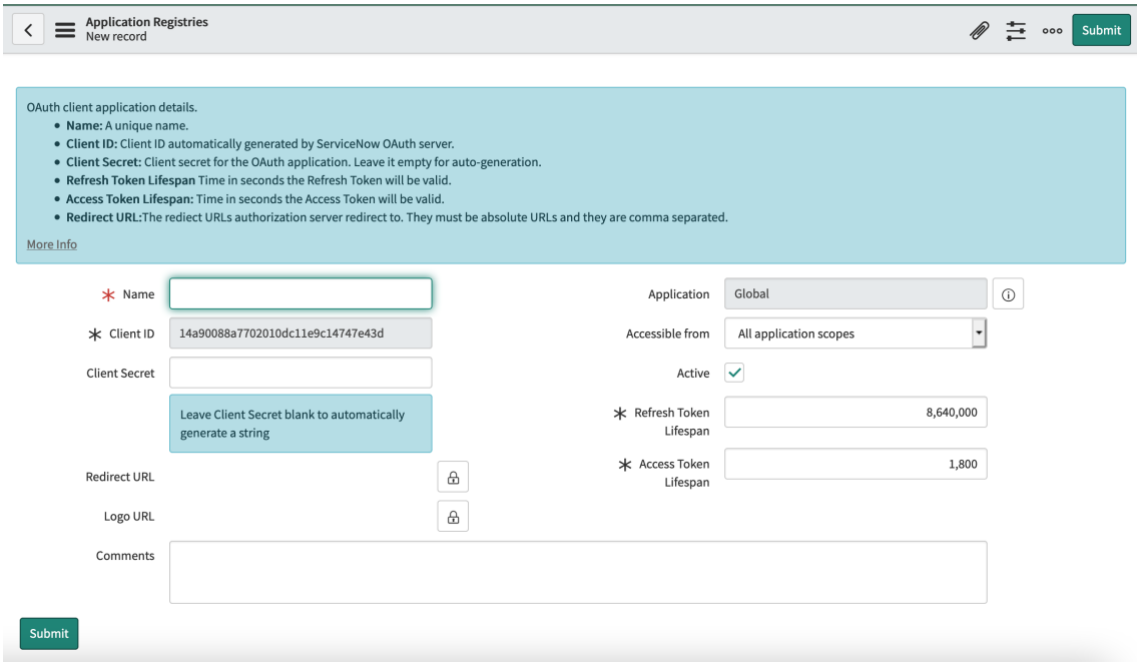

We will create an OAuth API endpoint for external clients. Click on ‘New’.



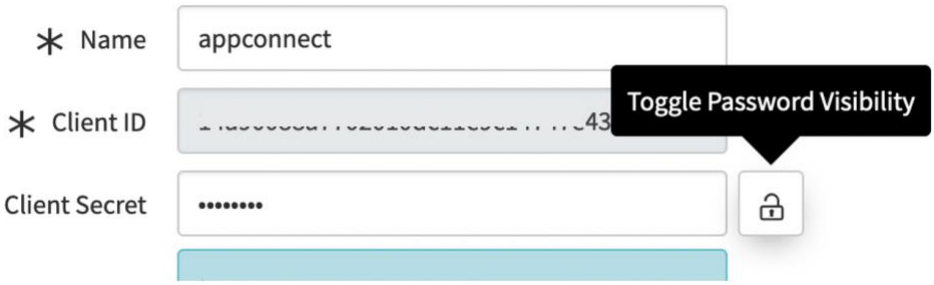

Select ‘Create an OAuth API endpoint for external clients’.




In the config panel give it a unique name and hit submit.




This will create a new OAuth endpoint with Client ID and Client Secret generated. You can view these details by clicking and viewing the new endpoint. You will be providing Client ID and Client Secret




Great! We've now a running ServiceNow instance with an Oauth app/endpoint registered for it which represents our App Connect!

Please keep in mind that this is a free developer instance, it will be reclaimed in 10 Days and also to support the developer program, instances hibernate when they are idle. But you can easily restore instance from the reclaimed instance's backup. Similarly, you can easily wake up a hibernating instance. (Learn more; https://developer.servicenow.com/dev.do#!/guides/orlando/now-platform/pdi-guide/understanding-pdis)

You'll be later asked for the following details about your ServiceNow instance and Oauth endpoint to integrate with the service. Please keep them safe.

- The URL of your ServiceNow instance, in the following form: https://<servicenow-id>.service-now.com/
- The username and password that you use to log in to the instance.
- ClientID and Secret of your ServiceNow application (Oauth endpoint).

## 4.2 Setting up the IBM Watson Services

You will need an IBM Cloud account to do this. You can use your existing one if you wish or you can set up a new one.

IBM cloud access is free and can be provisioned instantly.

Once you have an account, all of the Watson services have 'lite' plans which allow you to use them for free – the only restriction is the number of calls you can make per month. Don't worry, we won't be getting anywhere near that number – and you won't get charged if you hit the limit, it will just stop working until the next month.

### Logging in to IBM Cloud

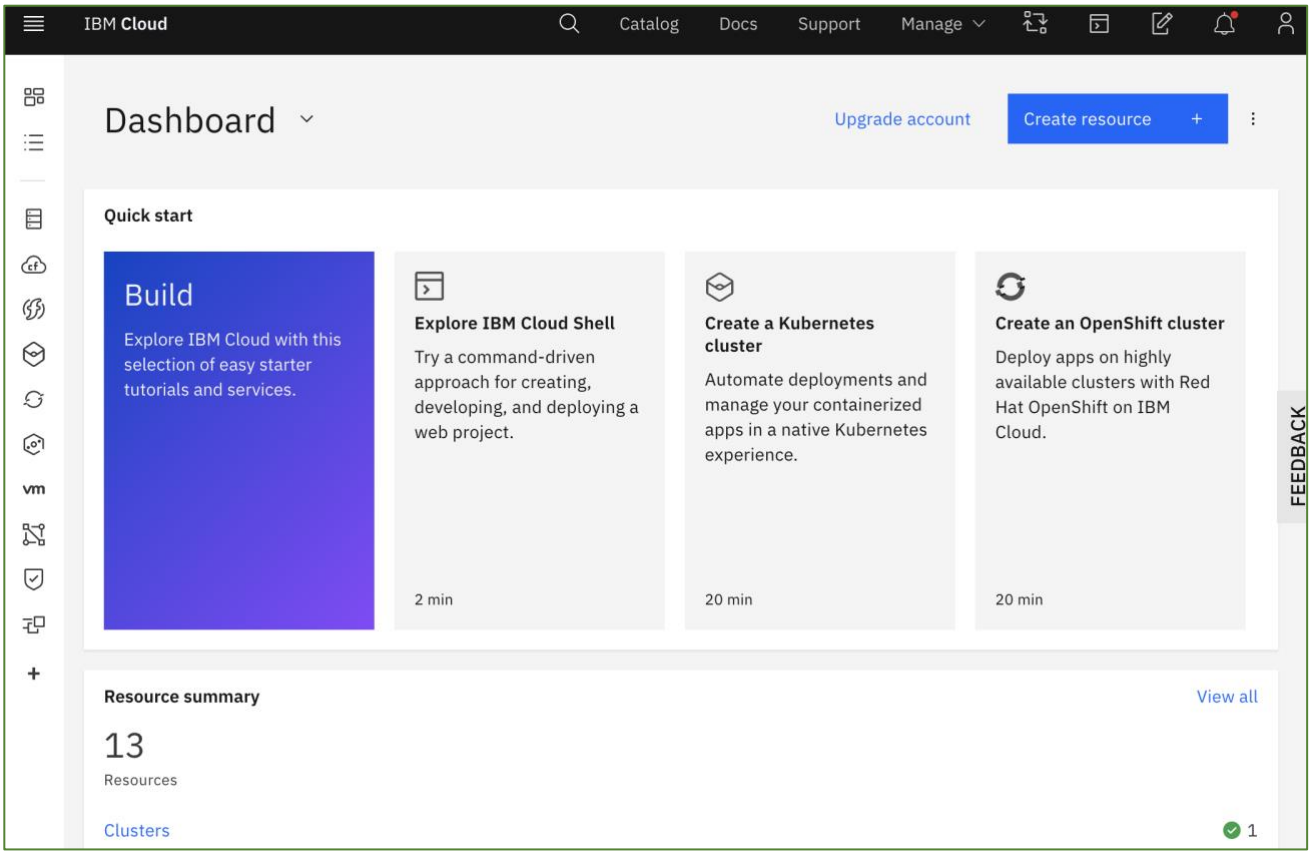
The IBM Cloud can be accessed at https://cloud.ibm.com.

If you've not been to the IBM cloud recently, the UI has just had a makeover, so yes, you are in the right place if you don't recognize it.

If you don't have an IBM ID then click 'Create an account' – all you need is an email, you don't need a credit card.

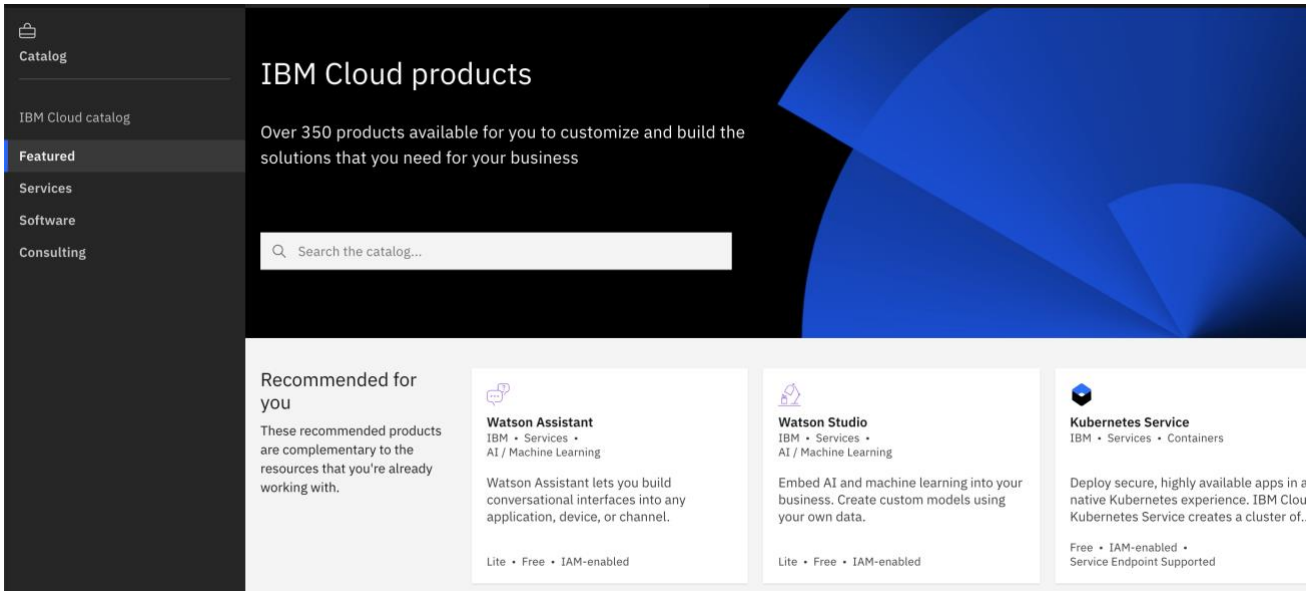
When you have an IBM ID, sign in at https://cloud.ibm.com (Depending on your company, e.g. if you're an IBMer, you may go through a Single Sign-on process).

Once you're in, you'll be presented with the cloud dashboard showing which services you have provisioned:



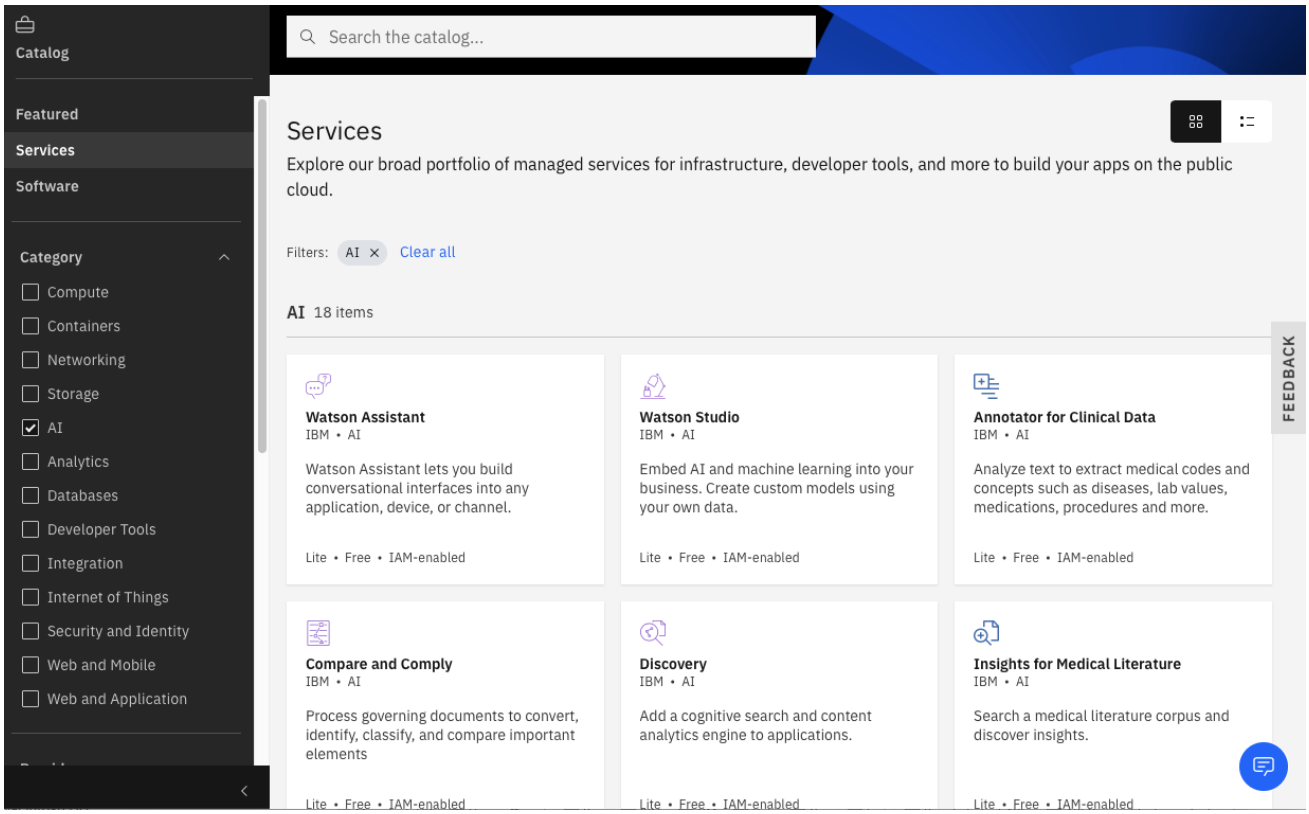

### 4.2.1 Creating your free Lite-Plan IBM Watson Visual Recognition

On the IBM Cloud Dashboard, click ‘Catalog’.

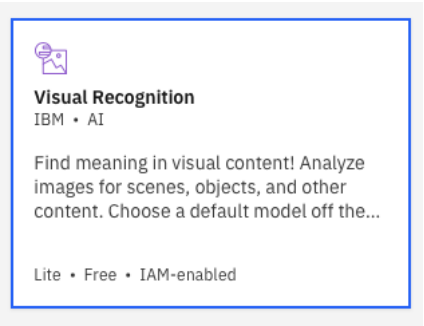



You’ll see a list of services (if not, click on ‘services’).

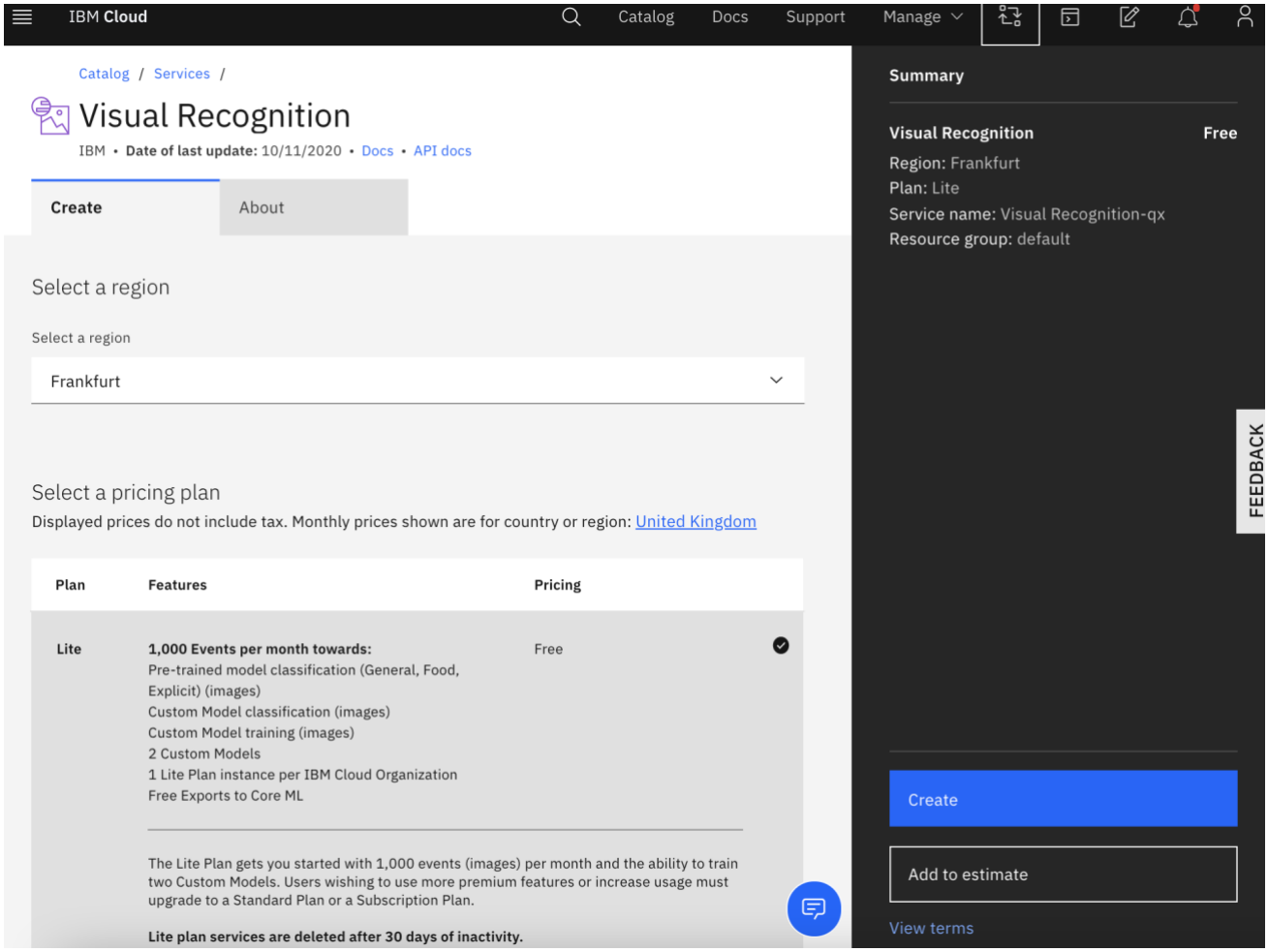
Check the ‘AI’ filter checkbox on the left to filter for Watson services. (You can also search for them by name)




Scroll down and click on the ‘Visual Recognition’ tile.



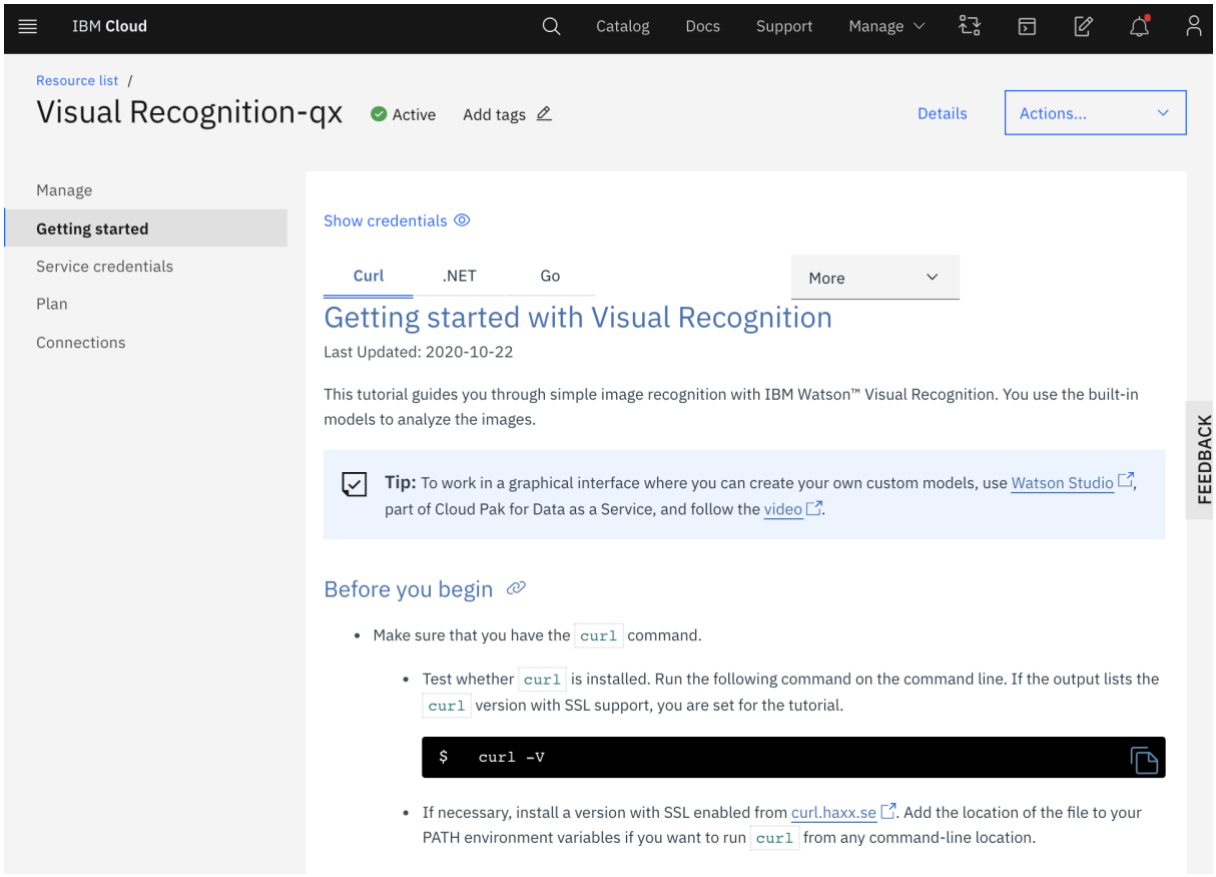

Inside, you’ll be able create a lite plan (free) instance as shown in the screenshot below.




Select the free ‘lite’ plan and provision the service.

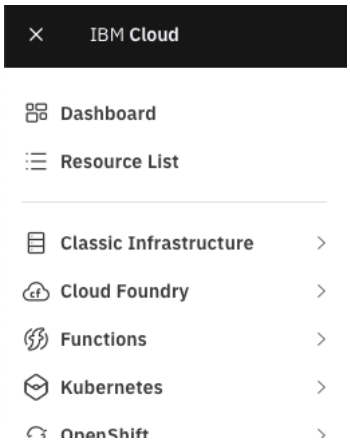
You can change the service name to something more memorable if you wish.

Once you create the service, you will be redirected to ‘Getting Started’ page of your service where you can learn the basics and go through tutorials.

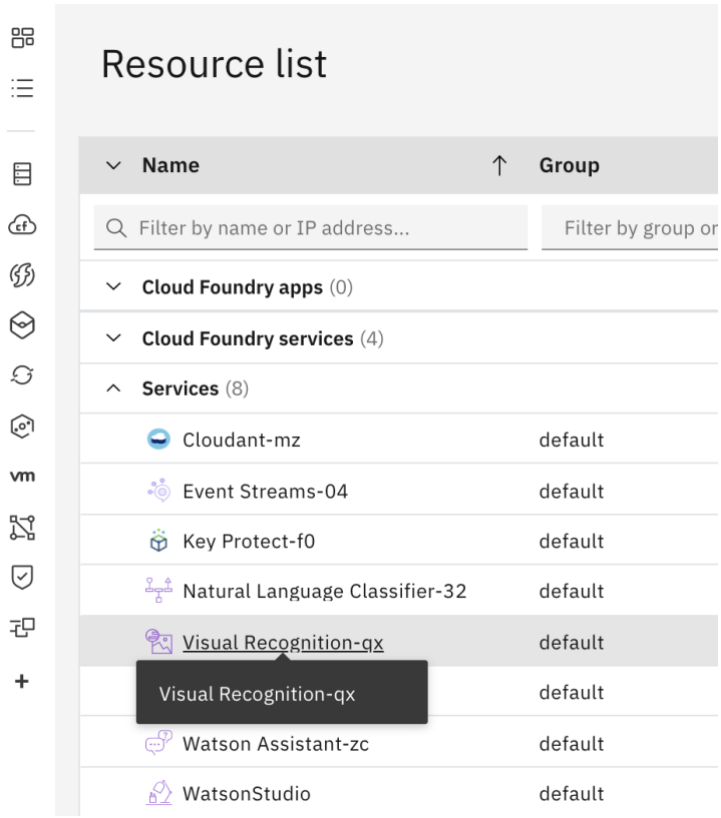



Or, you'll be able to access the service later from your cloud dashboard (to get to the cloud Dashboard,

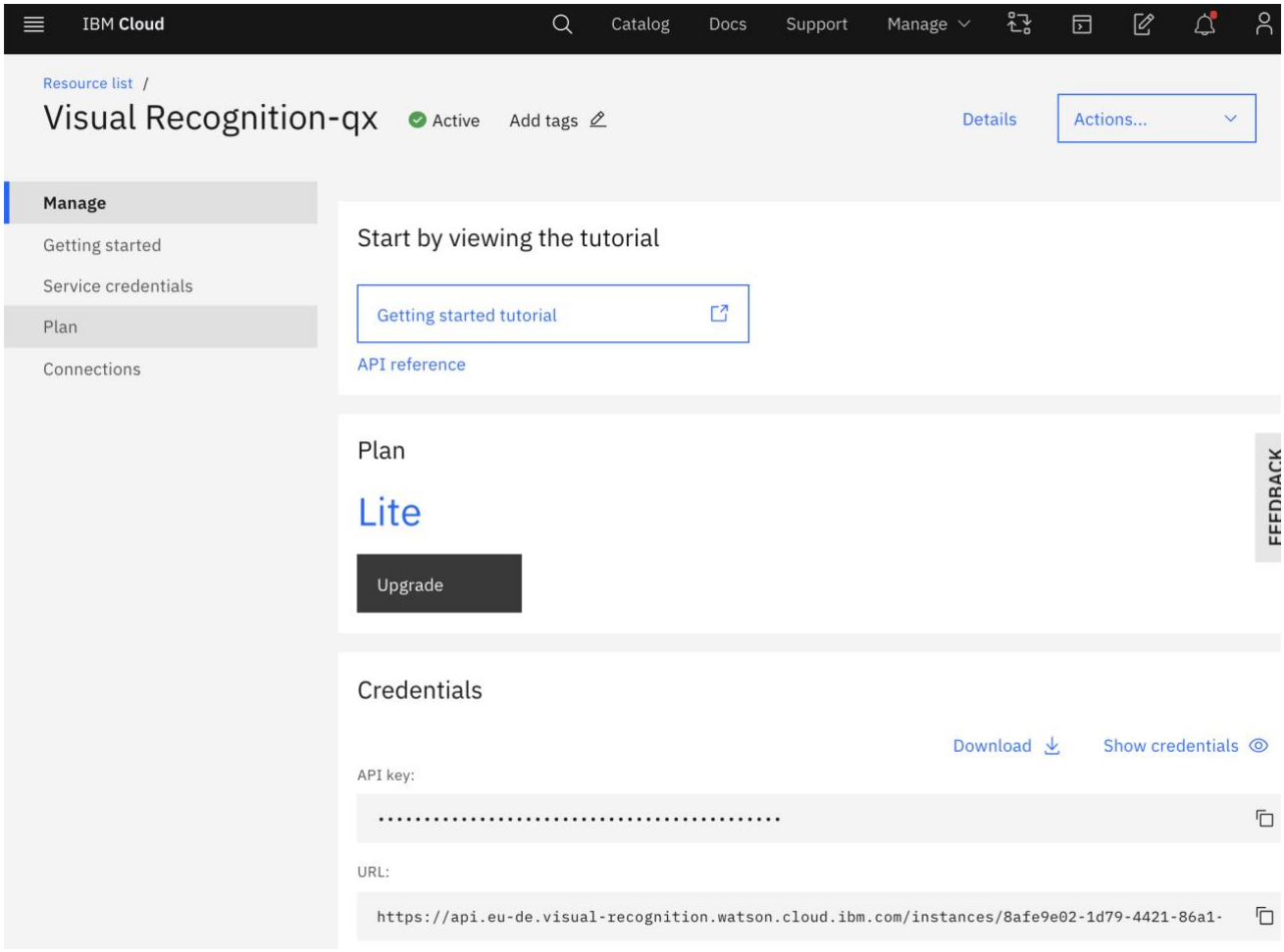
click the 'hamburger' menu at the top left of the screen and select 'Dashboard')




Look under 'Services' and you'll find your newly created service.



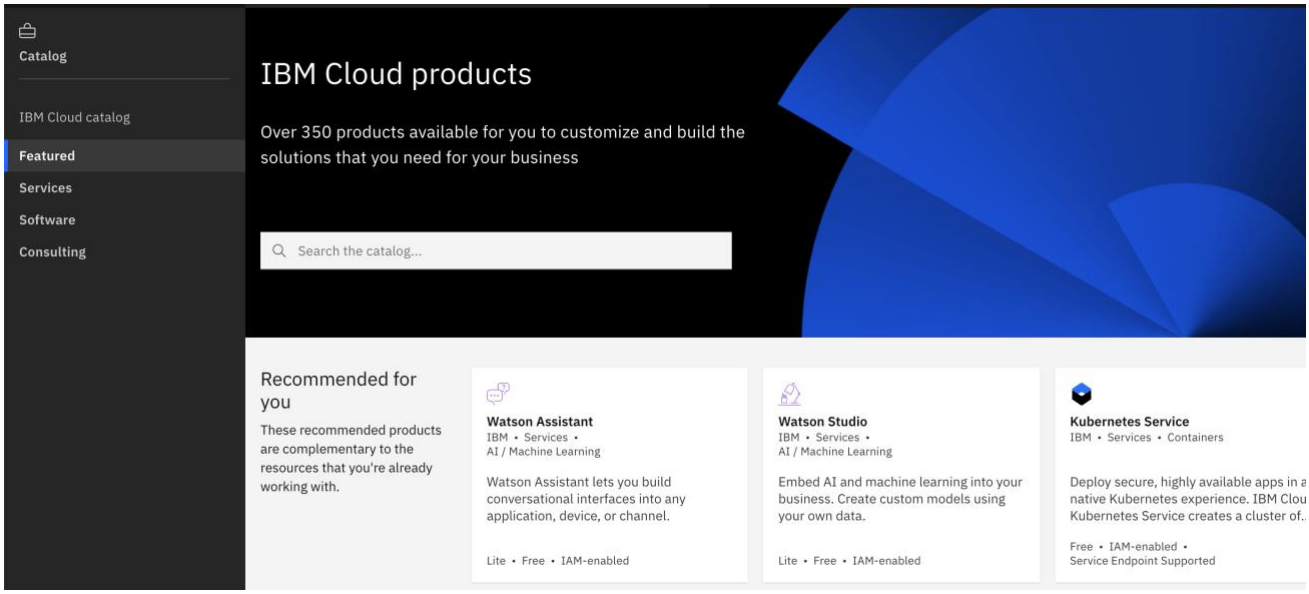

Click on your new service and you'll see the 'Manage' tab:




The API key and URL are what we are going to need to integrate with the service. You can click ‘Show credentials’ and copy/paste them somewhere for later use in this lab or you can click ‘Download’ and they will be downloaded as a text file for you.

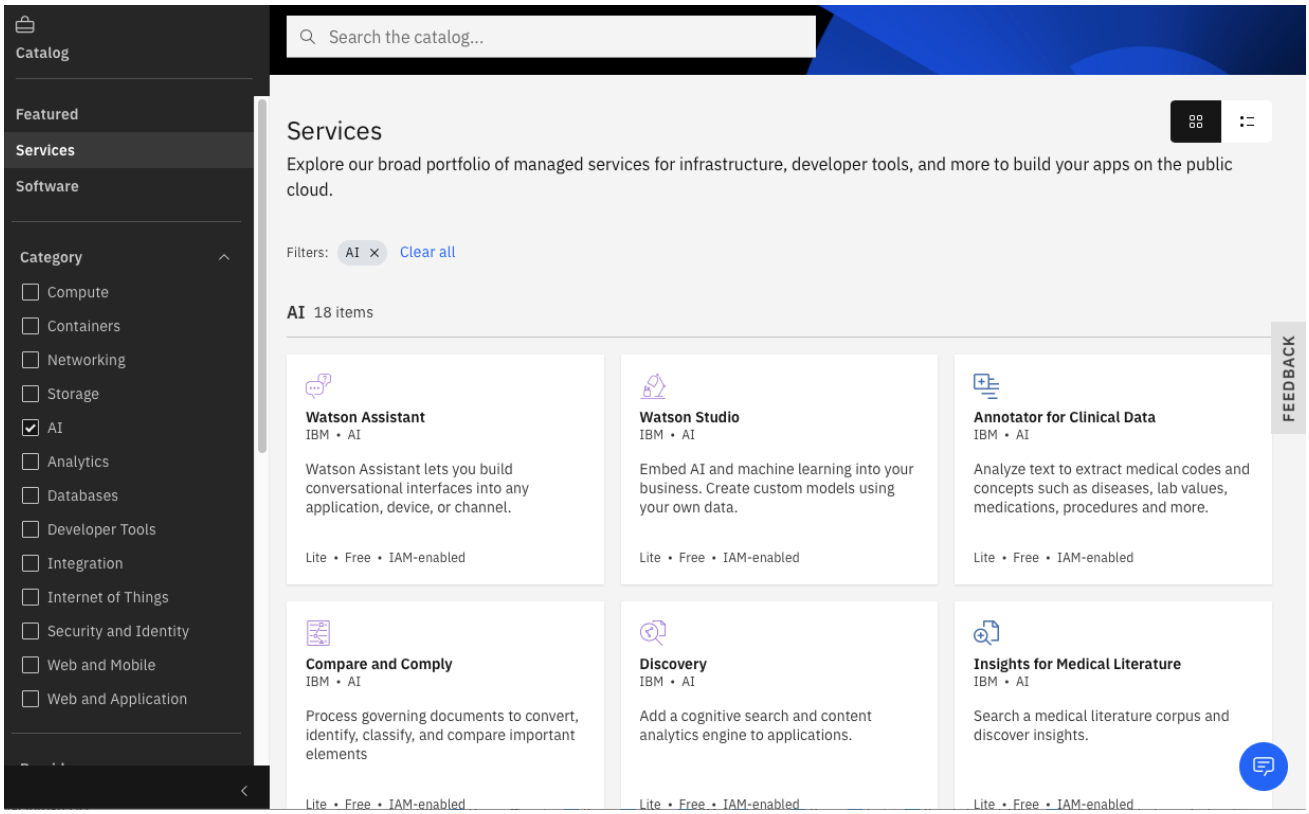
### 4.2.1 Creating your free Lite-Plan IBM Watson Assistant

On the IBM Cloud Dashboard, click ‘Catalog’.

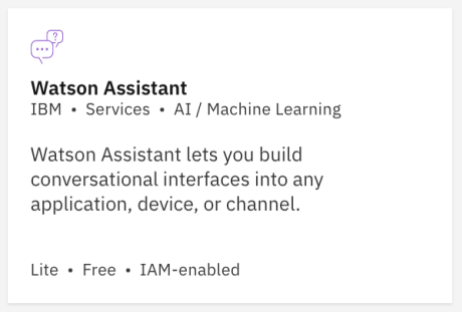



You'll see a list of services (if not, click on 'services').

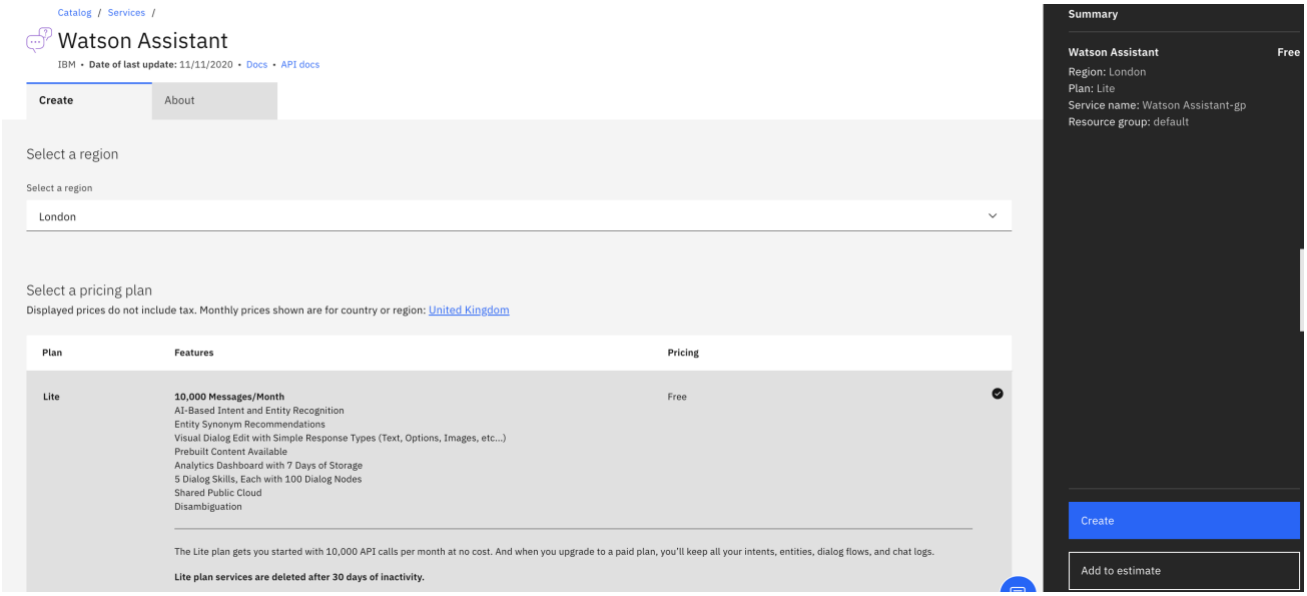
Check the 'AI' filter checkbox on the left to filter for Watson services. (You can also search for them by name)




Scroll down and click on the 'Visual Assistant' tile.

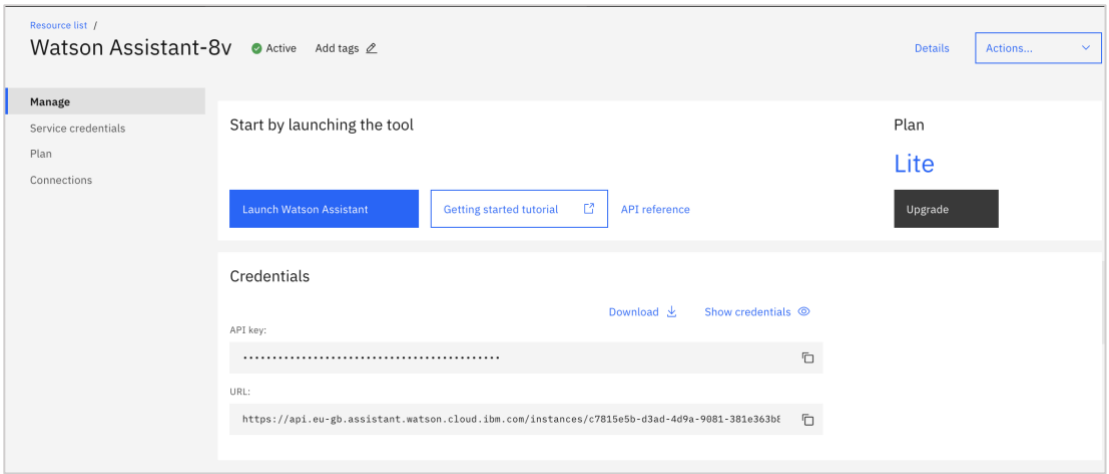



Inside, you'll be able create a lite plan (free) instance as shown in the screenshot below.




Select the free 'lite' plan and provision the service.

You can change the service name to something more memorable if you wish. Once you create the service, you will be redirected to home page of your service.




Congratulations, you've provisioned a free Watson Assistant service. We will be later using this service to create a chatbot.

IBM

## 4.3 Setting up the Weather Data Connector

You can use App Connect to retrieve details on the following objects:

- Historical Data
- Forecast
- Near Locations
- Locations
- Location by Point
- Current Conditions

To use App Connect to integrate IBM Weather Company Data with your other applications, you need to connect App Connect to your IBM Weather Company Data account. To do that, you'll need to provide the following information:

**IBM Weather Company Data API key.**
Claim your 30-day free API key from here - https://epwt-www.mybluemix.net/software/support/trial/cst/programwebsite.wss?siteId=443&tabId=774&w=1

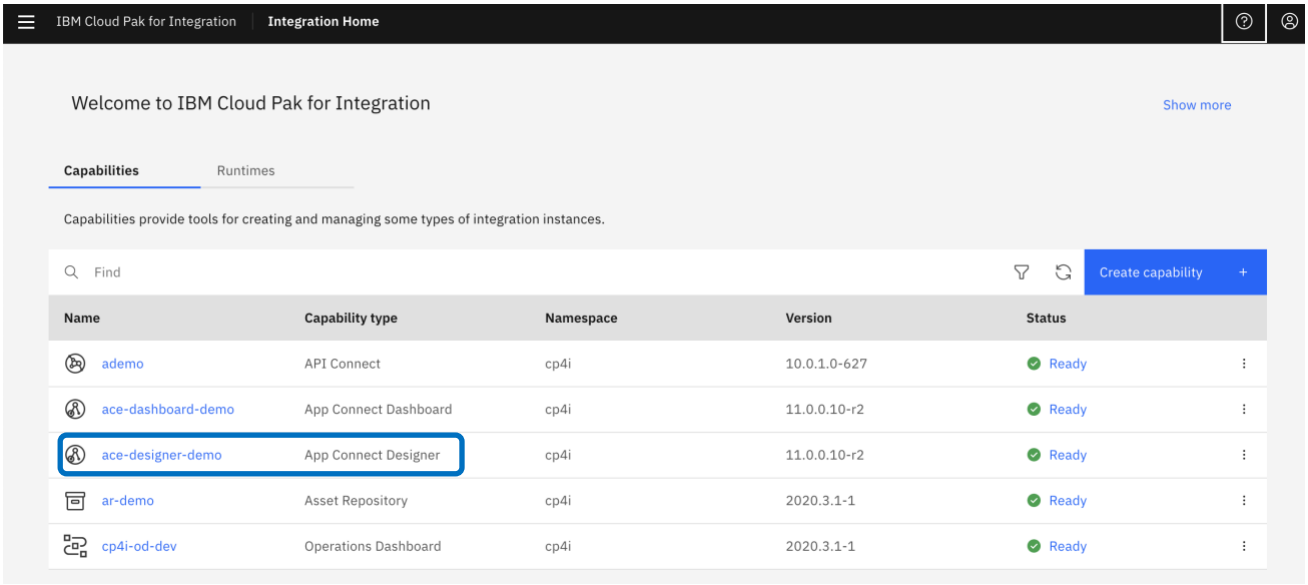
# 5 Getting into the Cloud Pak and Building Your Integration

Getting started with Cloud Pak for Integration (CP4I) on Red Hat OpenShift Kubernetes Service (ROKS) on IBM Cloud has never been easier with one-click install. The following guide walks you through how to deploy Cloud Pak Integration on ROKS cluster. https://ibm-garage-tsa.github.io/cp4i-demohub/cp4i-on-roks/

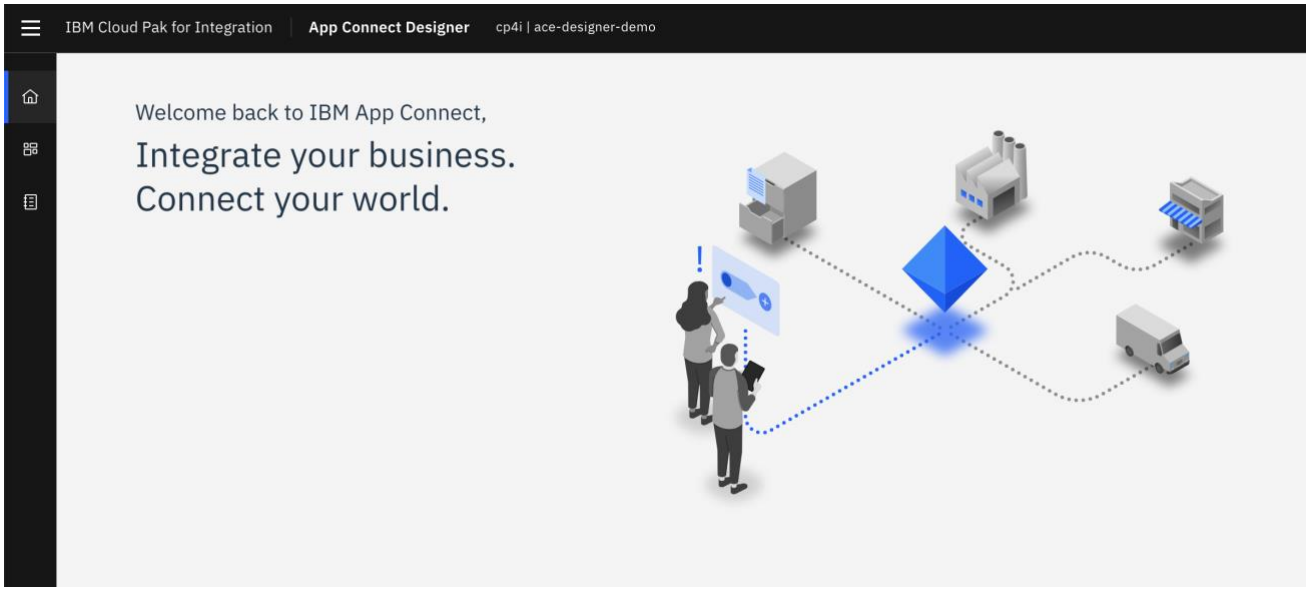
We're going to be using API Connect, App Connect and the Asset Repository for this lab.

## 5.1 Accessing the Designer Integration Tooling

For either method, menu or instance view, click on 'ace-designer-demo' which is our instance of the designer tooling for this lab.




You'll arrive at the App Connect Designer here:




This is where we can create all of our API integration flows and also manage our connectivity to our services and endpoints. You can create many integration flows and manage them all here.

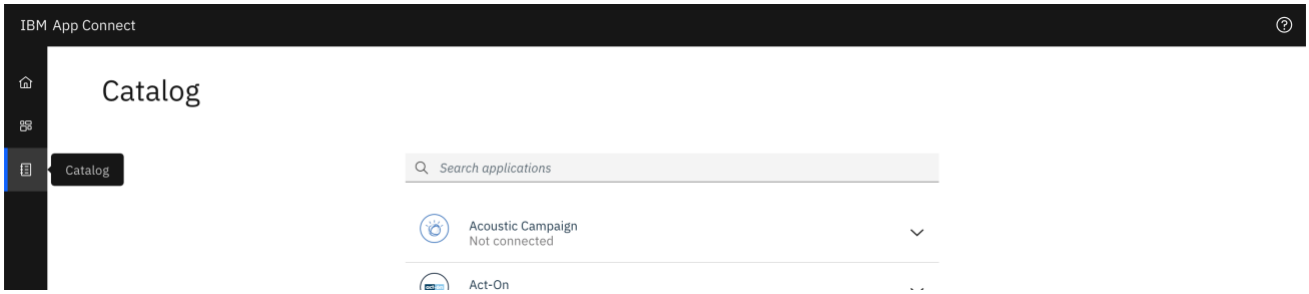
## 5.2 Connecting the tooling to our endpoints

Let's go to the connector catalog:

The connector catalog appears with a list of the cloud pak connectors which are installed locally to this lab. There are many more connectors available although not all all of them run 'locally'. Some of the connectors are currently available in the pak locally, all of them are available on the IBM cloud – you can use the ones that run on the IBM cloud directly from ICP4i designer as well – you just need to link to your IBM cloud account, which we won't be doing in this lab.

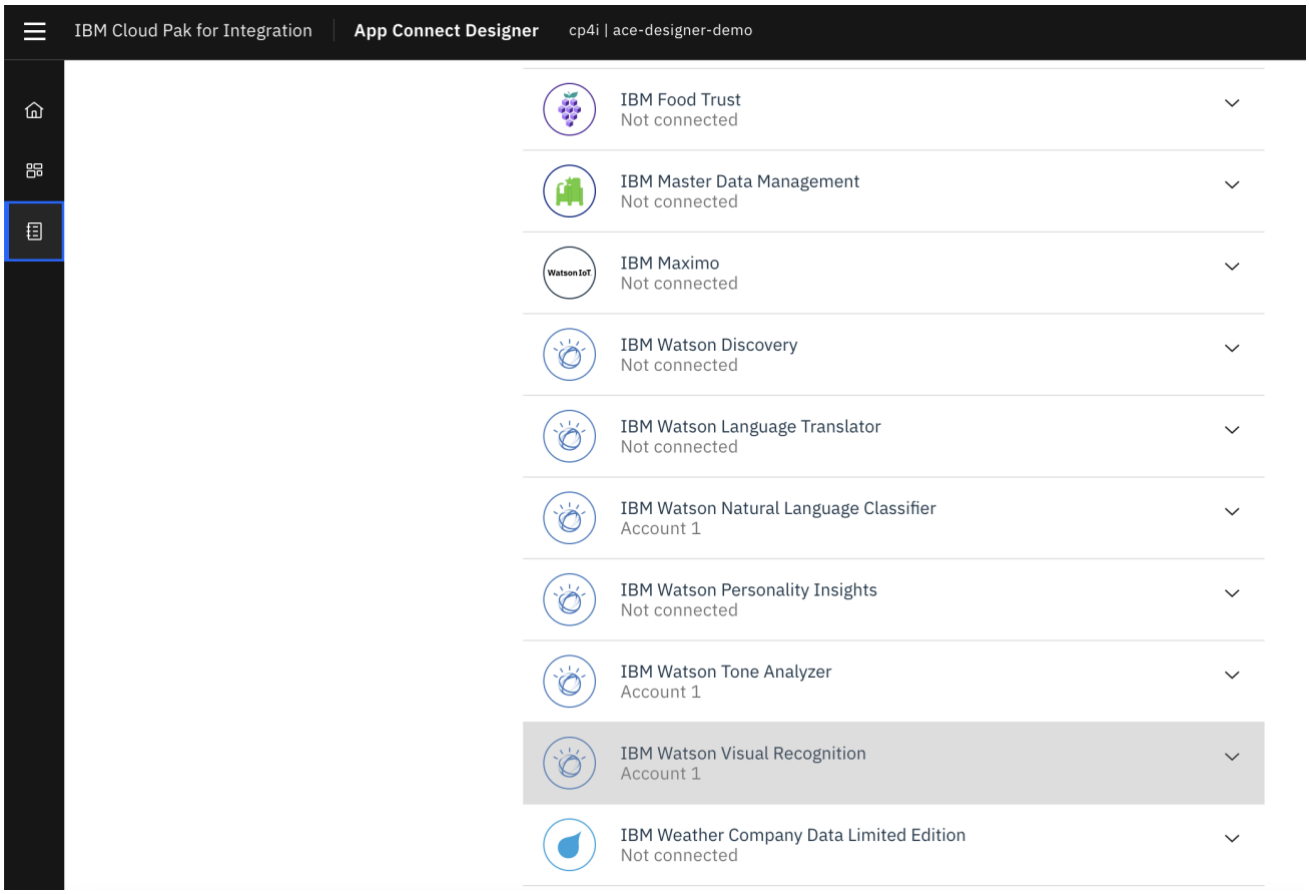
More connectors are being developed constantly – for a list, look here:
https://www.ibm.com/cloud/app-connect/connectors/

You can choose whether you want to run the connectors locally or on the IBM cloud. For this lab, we will run them locally:

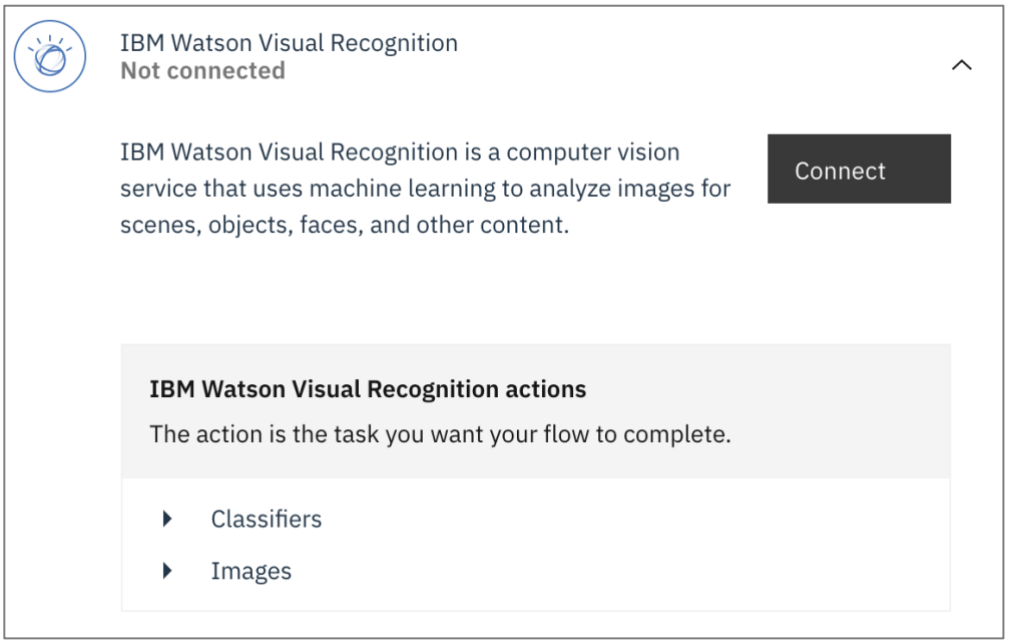


### 5.2.1 Connect to IBM Watson Visual Recognition

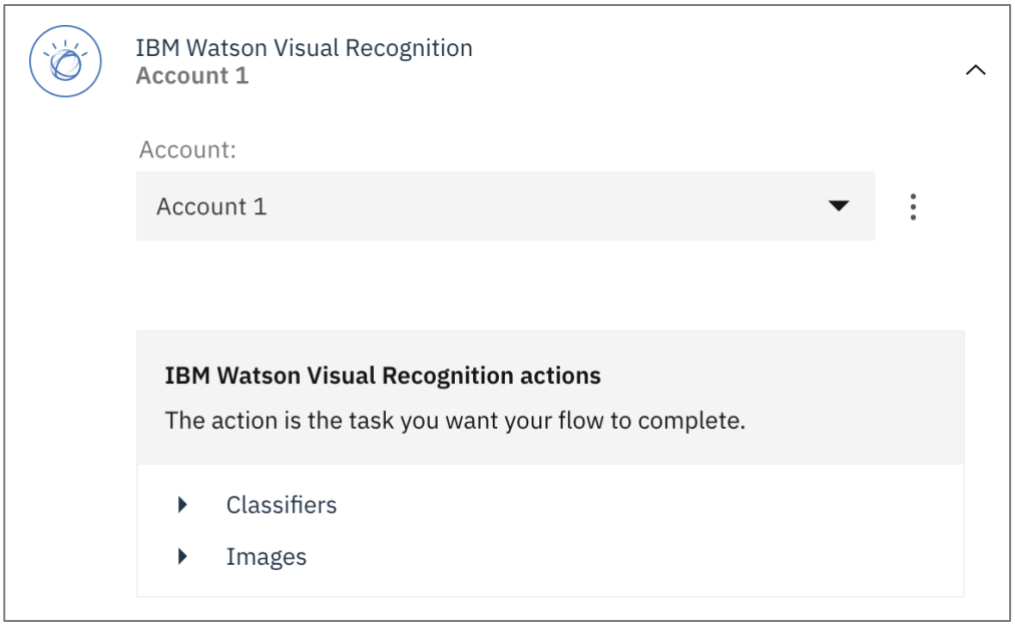
Let's set up our Watson AI endpoint – scroll down until you see the IBM Watson connectors:




Click on 'IBM Watson Visual Recognition'. You'll see that the connector expands and shows you the actions available for the connector.  CP4I connectors are smart connectors and are metadata driven – you don't need to know what functions and data are in the endpoint – the connectors will usually show them to you.

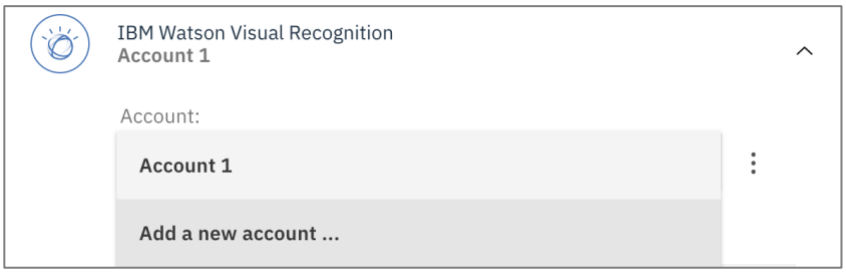



Click on ‘Connect’

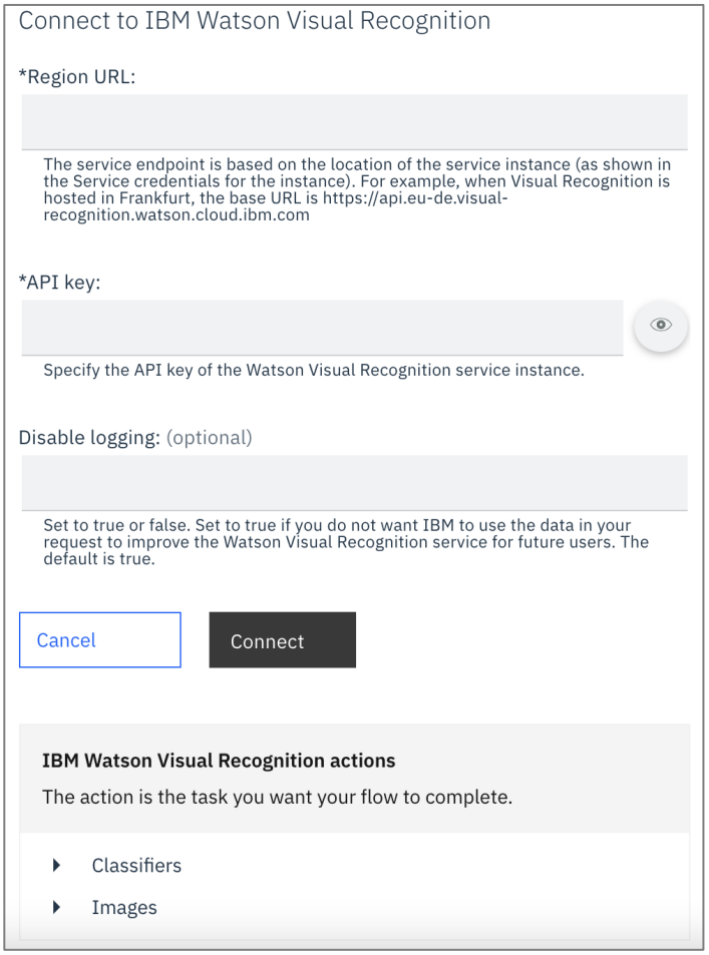



Note that if you have already an account associated with the connector, you can click on ‘Add a new account’. You can have multiple accounts connected to each application or API. You can also rename your App Connect accounts for ease of identification, update the credentials for your accounts when necessary, and remove accounts that you no longer need.

The ability to connect to multiple accounts enables you to create flows that use different accounts to connect to different instances of an application; for example, test and production instances, or instances in two different sites or geographies.



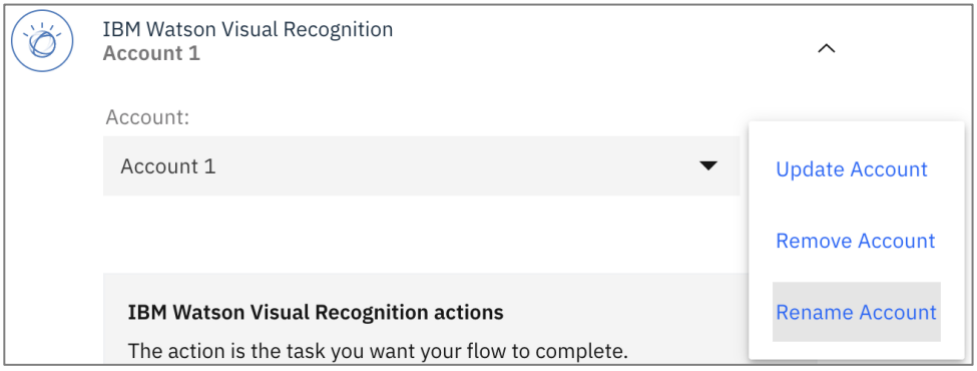

To connect to your Watson Visual Recognition account, you'll need credentials – otherwise anyone could connect to it. The service is protected by an API key. You'll now be asked for the API key that you kept safe from before: Enter it here and click 'connect'.



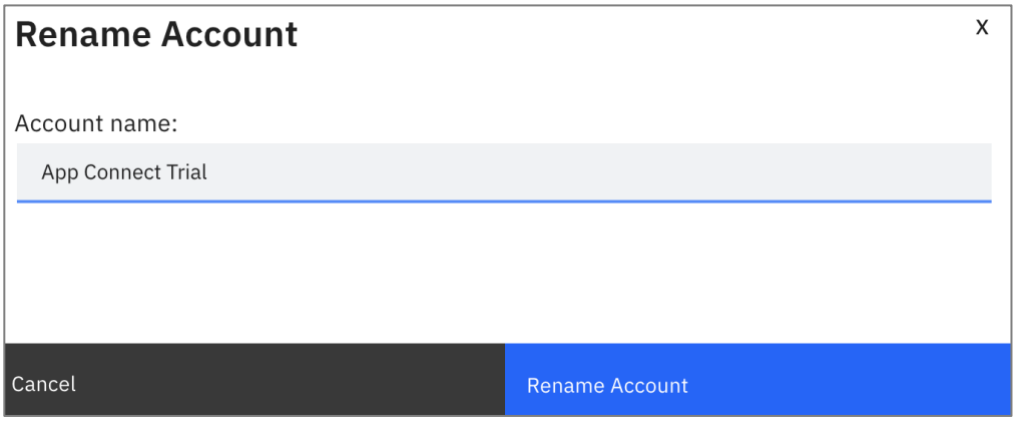

(Hint: you can use the 'eye' button to show the API key to check it's correct)

IMPORTANT: DON'T MOVE ON YET! You'll see 'Account 1' as the name of the account. WE NEED TO RENAME THE ACCOUNT FOR THE LAB TO WORK SEAMLESSLY.

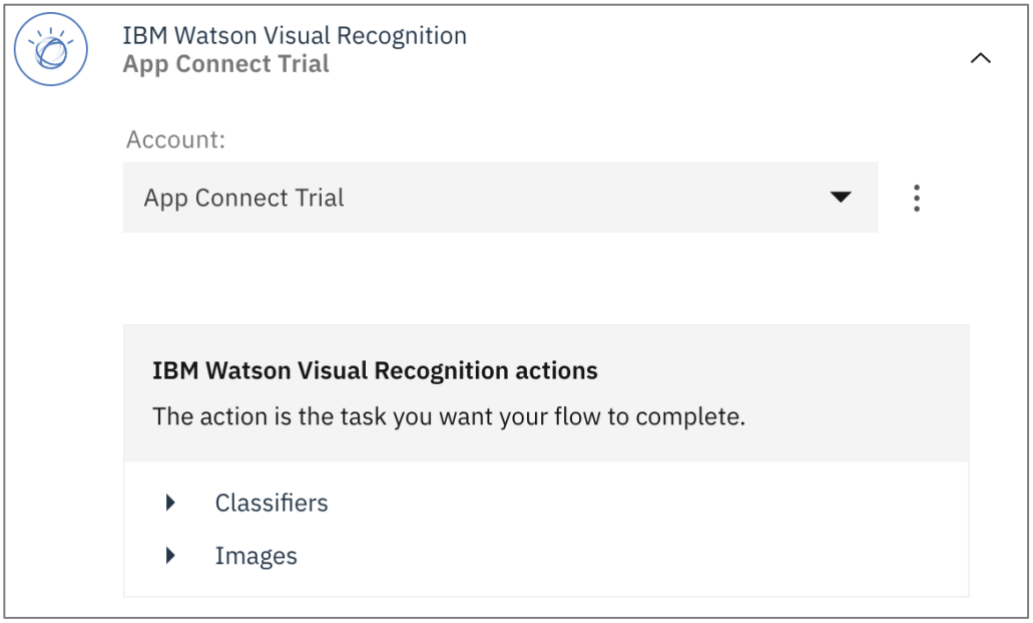
To rename your account, Click the three dots menu and click ‘Rename Account’.




In the dialog box, name the account ‘App Connect Trial’ (exactly as shown – capitals on the first letter of the words, spaces between the words) and click ‘Rename Account’ as shown below.



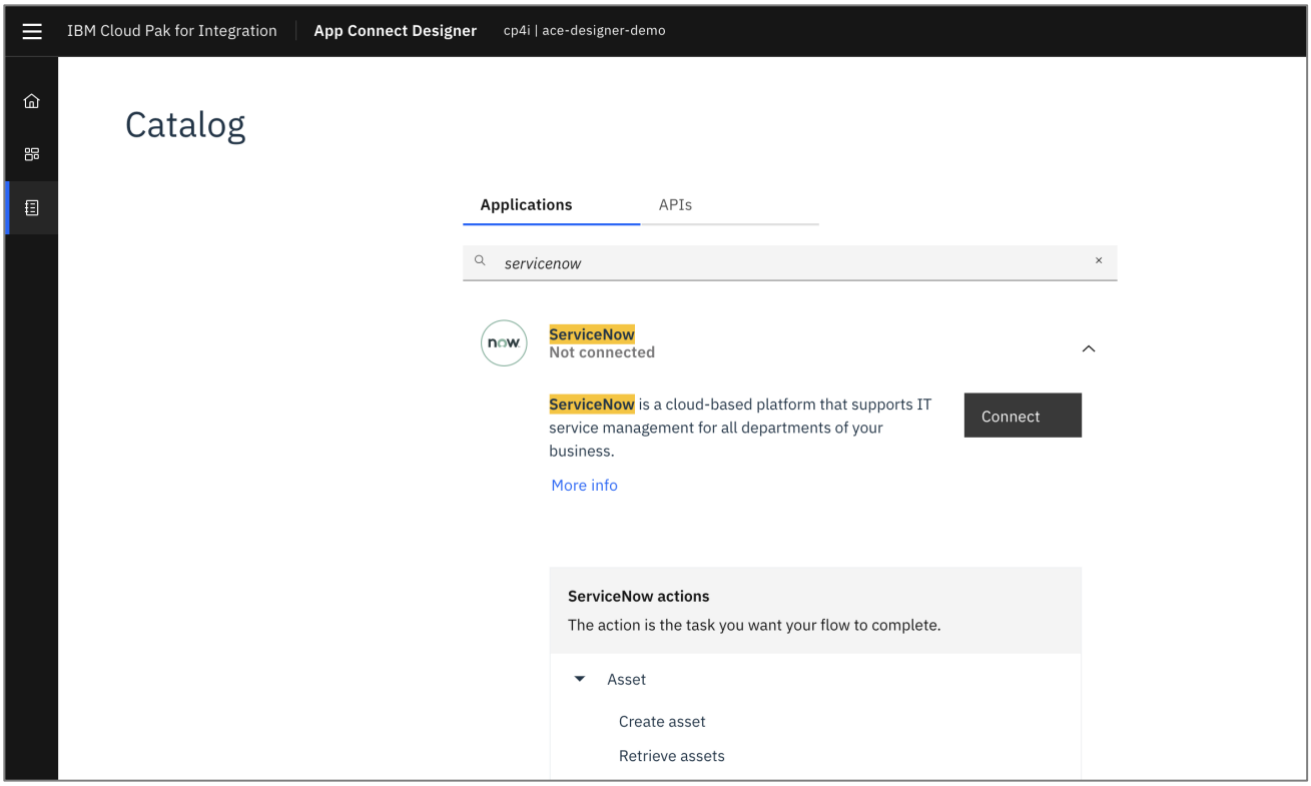

Your connector should now look like this.



### 5.2.2 Connect to ServiceNow

Let’s set up our ServiceNow endpoint – scroll down until you see the ServiceNow connector or you can locate it by filtering via the search box.

Click on ‘Connect’.

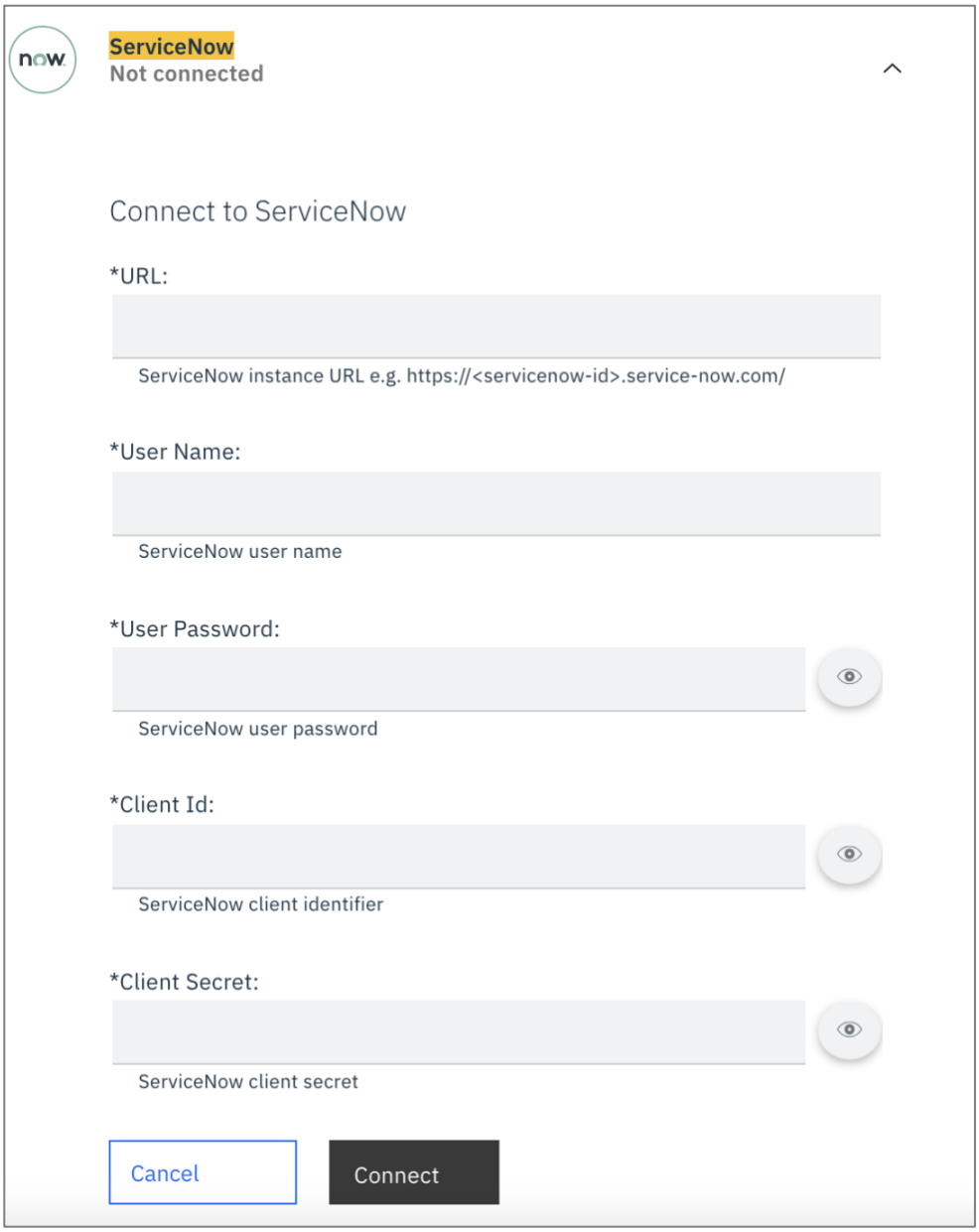



You’ll now be asked for the following details that you kept safe from before: Enter it here and click ‘Connect’.

The **URL of your ServiceNow instance**, in the following form: `https://<servicenow-id>.service-now.com/`

The **username and password** that you use to log in to the instance.

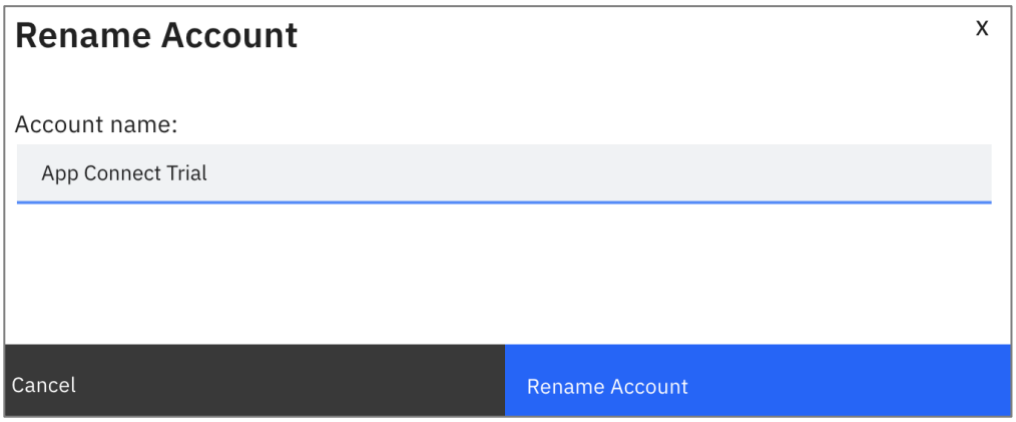
**ClientID** and **Secret** of your ServiceNow application (Oauth endpoint).



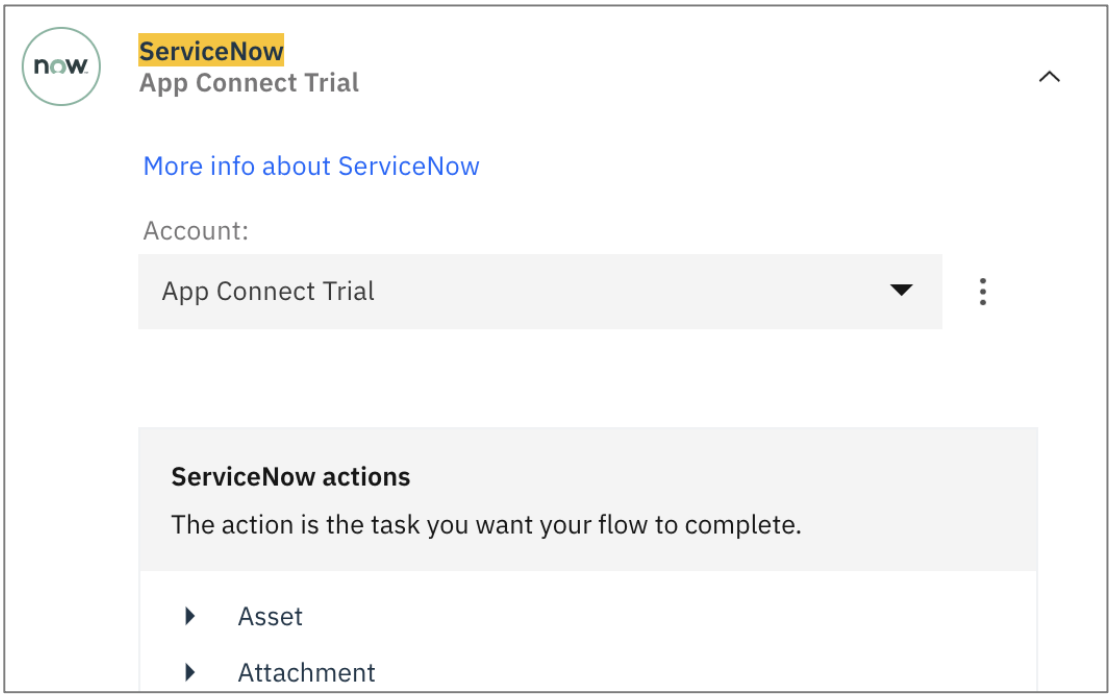

Similarly, we want to rename the account for the Lab to work seamlessly.

To rename your account, Click the three dots menu and click ‘Rename Account’.

In the dialog box, name the account ‘App Connect Trial’ (exactly as shown –capitals on the first letter of the words, spaces between the words) and click ‘Rename Account’ as shown below.



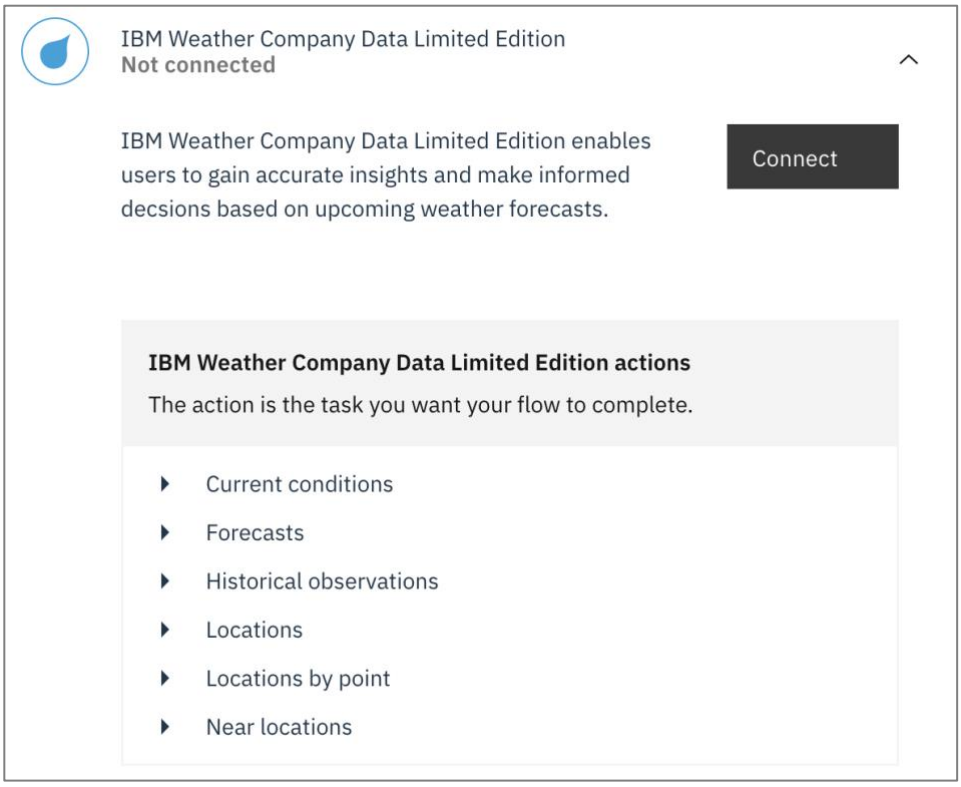

Your connector should now look like this.



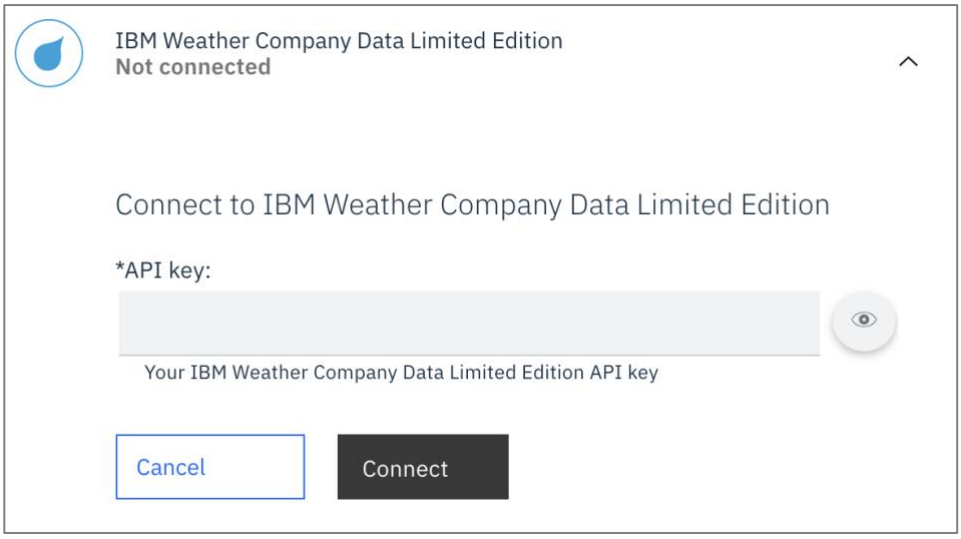

### 5.2.3 Connect to IBM Weather

Let’s set up our IBM Weather endpoint – scroll down until you see the IBM Weather connector or filter via the search box.

Click on ‘Connect’.



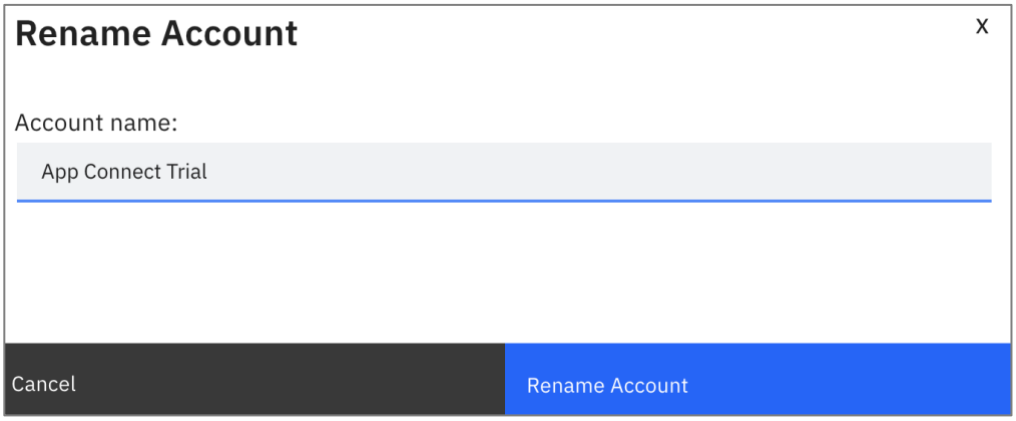

The service is protected by an API key. You'll now be asked for the API key that you kept safe from before: Enter it here and click 'Connect'.



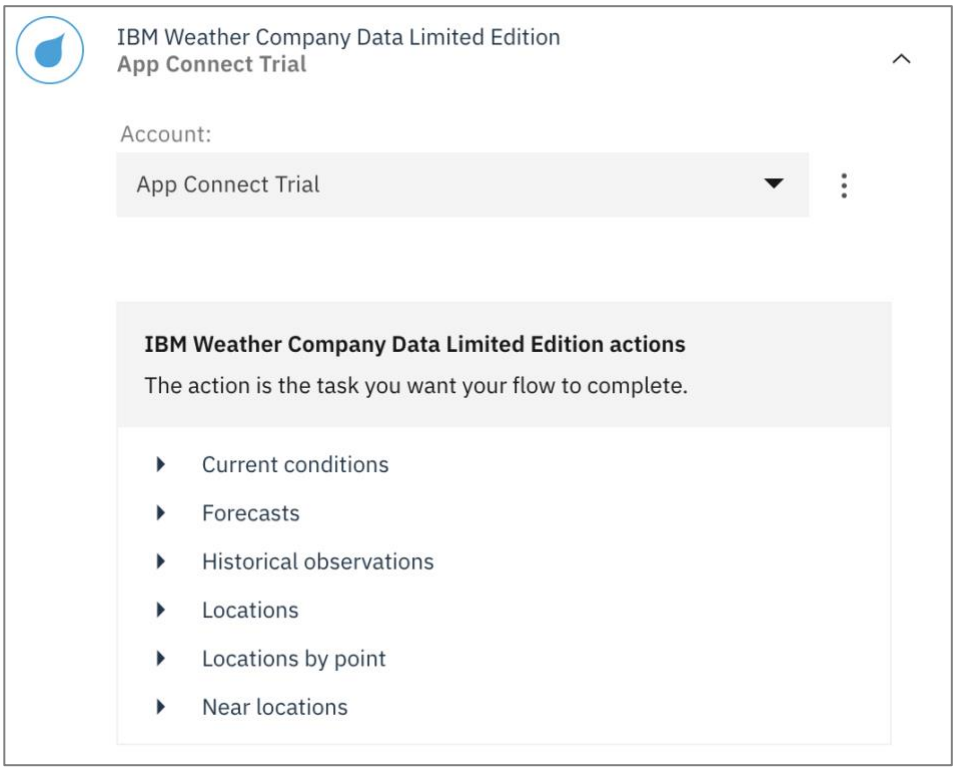

Similarly, we want to rename the account for the Lab to work seamlessly.

To rename your account, Click the three dots menu and click 'Rename Account'.


In the dialog box, name the account ‘App Connect Trial’ (exactly as shown – capitals on the first letter of the words, spaces between the words) and click ‘Rename Account’ as shown below.



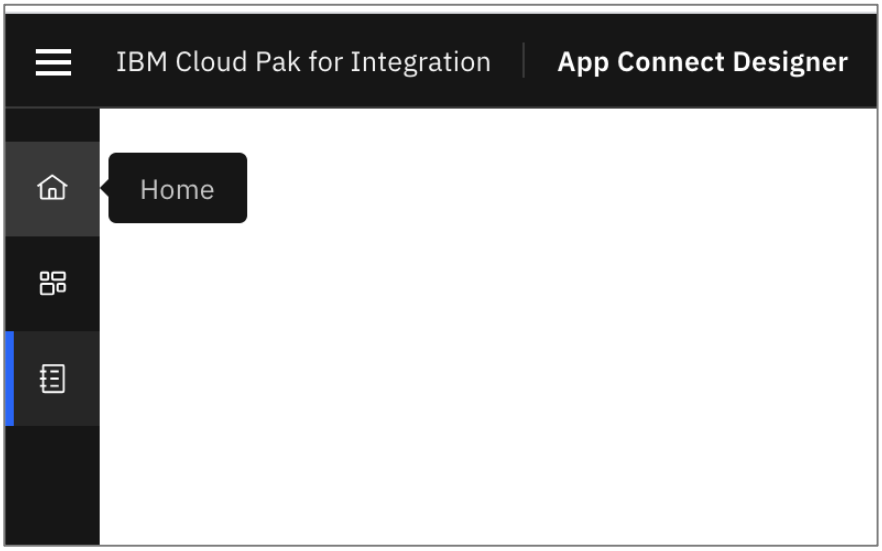

Your connector should now look like this.



Great! We're now all connected up! Let's go and see our flow!

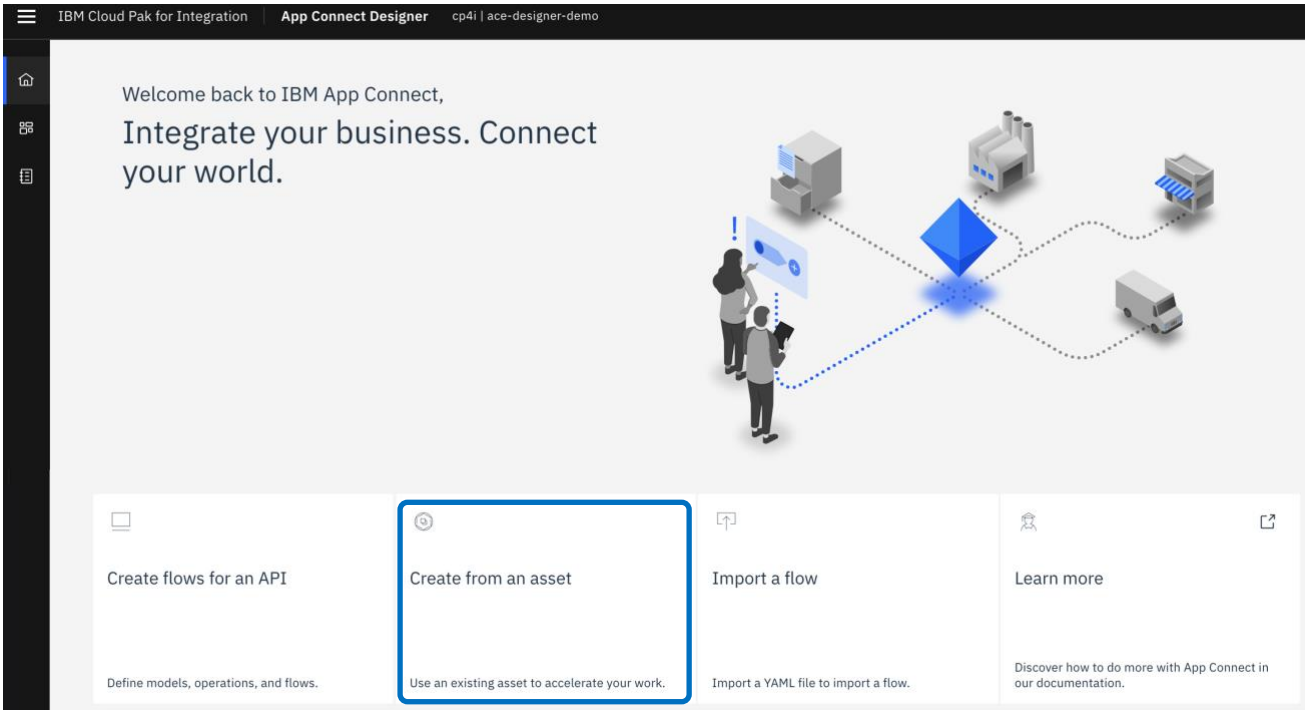
## 5.3 Importing the Integration flow into designer

Go back to the 'Home' page in Designer by clicking the 'home' icon.



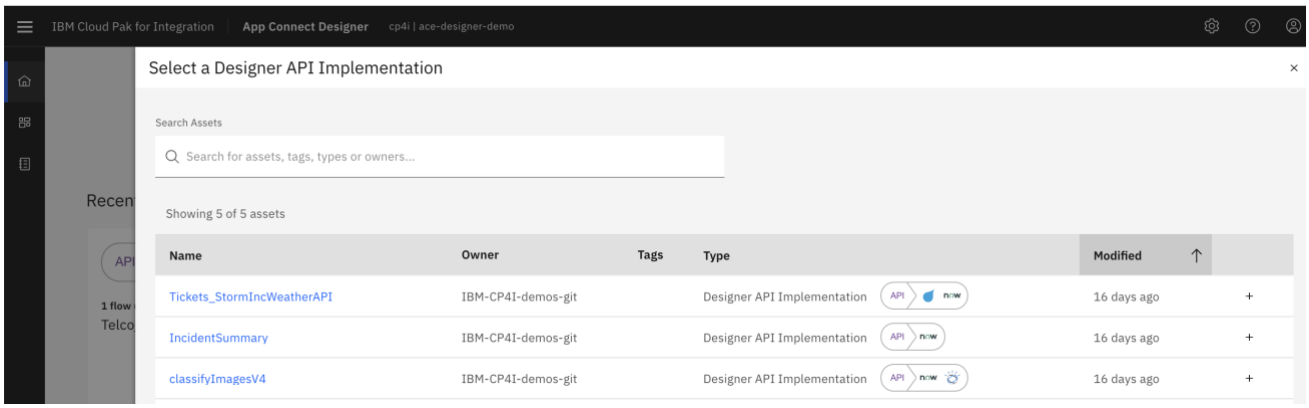

We're going to import our flow from the Asset Repository: The 1-click install has put it there for you...

Click on 'Create from an asset'.



We have a flow to use already stored in the Asset Repository: We're going to import it to save you typing and clicking!

There is a lot of detailed designer flow documentation for when you want to delve deeper – a good place to start is [https://ibm.biz/learnappconnect](https://ibm.biz/learnappconnect)



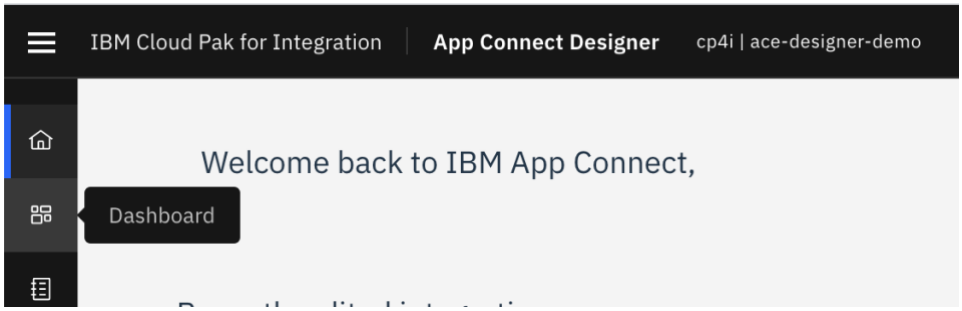

Click on the '+' sign to the right on the 'Tickets_StormIncWeatherAPI' asset and create from the asset.

Repeat the same for 'IncidentSummary' and 'classifyImagesV4' assets.

## 5.4 Reviewing our API Integration Flows:

This section is for you understand how the flow is built, as the flows are pre-built and we won't change it in the lab, you can skip straight to 'Starting the flow' section and come back here later.

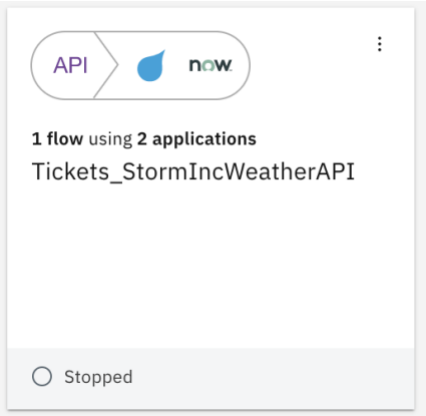
Go back to the 'Dashboard' page in Designer by clicking the 'Dashboard' icon where you can access all your flows.




### 5.4.1 The 'Tickets_StormIncWeatherAPI' API flow

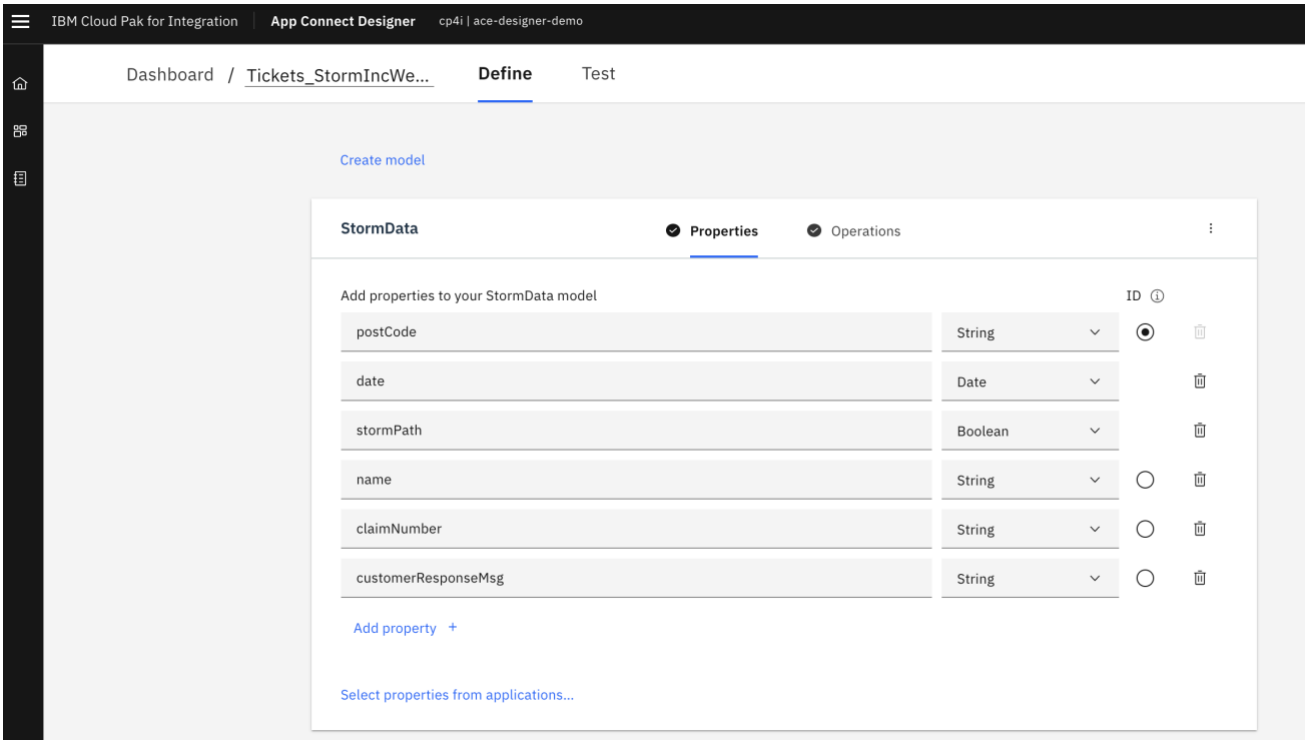
This API checks the weather on a date and location and opens a ServiceNow ticket if it determines that a storm has happened.

Click on the ‘Tickets_StormIncWeatherAPI’ API flow tile.

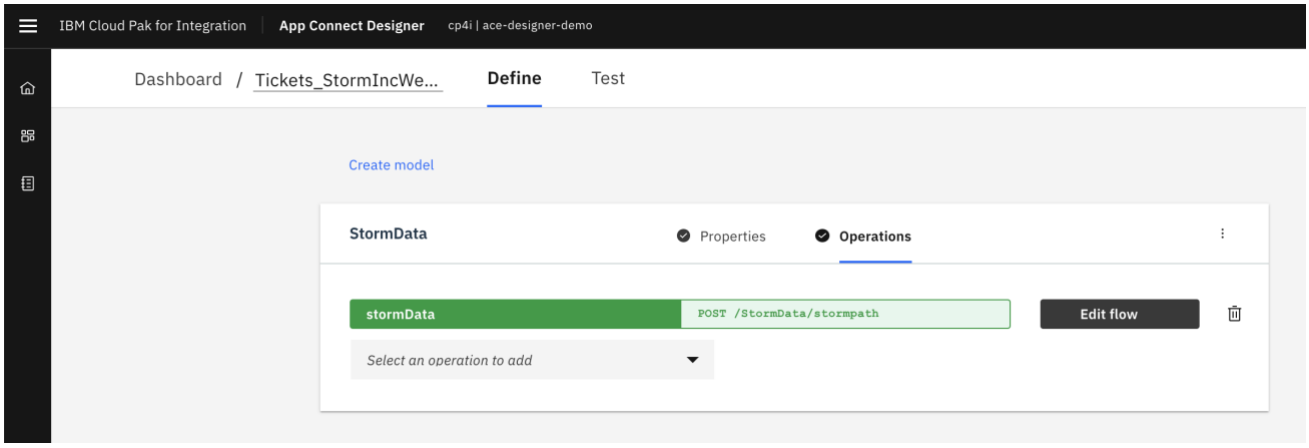



What you can see first is our API model.

App Connect Designer builds your API for you – you don’t need to worry about OpenAPI specs or Swagger editors – it’s all built in. To create your API, you just type in the names of the fields you want to use in plain English.




Now that we’ve told the API what data to use, we need to define what actions to perform on that data. Click ‘Operations’.

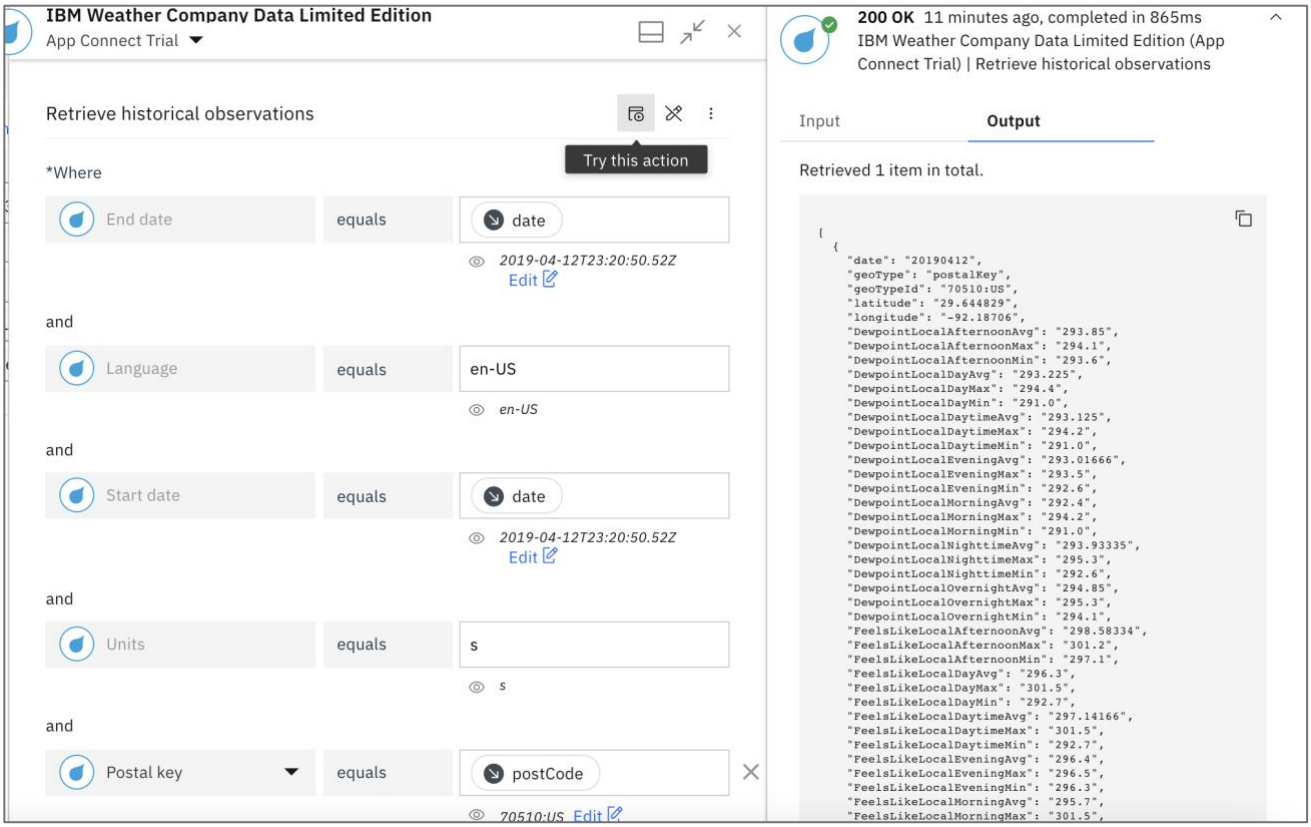



Click ‘Edit flow’ to see the details of the flow.


The first step in the flow uses date and location in the request input and retrieves the weather observations on that specified date and location.

You can also test this step independently by defining sample data and clicking ‘Try this action’.

You will be amazed by how much data you can retrieve related to Weather observation. For the purpose of this demo, we will use wind and precipitation to evaluate the storm condition.




The ‘If” construct checks if the ‘wind speed is greater than or equal to 20’ and ‘precipitation is greater than or equal to 10’.

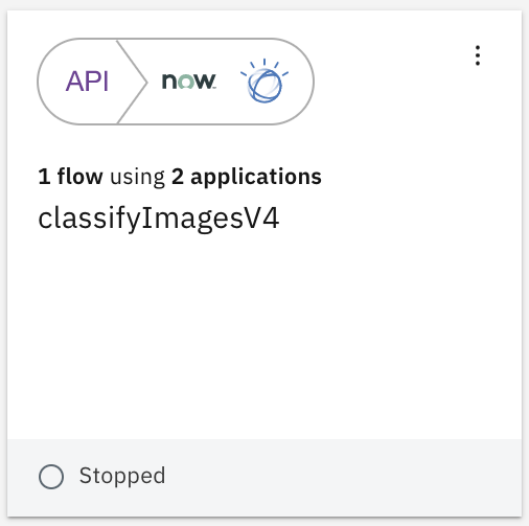

If the above condition evaluates to true, we assume there is a storm occurrence and create a provisional storm claim in ServiceNow and return the claim id and the evaluation result.

If the above condition evaluates to false, we directly return the evaluation result.

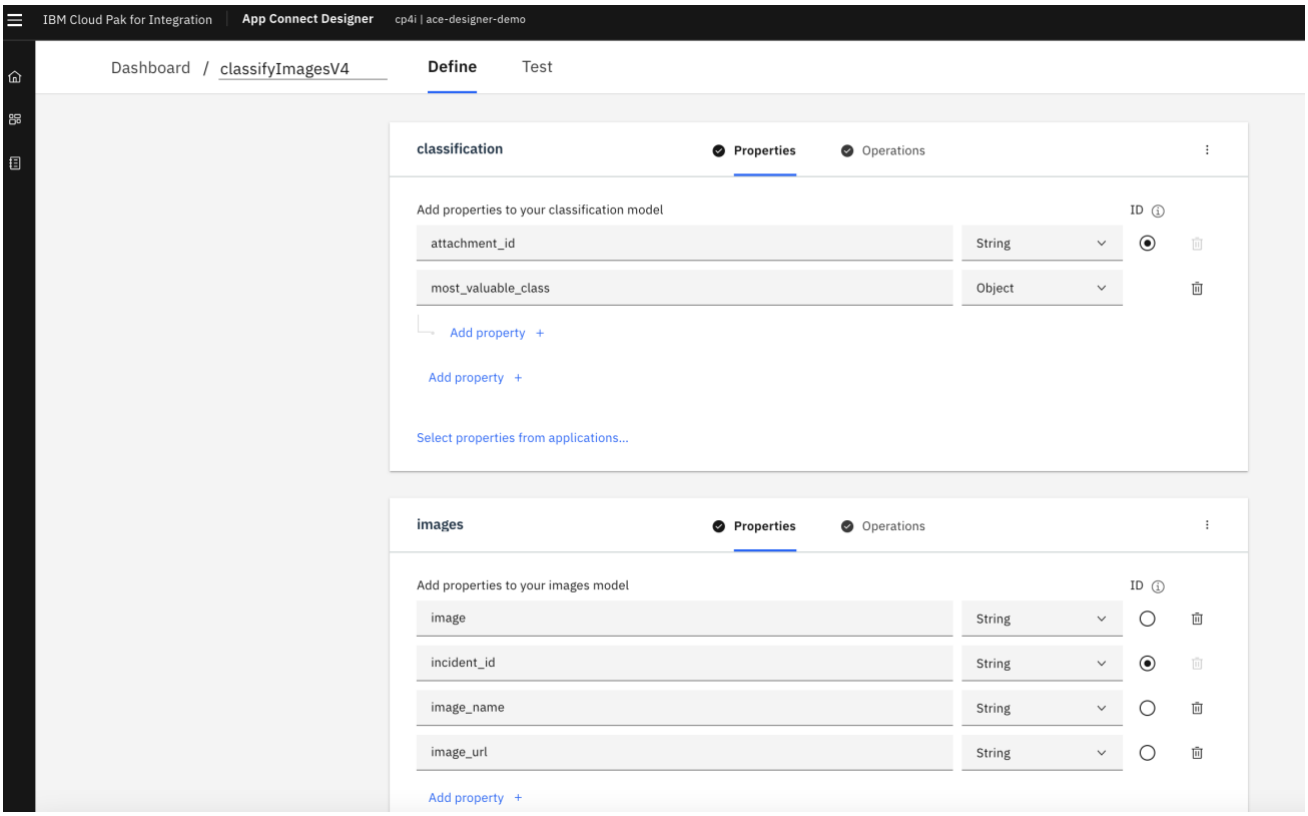
### 5.4.2 The ‘classifyImages’ API flow

This API takes an image and a ServiceNow ticket number as input, analyses the image with Watson Image Recognition and finally updates the ticket with analysis results.

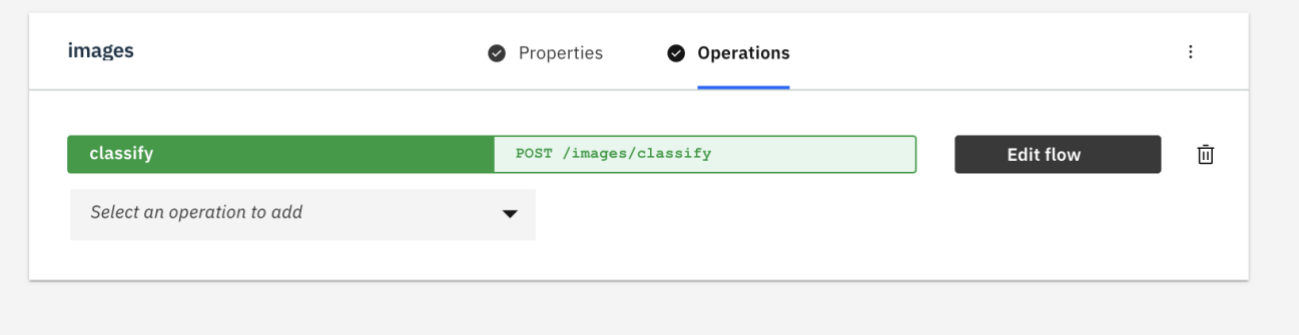
Click on the ‘classifyImagesV4’ API flow tile.



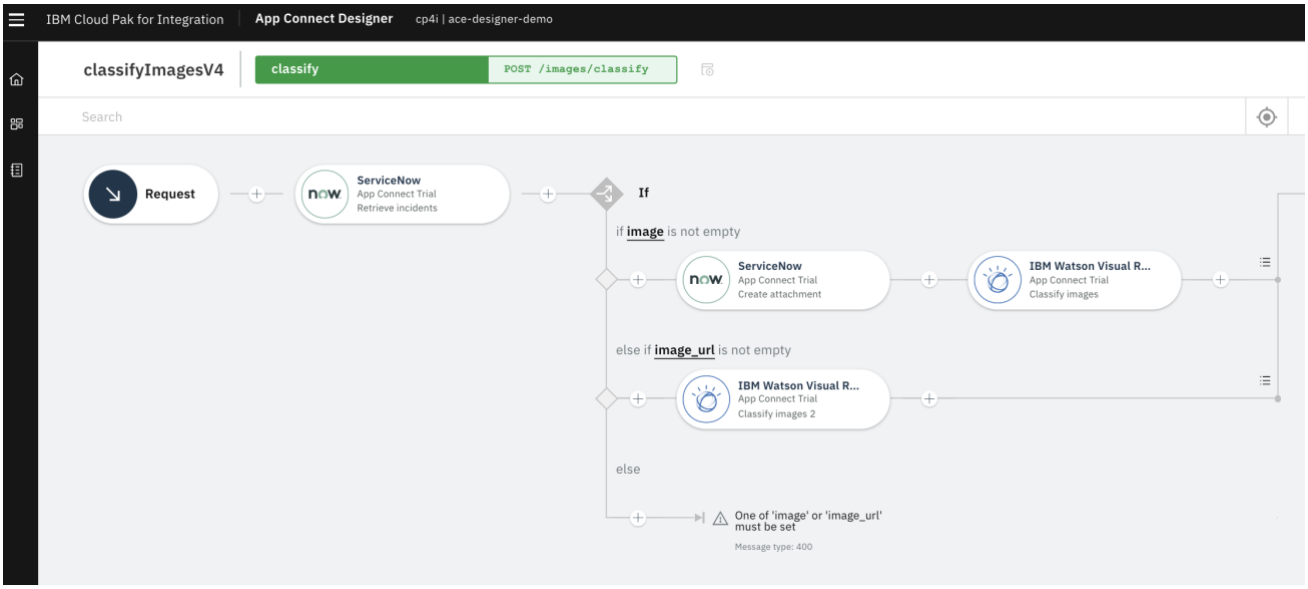

What you can see first is our API model.



Now that we've told the API what data to use, we need to define what actions to perform on that data. Click 'Operations' of 'images' data model.




Click 'Edit flow' to see the details of the flow.




Retrieves the ServiceNow ticket.

Checks that either an image (file base64 encoded) or a URL to an image has been supplied.

- If an image has been supplied, it adds it as an attachment to ServiceNow
- If a URL has been supplied, it does not add an attachment and adds the attachment ID as the IF output context
- If neither supplied, exits the flow with error

Invokes Watson Visual Recognition to get the recognised 'classes' for the image from the 'default classifier'

Uses the JSON parse node to set an arbitrary set of 'claimable values' for a few classes

The set variable node augments the discovered classes with their claimable value.

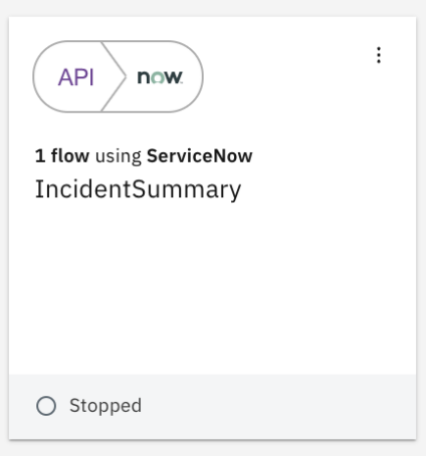
It responds with what the image was recognized as, and its maximum claimable value. It also augments the ticket.

- Adds the image as an attachment to the ticket
- Augments the description field with a JSON structure that holds the information discovered (for later retrieval)

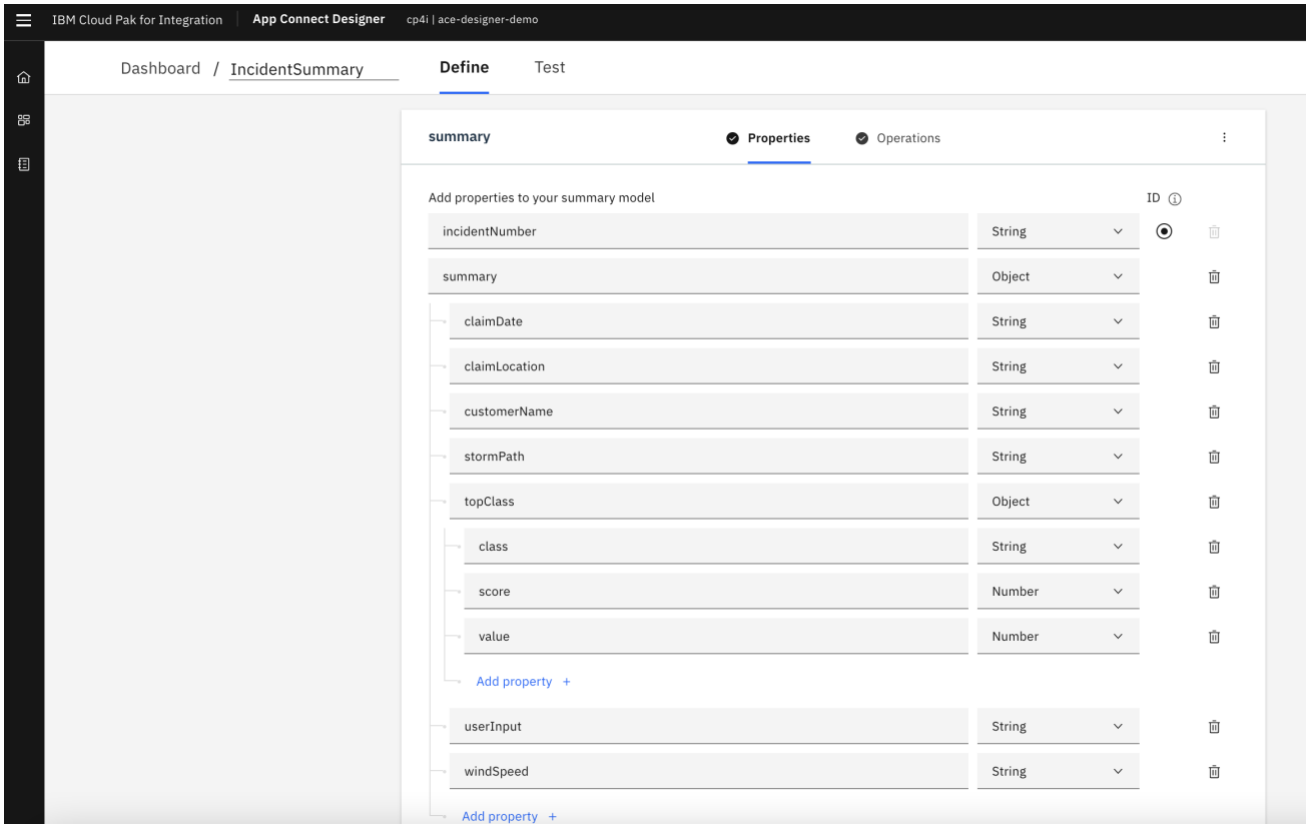
### 5.4.3 The ‘IncidentSummary’ API flow

This API simply takes a ServiceNow ticket number as input, returns the details.

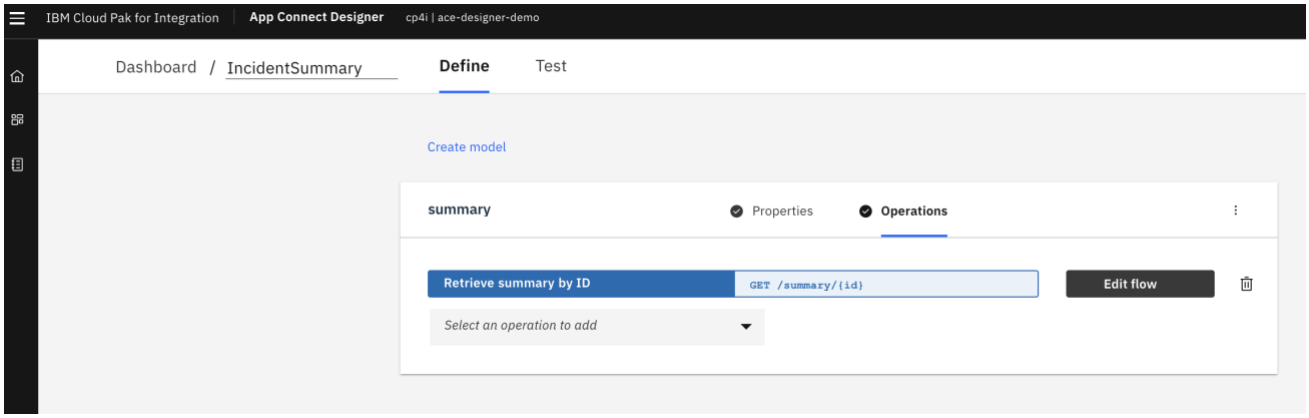
Click on the ‘IncidentSummary’ API flow tile.



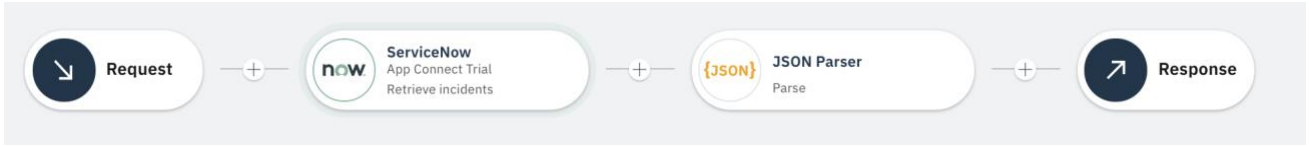

What you can see first is our API model.




Now that we've told the API what data to use, we need to define what actions to perform on that data. Click 'Operations'.




Click 'Edit flow' to see the details of the flow.




## 5.5 Testing our API Integration Flows

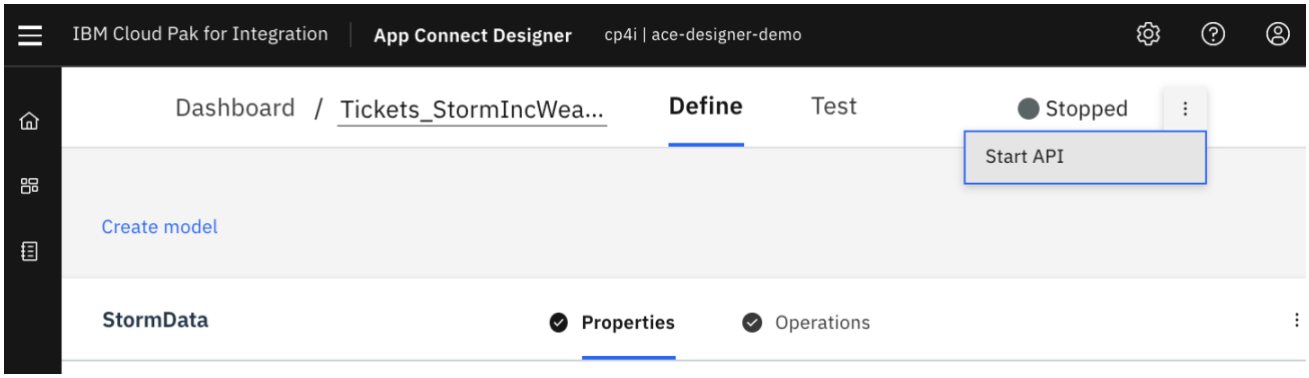
Now we've built our API, we need to test it. When you create an API flow in your App Connect Designer instance, the definition provides an API that you can expose.
After you start the flow, you can verify its behaviour by using the built-in test facility to call the endpoints for each of the implemented API operations.

In the course of this lab, we want to test our APIs using built-in test facilities. This will give the us the assurance to promote our flows from Designer runtime (which is a development environment) to integration runtime.

If the flow is not already open, click 'Dashboard' in the navigation pane and then click the flow tile.

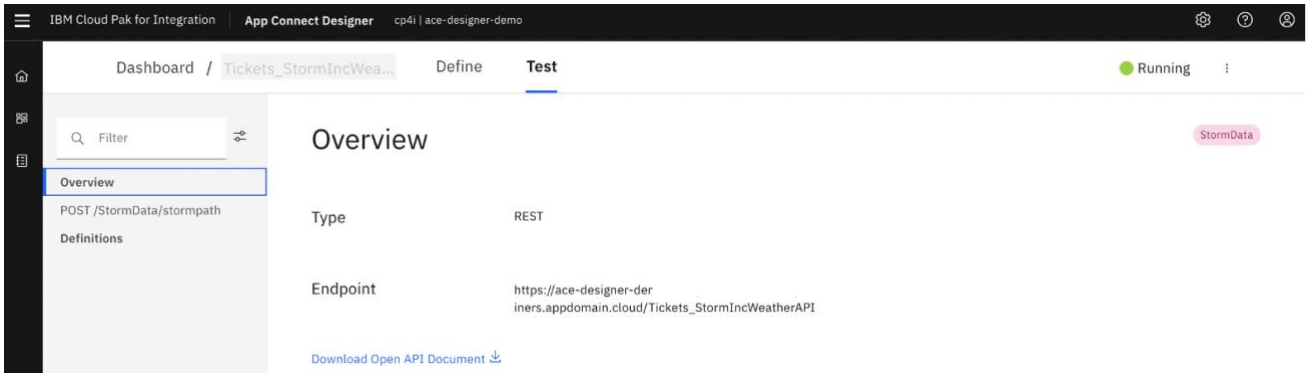
Let's start with 'Tickets_StormIncWeatherAPI' flow.

Click 'Start API'




Click the 'Test' tab.

The Overview page displays the API type and the base URL for the API endpoint. To the right of the "Overview" title, a tag is provided for each model that is defined in the API flow.




A Download Open API Document link is also provided for the OpenAPI document that describes the API.

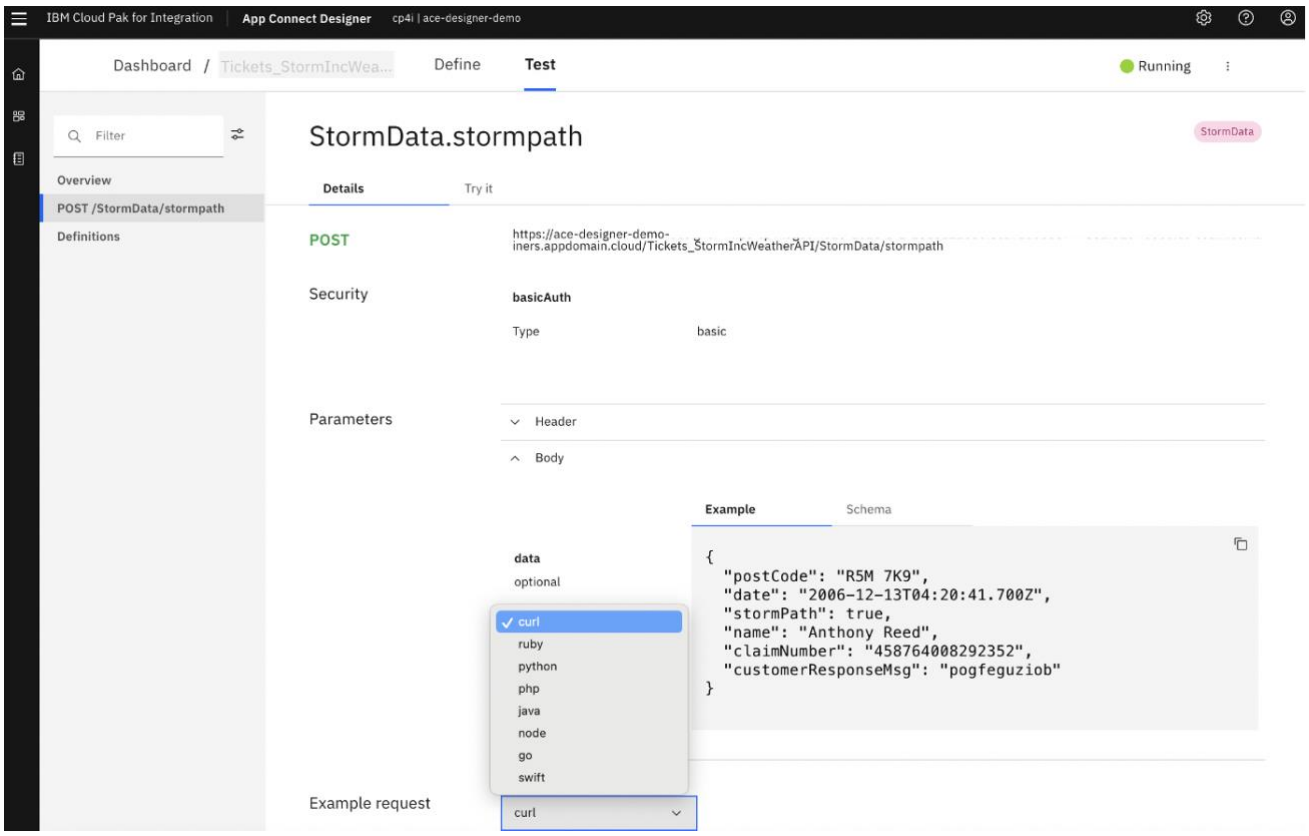
If you download this document, it is saved as a YAML file to the default download location that is configured for your browser. The format of the file name is '*flowName-version.yaml'*; for example, Tickets_StormIncWeatherAPI-1.0.0.yaml. The version number is derived from the version of the API in the OpenAPI document and is always set to 1.0.0.

You can use the OpenAPI document to test your flow in any other test tool or you can proceed with the embedded test option.

From the left pane, click 'POST /StormData/stormpath' operation to view its details and test the behaviour. Notice that the tag shown will reflect the model that an operation belongs to.

The Details tab displays the following information:

- The HTTP method and request for the operation.
- The authentication method (security scheme) that the API uses.
- The header parameters in a collapsible section.
- The body, path, or query parameters with examples, and the schema if relevant, in collapsible sections. The parameters that you see will depend on the operation's settings.
- Tooling languages that can be used when making the request, and a code sample for calling the operation in the selected language.
- Response status codes for the operation, and the response body schema with an example.



Again, you can use the embedded test scripts to test your flow from your preferred tooling language or proceed with the embedded test option.

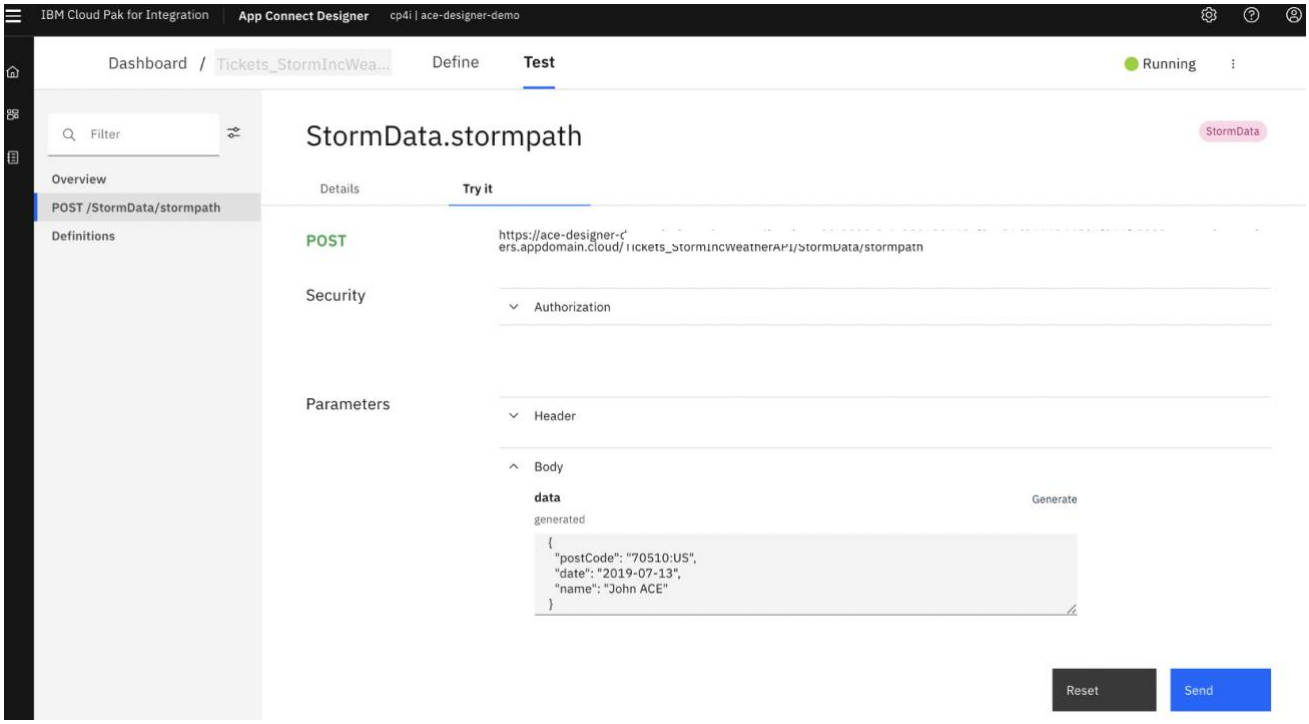
To proceed with the embedded test option, click ‘Try it’.

Use the below test data which passes the storm validation and creates a ServiceNow incident.

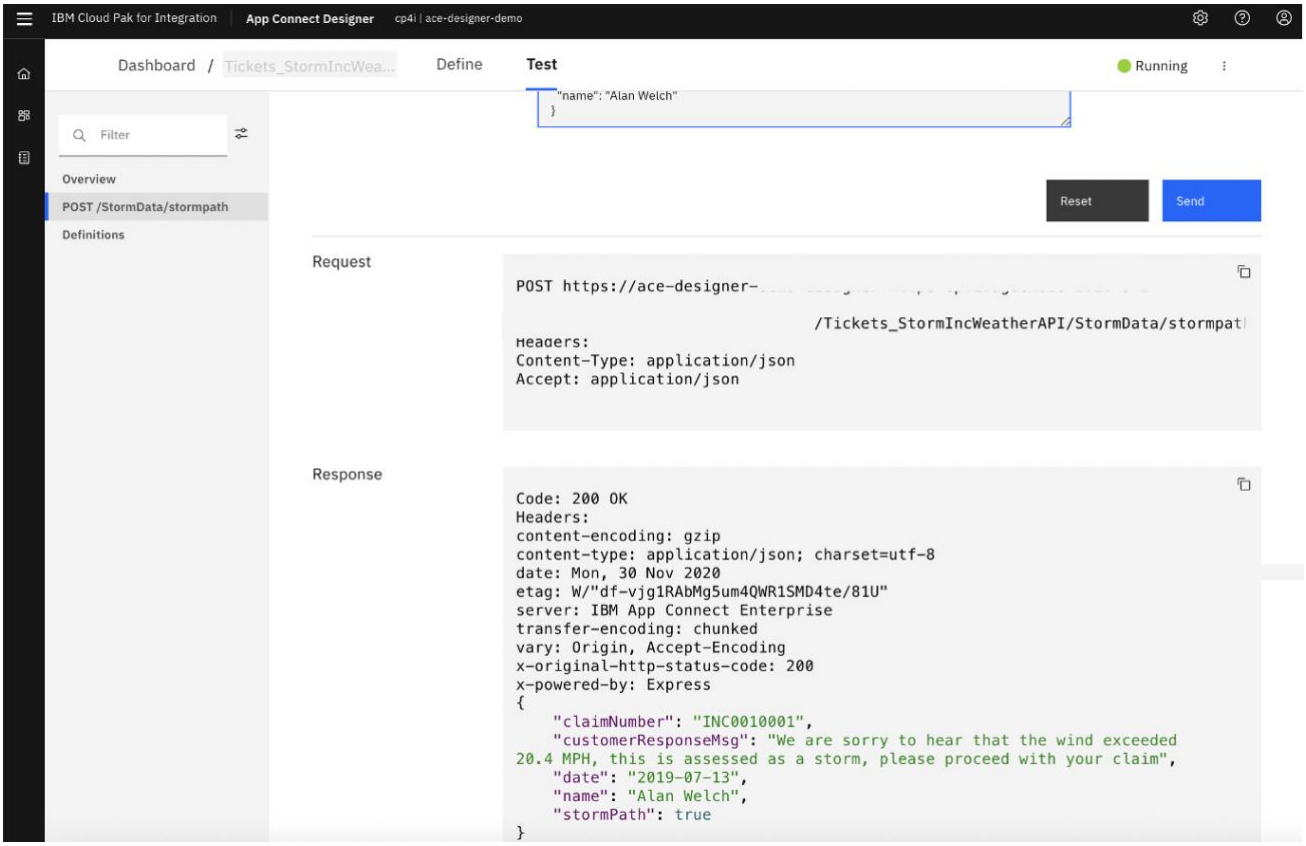
```
{
  "postCode": "70510:US",
  "date": "2019-07-13",
  "name": "John ACE"
}
```

Note that authentication credentials are auto generated and displayed together with the request parameters.

Click ‘Send’.




Then review the request and response that are displayed.



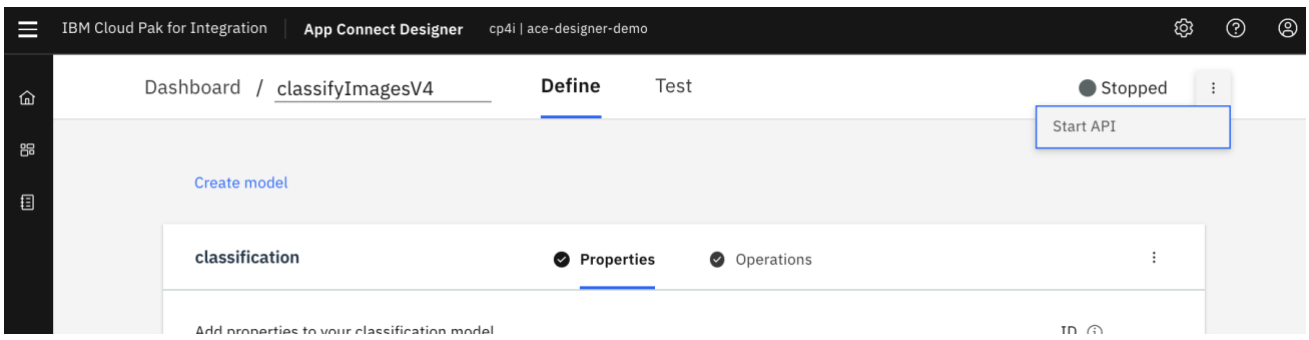

For our POST example, the Response section displays the success code that is returned (200 OK), the headers, and the claimNumber value that represents the ID assigned by ServiceNow.

We will use this claimNumber as an input parameter to test the other APIs.

Next, we will start ‘classifyImagesV4’ API.

If the flow is not already open, click ‘Dashboard’ in the navigation pane and then click the ‘classifyimagesV4’ flow tile.

Click ‘Start API’.

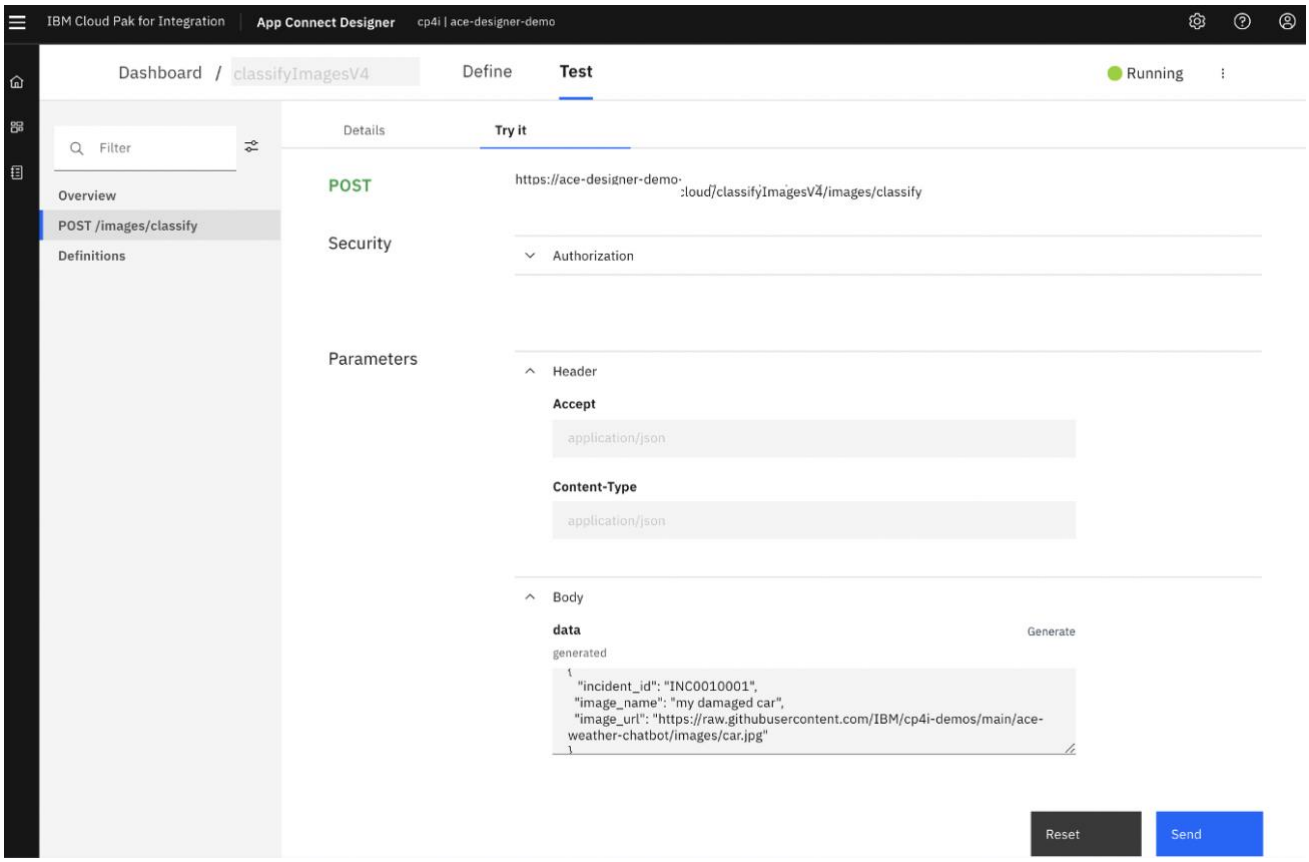


Click ‘POST /images/classify’ and ‘Try it’.

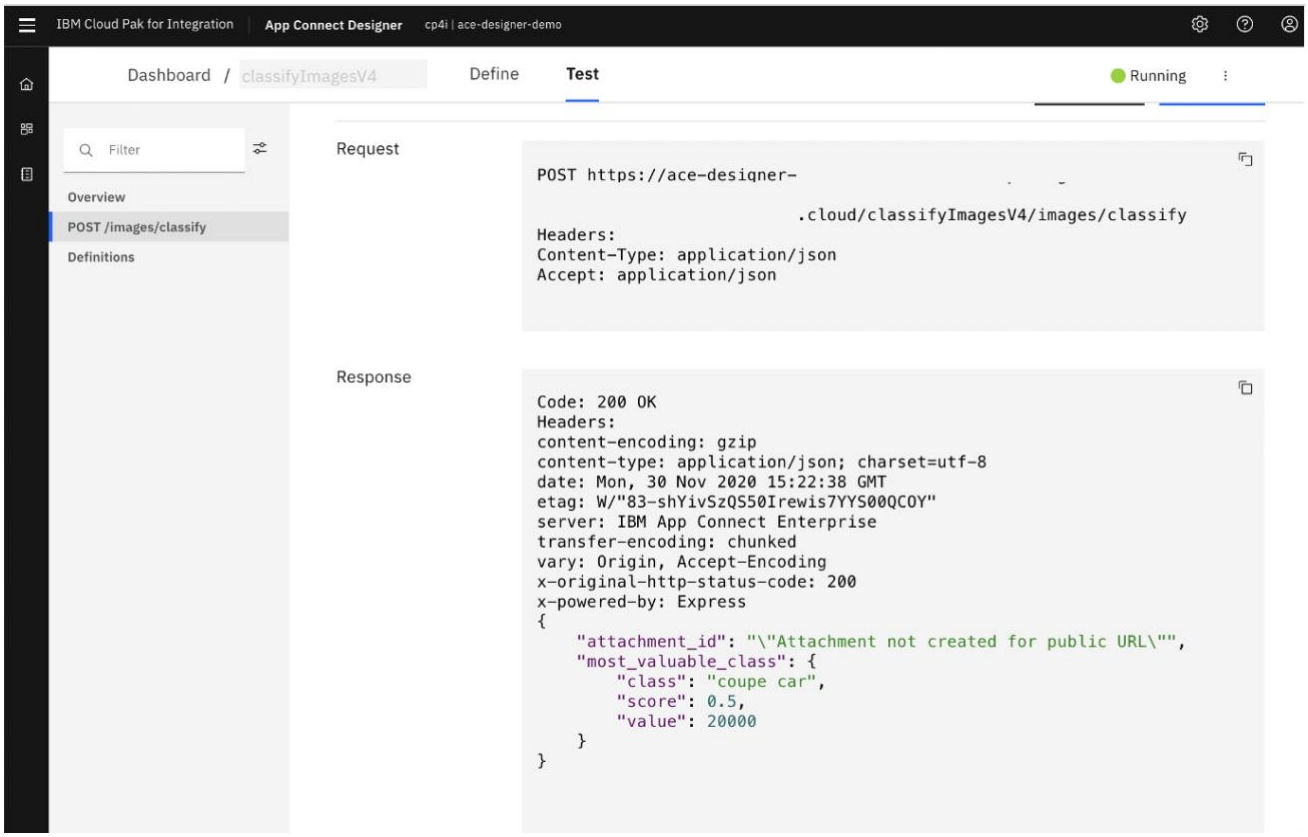
Use the below test data which passes the ServiceNow incident id created in the previous step along with the image url.

```
{
  "incident_id": "INC0010001",
  "image_name": "my damaged car",
  "image_url": "https://raw.githubusercontent.com/IBM/cp4i-demos/main/ace-weather-chatbot/images/car.jpg"
}
```

Click ‘Send’.




Then review the request and response that are displayed.

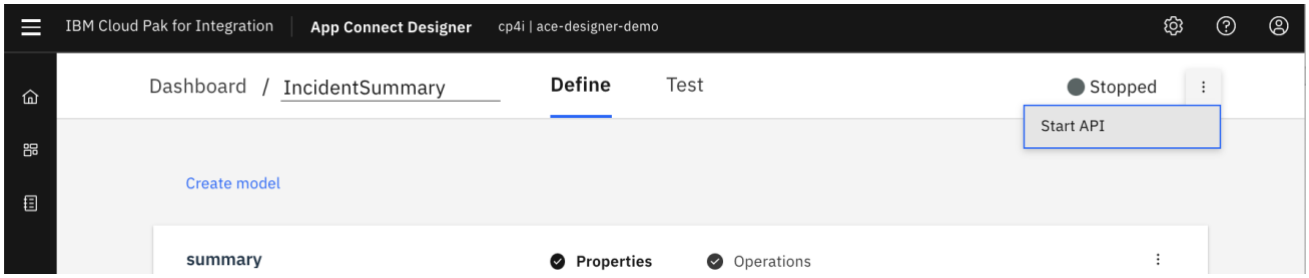



For our POST example, the Response section displays the success code that is returned (200 OK), the headers, and the classification values provided by Watson Image Recognition for the input image.

Lastly, we will start ‘incidentSummary’ API.

If the flow is not already open, click ‘Dashboard’ in the navigation pane and then click the ‘incidentSummary’ flow tile.

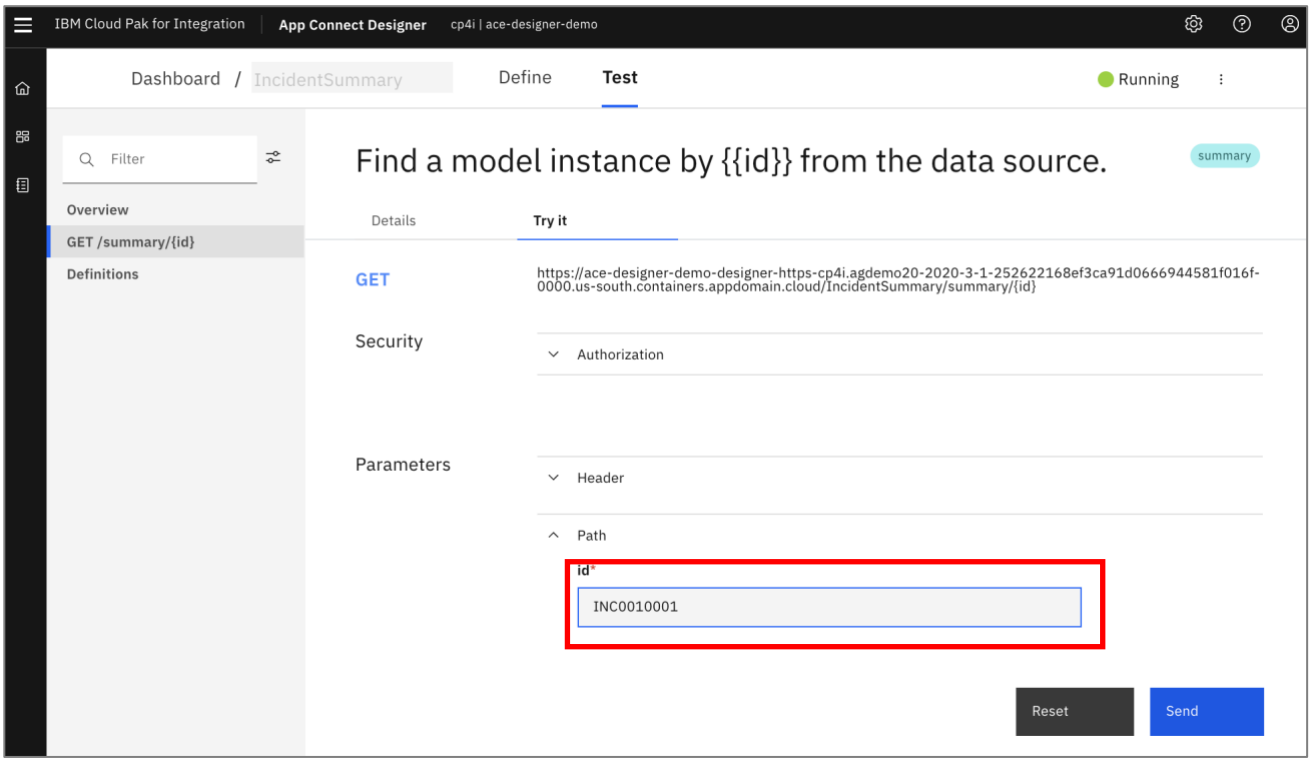
Click ‘Start API’.



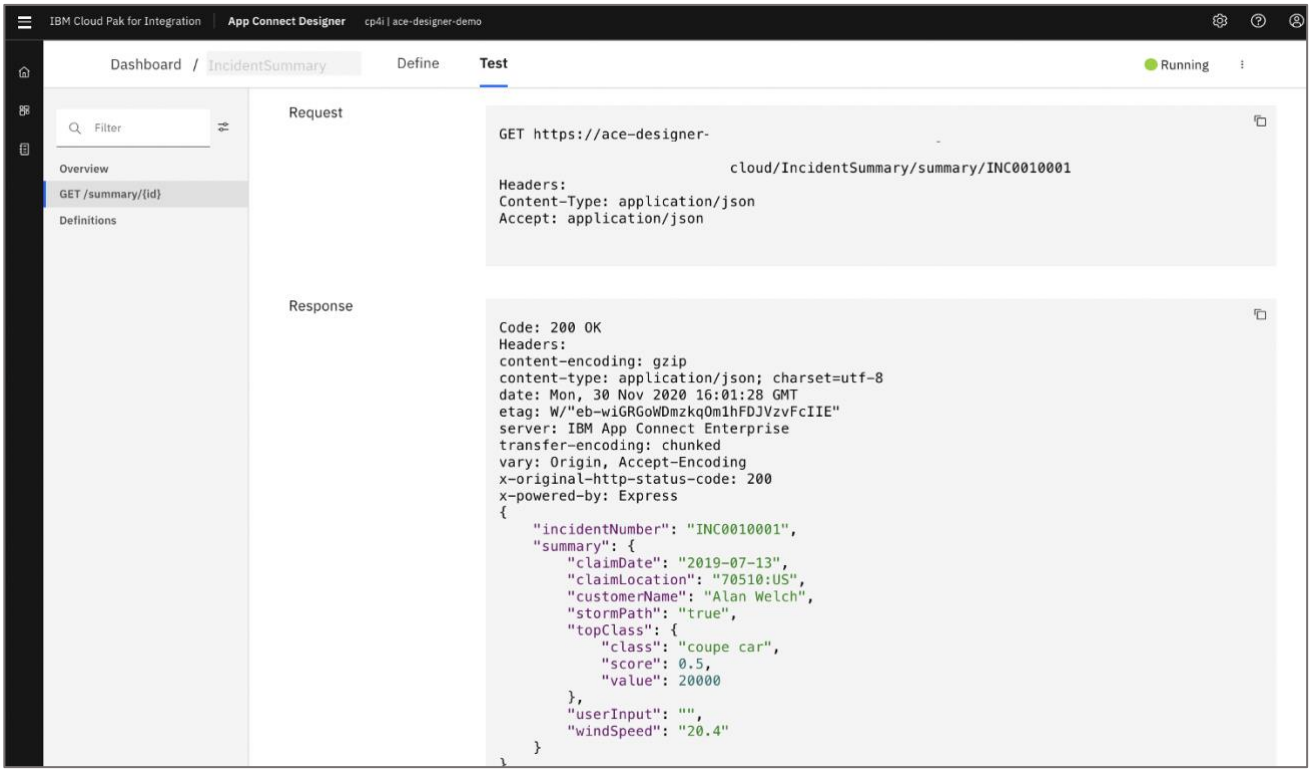

Click ‘GET /summary/{id}’ and ‘Try it’.

Use the ServiceNow incident id created in the previous steps.

Click ‘Send’.




Then review the request and response that are displayed.



For our GET example, the Response section displays the success code that is returned (200 OK), the headers, and the updated claim details.

We've now got our flow running in the designer and we've tested it – now we are ready deploy it 'for real' on the cloud pak runtime.

# 6 Deploying the Integration flow to CP4I Runtime via the App Connect Dashboard

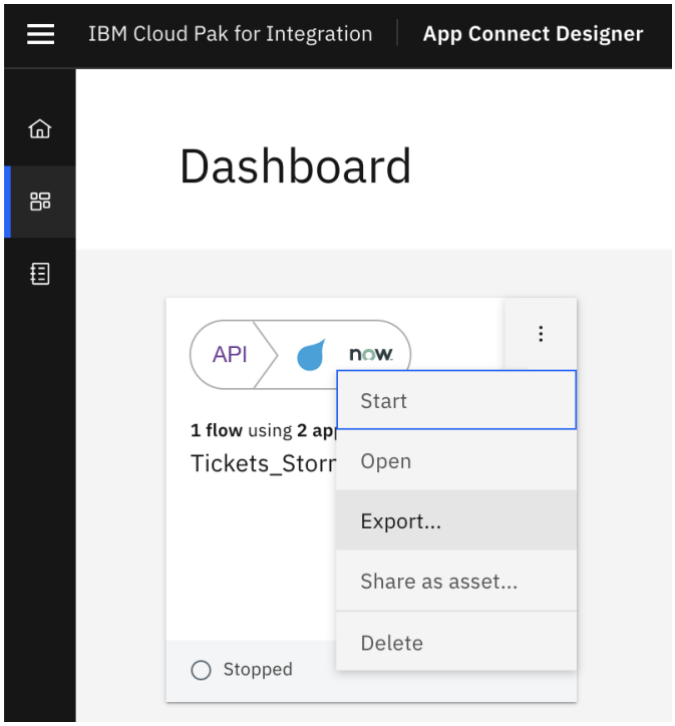
We've now got our flow running in the designer and we've tested it – now we need to deploy it 'for real' on the cloud pak runtime. To do this, we'll export a .bar file of our flow from the designer.

This .bar file contains everything in our flow –with the exception of the connector credentials, which we'll configure later in a Kubernetes secret.

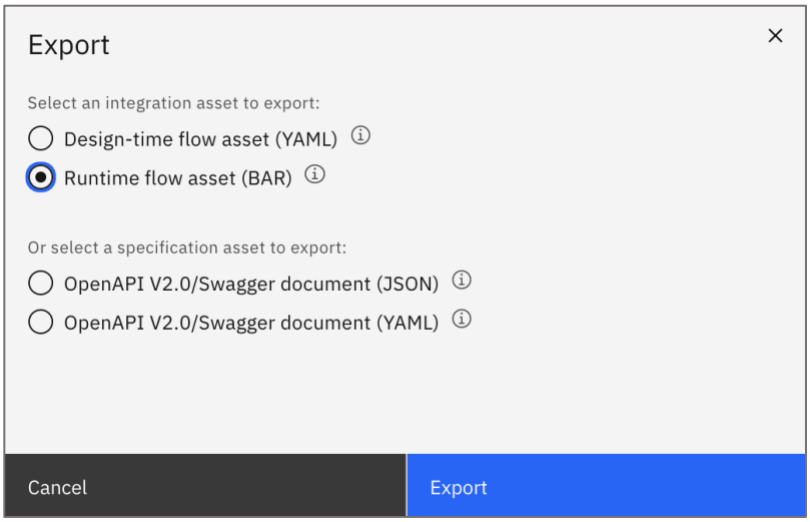
When we deploy, it will create a 3 HA replica container pods running on OpenShift – automatically.

## 6.1 Exporting the executable bar file:

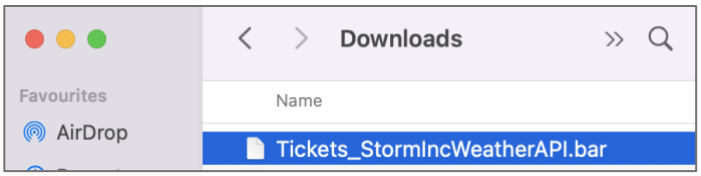
To export the .bar file, go into the designer dashboard and click the '...' menu on the integration tile and click 'Export...'




You'll get a dialog box. Select 'Runtime flow asset (BAR)' and click 'Export'.



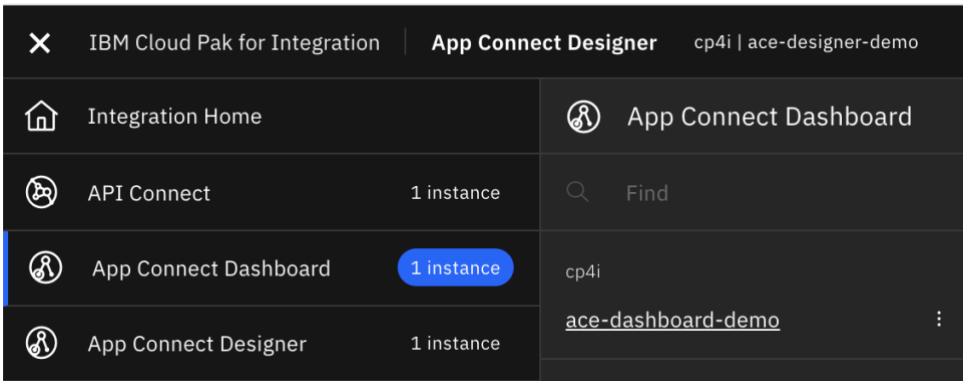

The browser may prompt you for a download location – otherwise it will place the 'Tickets_StormIncWeatherAPI.bar' file in the Downloads directory.



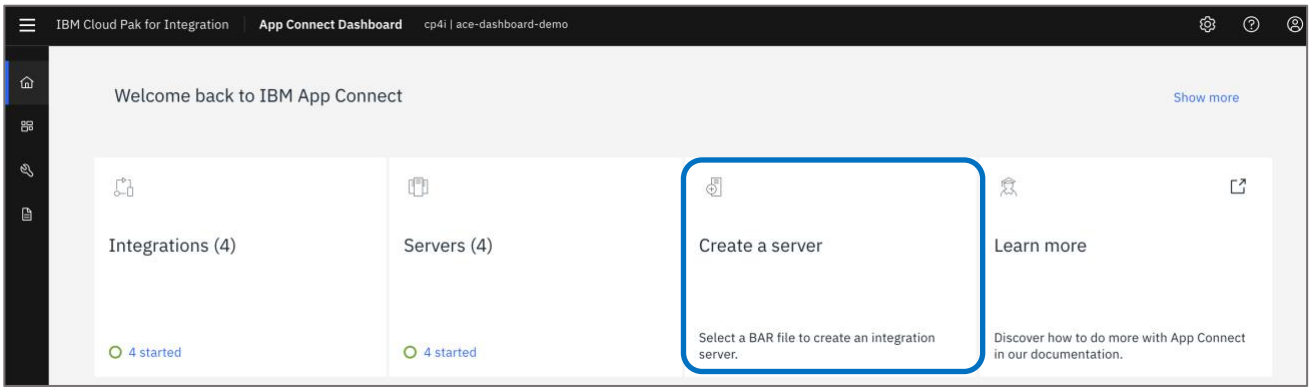

That's it – we now have our executable flow – let's see what we need to do to deploy it.

## 6.2 Navigating to the App Connect dashboard and importing the .bar file

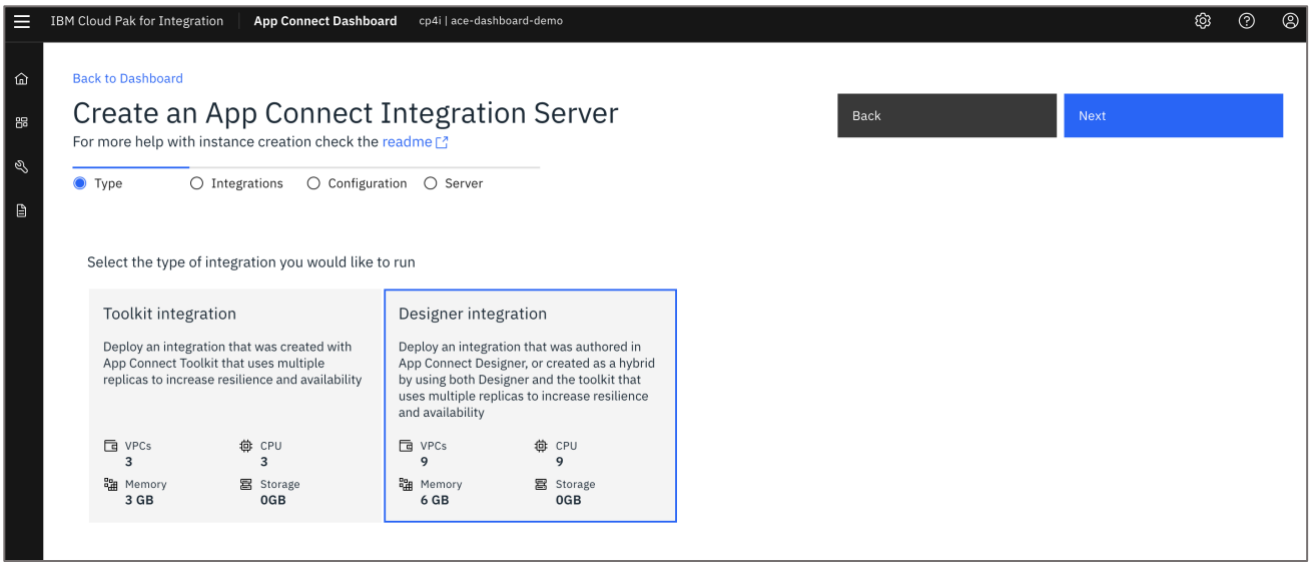
From the menu, click 'App Connect' and then click 'ace-dashboard-demo': This is the runtime, we need not the tooling.



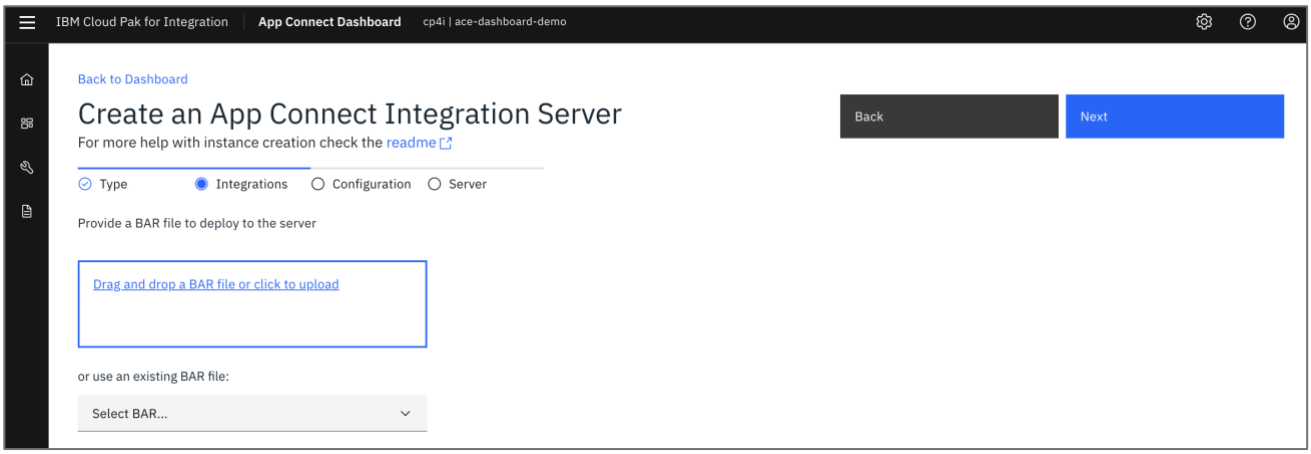

You'll then be taken to the App Connect Dashboard.
Click 'Create A Server'




Now we need to select the kind of tooling we used to build the integration. We used App Connect Designer, so click that and click ‘Next’.




You’re now prompted to upload the .bar file you exported before. In the dialog box, click ‘Drag and drop a BAR file or click to upload’.

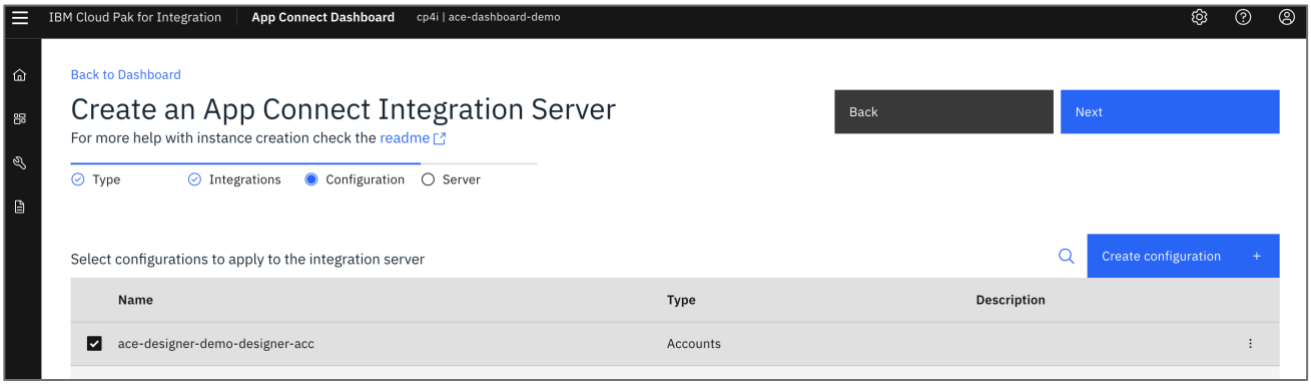



Browse to the location of the ‘Tickets_StormIncWeatherAPI.bar’ file that you exported from designer and select it with ‘Open’ and then click ‘Next’.

In the next step, we need to choose configurations for the connector account credentials as we want to use locally deployed connectors.  For the purpose of this demo, we will use ‘ace-designer-demo-designer-acc’ accounts configuration which holds all the connector credentials we configured previously in App Connect Designer.

Select that as below and click ‘Next’.

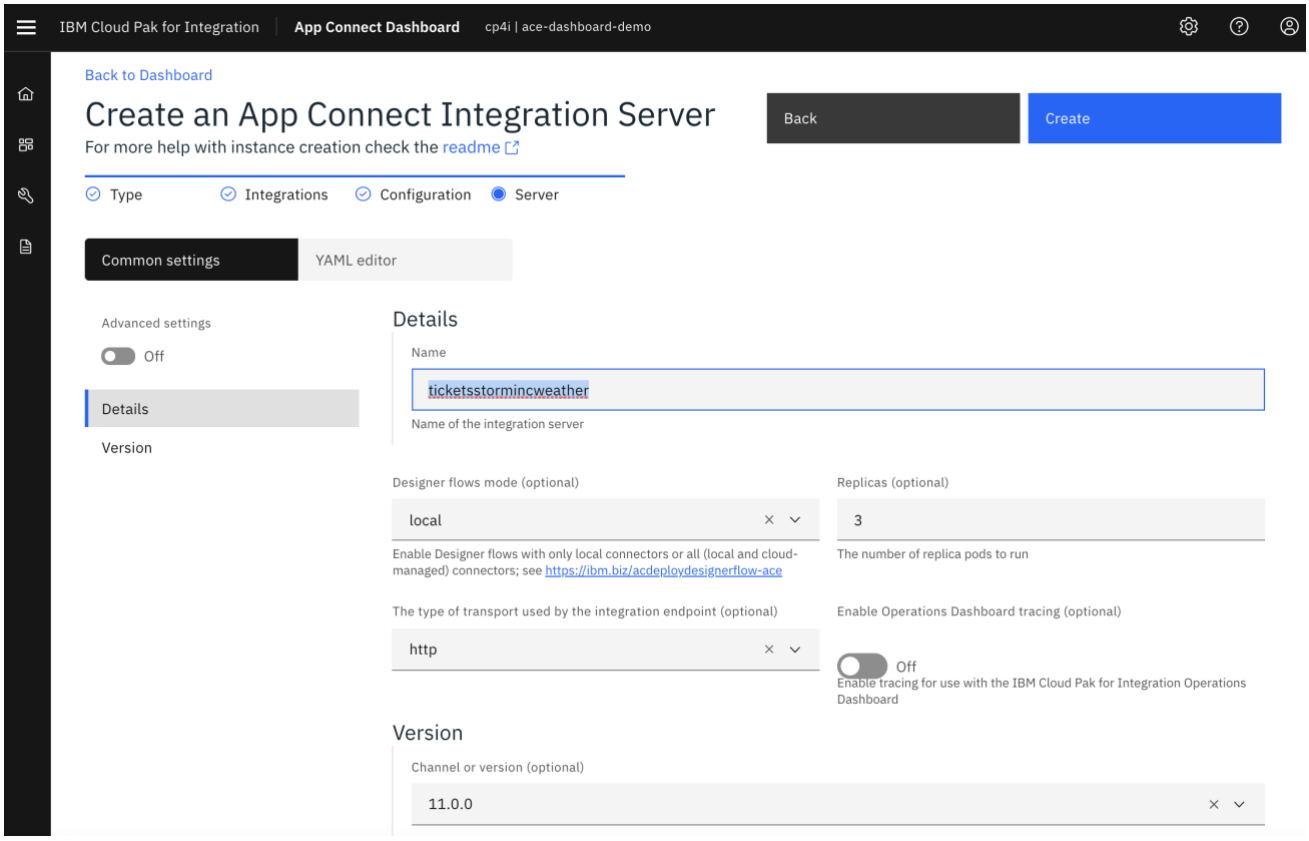
(note: For the purpose of this demo, you don’t need to click ‘create configuration’ here unless you want to use different credentials for your connector accounts.)




You’re now on the last screen!

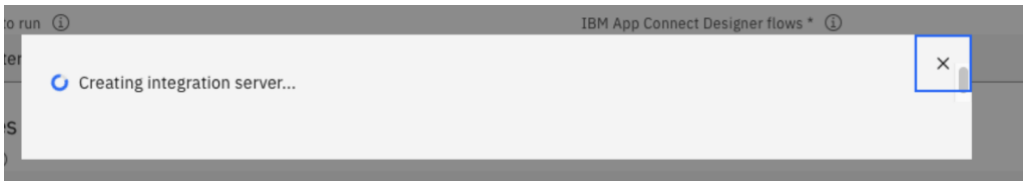
Enter a name for our integration server – note that permitted characters are lowercase alphanumeric and "-” and must start and end with lowercase alphanumeric characters.

In these flows we will use local connectors only, so we select ‘local’ for ‘Designer flows mode’.  This option will extend the functionality of each pod by deploying sidecar containers, which are needed to run APIs that are authored in App Connect Designer, and local connectors.




Now click ‘Create’

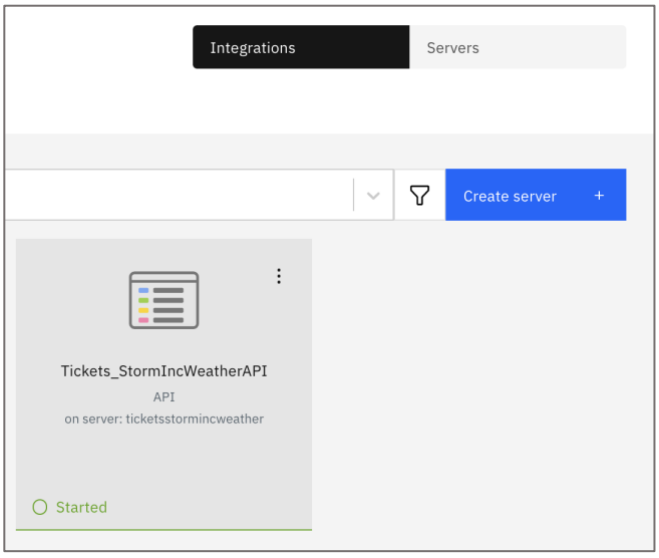
You’ll see:

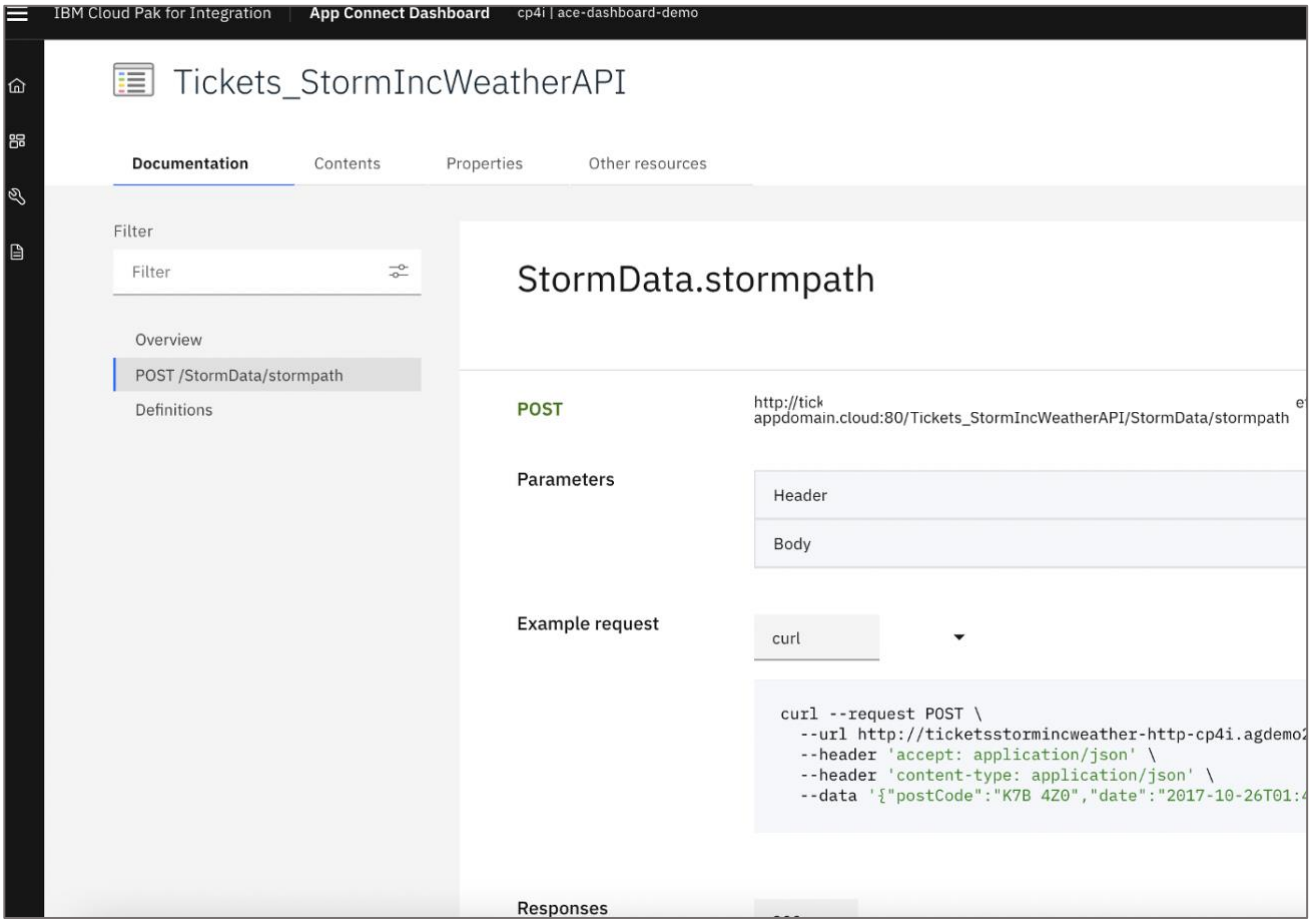



When you get refreshed screen, you will see the integration server displayed as a tile on the Servers page of the dashboard, with an initial status of Unavailable (⚠), which then changes to Started when the deployment completes. (so, DON’T PANIC! – this is the cloud pak spinning up 3 pods of the integration server – it won’t show a green tick until all the pods are running. Give it a couple of minutes or so and refresh your browser.)

The Servers page will also display any other integration servers that are installed in the same namespace.

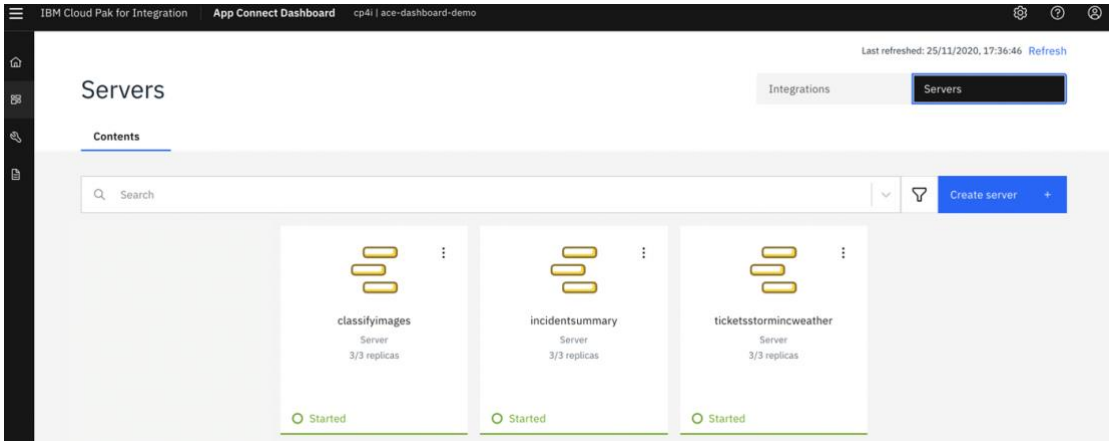
Switch to ‘Integration’ view and click the ‘Tickets_StormIncWeatherAPI’ tile to see further the details about your integration API.




You can see the REST operation; example test request and you can even download the OpenAPI (also called swagger) document. Hence, after you deploy an integration server to the cluster, you can view the underlying API definition. You can then test the API by using the built-in testing facilities. (learn more)

Repeat the same steps for ‘IncidentSummary’ and ‘classifyImagesV4’ assets. Eventually you will have all the integration APIs deployed as shown below.




Now would be a good time to test it again.

You can also use the below 'curl' request examples, where you need to replace with your hostnames. Also remember to use the incident number returned from the first call in the subsequent calls.

**1) Call ticketsstormincweather API**

```
curl --request POST \
  --url
http://REPLACE_ticketsstormincweather_HOSTNAME/Tickets_StormIncWeatherAPI/StormData/stormp
ath \
  --header 'accept: application/json' \
  --header 'content-type: application/json' \
  --data '{"postCode": "70510:US", "date": "2019-07-13", "name": "Alan ACE"}'
```

**2) Call classifyimages API**

```
curl --request POST \
  --url http://REPLACE_classifyimages_HOSTNAME/classifyImagesV4/images/classify \
  --header 'accept: application/json' \
  --header 'content-type: application/json' \
  --data '{"incident_id": "REPLACE_INCIDENTNUMBER", "image_name": "my damaged
car","image_url": "https://raw.githubusercontent.com/IBM/cp4i-demos/main/ace-weather-
chatbot/images/car.jpg"}'
```

**3) Call incidentsummary API**

```
curl --request GET \
  --url
http://REPLACE_incidentsummary_HOSTNAME/IncidentSummary/summary/REPLACE_INCIDENTNUMBER \
  --header 'accept: application/json'
```

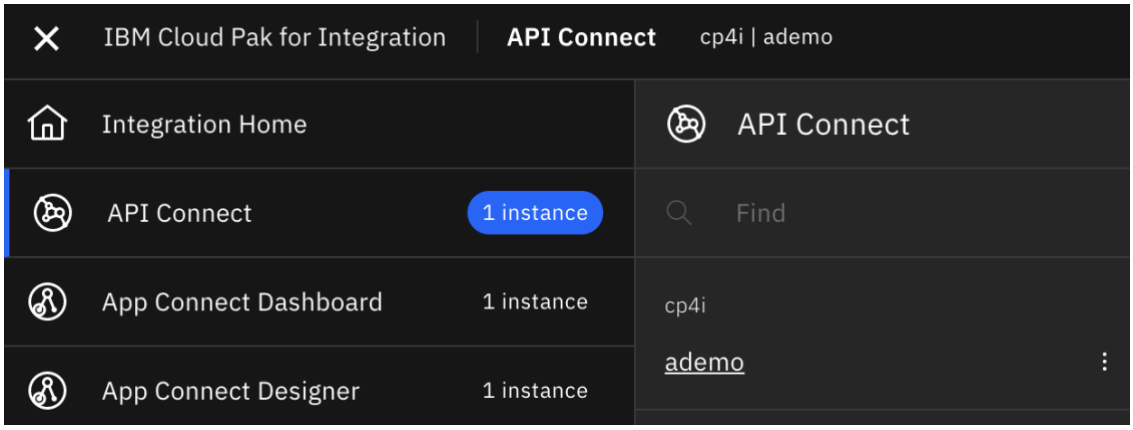
# 7 Managing our API using API Connect

Now it is the time to configure our facade API in API Connect, let's go there and do some API Management.

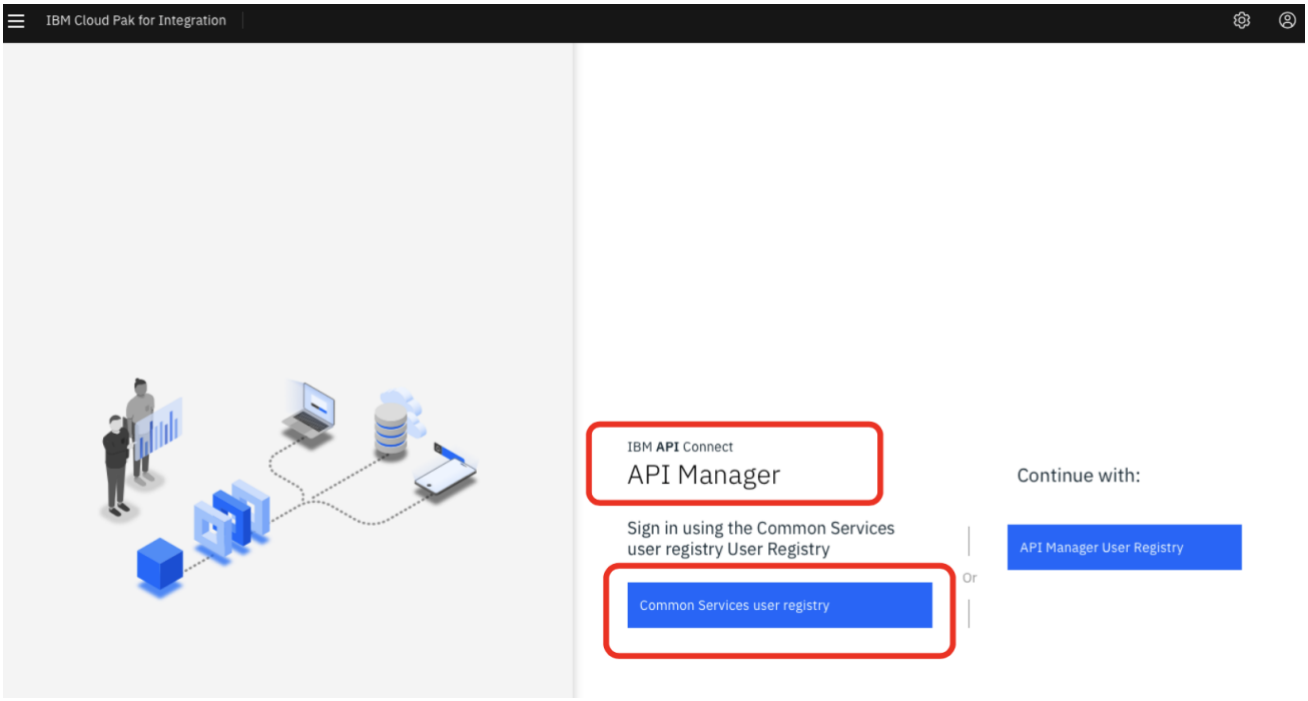
We want to be able to add security, define some rate-limiting plans and publish the API to a secure gateway.

In addition we want to be able to use a self-service portal so that consumers can browse our APIs and sign up to use them.

Using the hamburger menu, click on 'ademo'. This time, click on the name – we want to go to the API manager, not the APIC cloud manager…




You'll be asked to log into the API Manager. Click on 'IBM Common Services user registry'. You should be logged in automatically using SSO. If not, use 'admin' and the same password you used to log into the cloud pak home page.



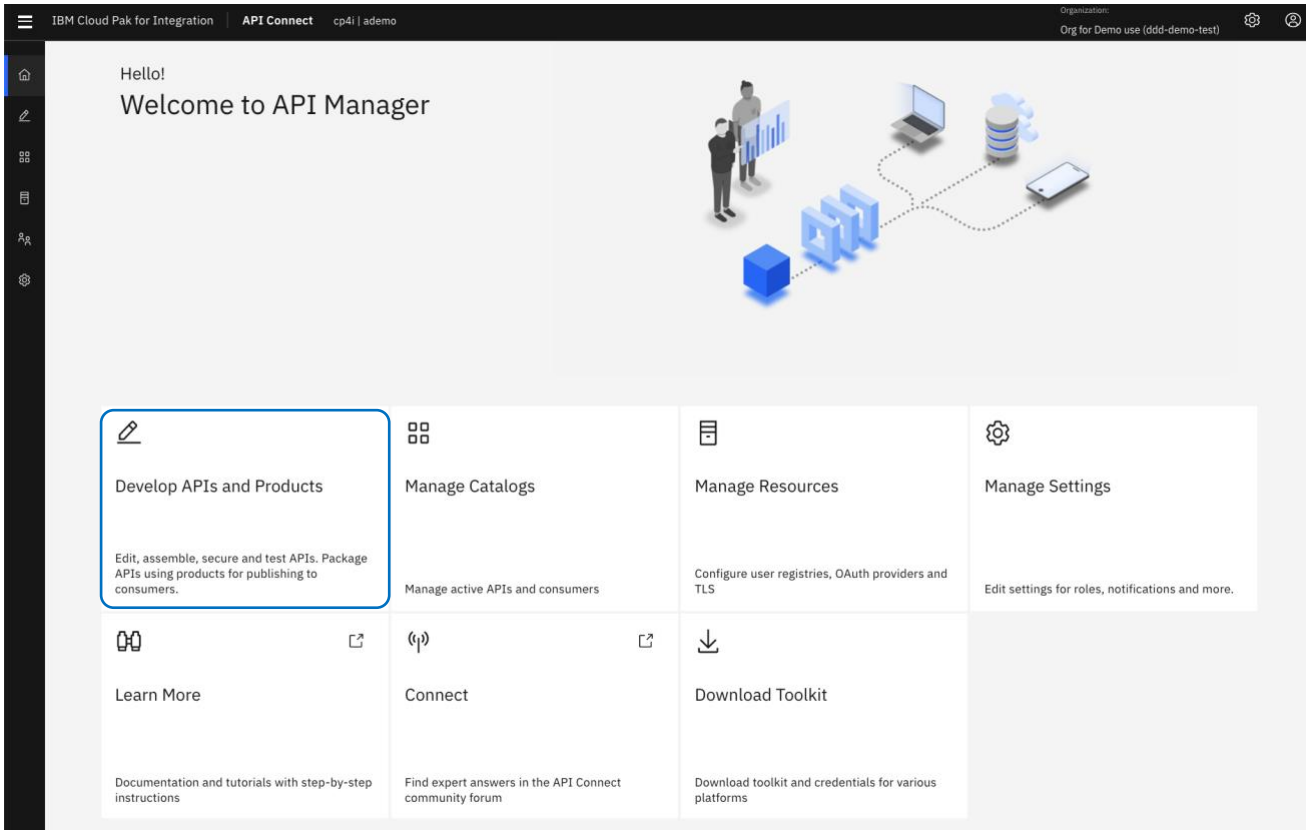

Make sure it says ‘Welcome to API Manager’.

Also check out the organisation at the top-right: Make sure it says ‘Org for Demo use’ – If it doesn’t, click on it and change it so it does.

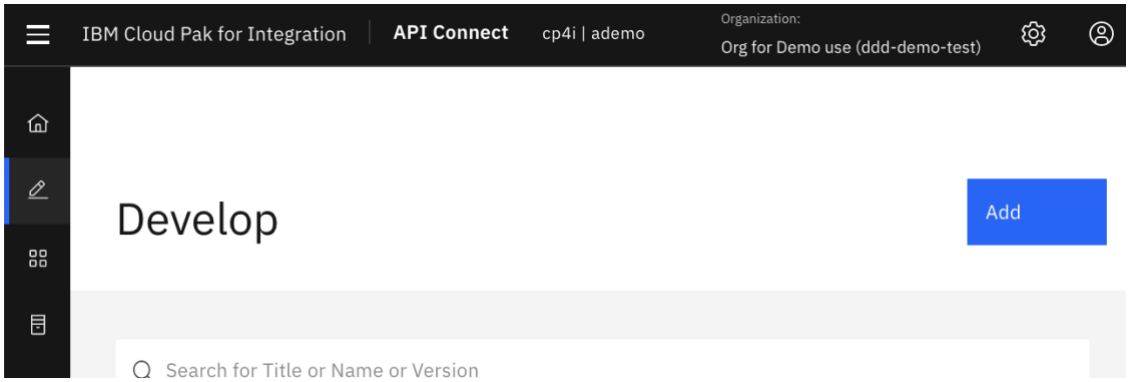
## 7.1 Importing our Facade API flow into API Manager

We’re going to import our API flow from the Asset Repository: The 1-click install has put it there for you…

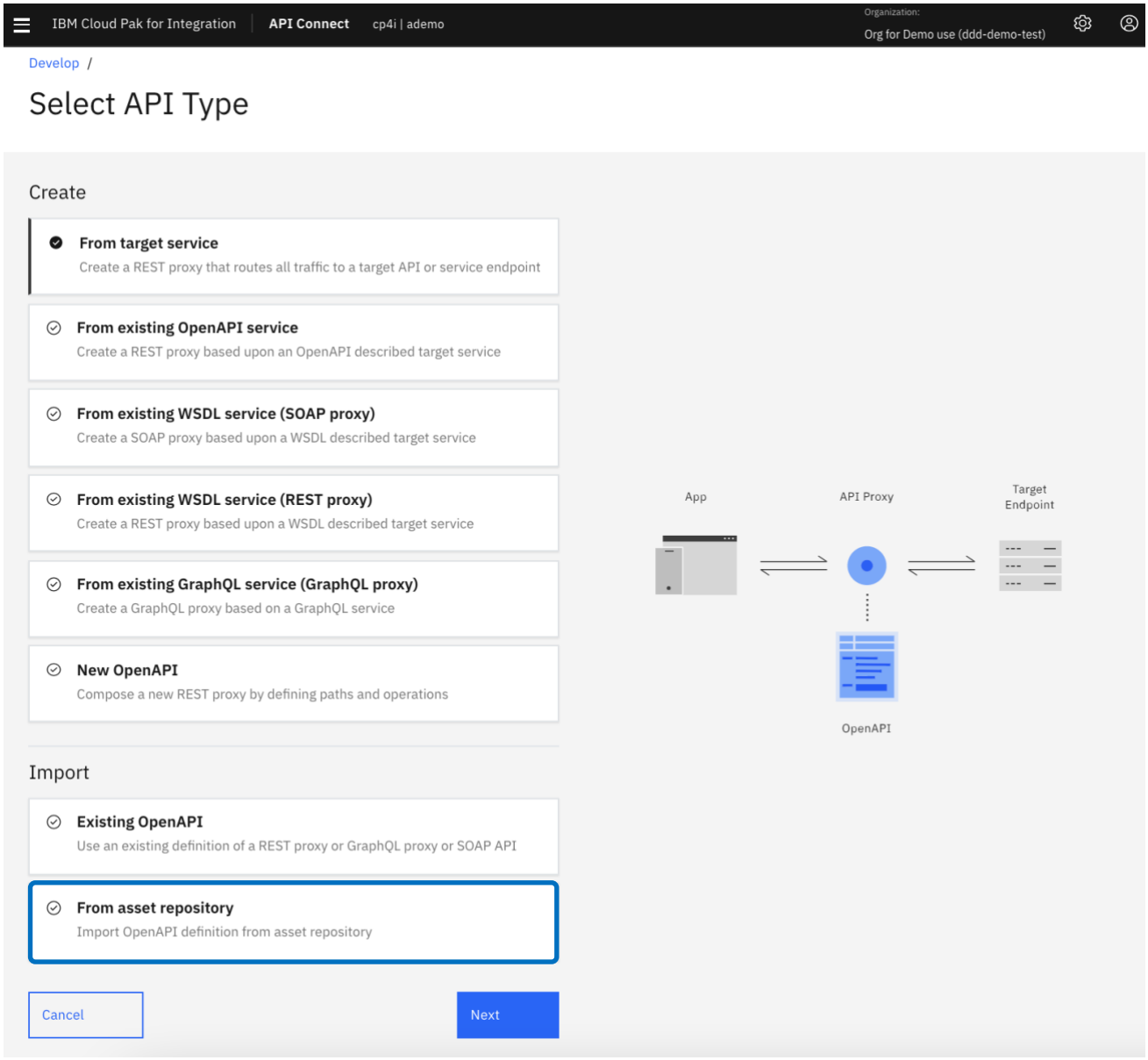
Click ‘Develop APIs and Products’ tile.




Click ‘Add’.

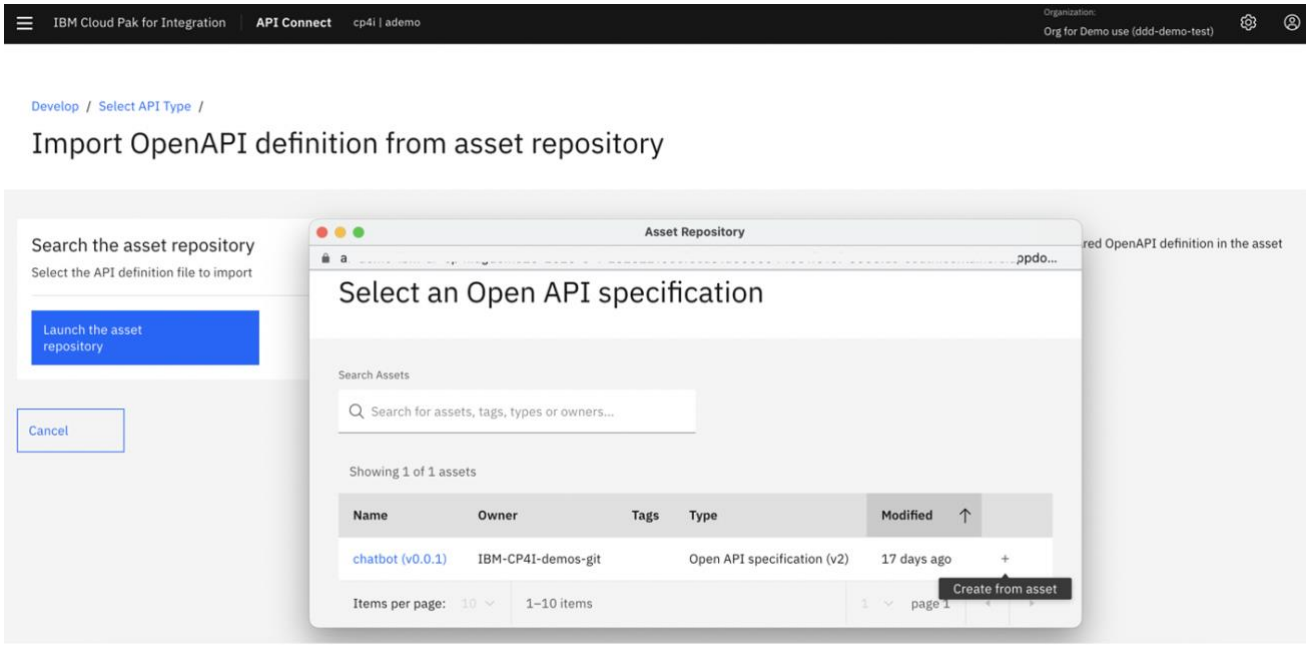



Select ‘From asset repository’ and click ‘Next’



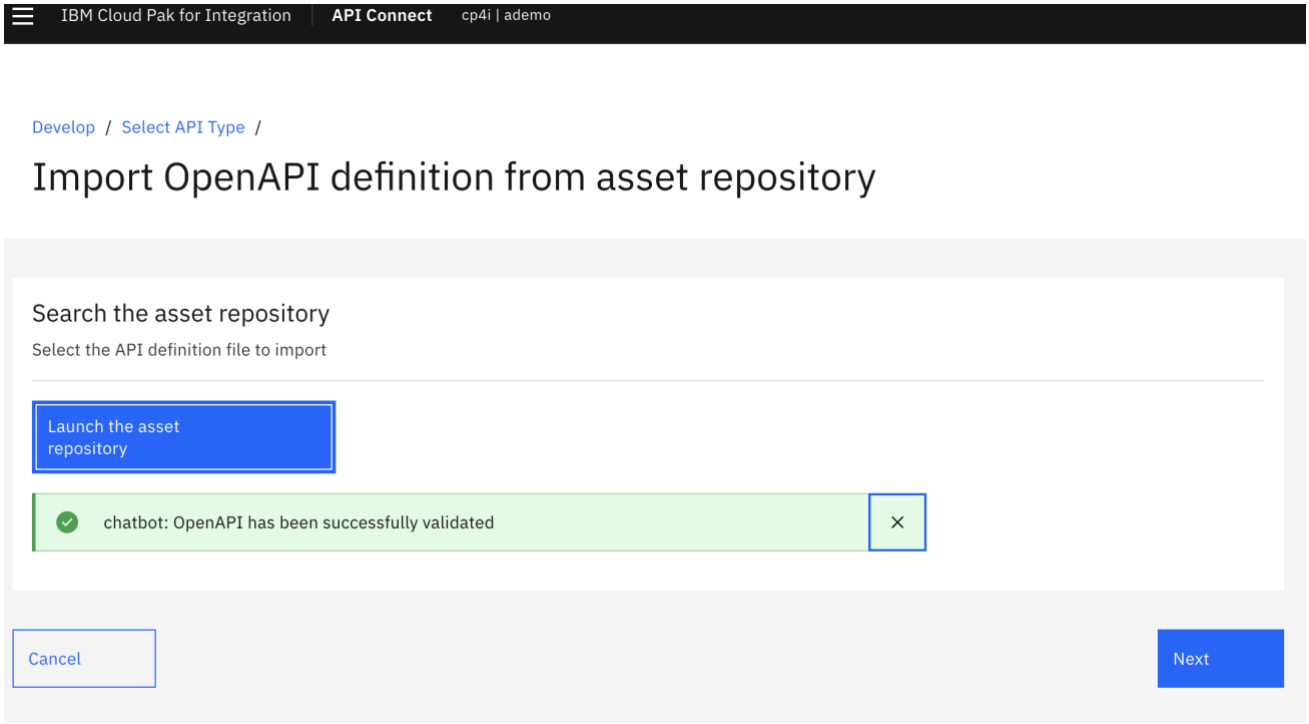

In the next window you will be provided an option to launch the asset repository. When you click on it a pop-up window opens. You should be logged in automatically using SSO. If not, use ‘admin’ and the same password you used to log into the cloud pak home page. which might ask you to login again.

Select the ‘chatbot’ Open API specification.



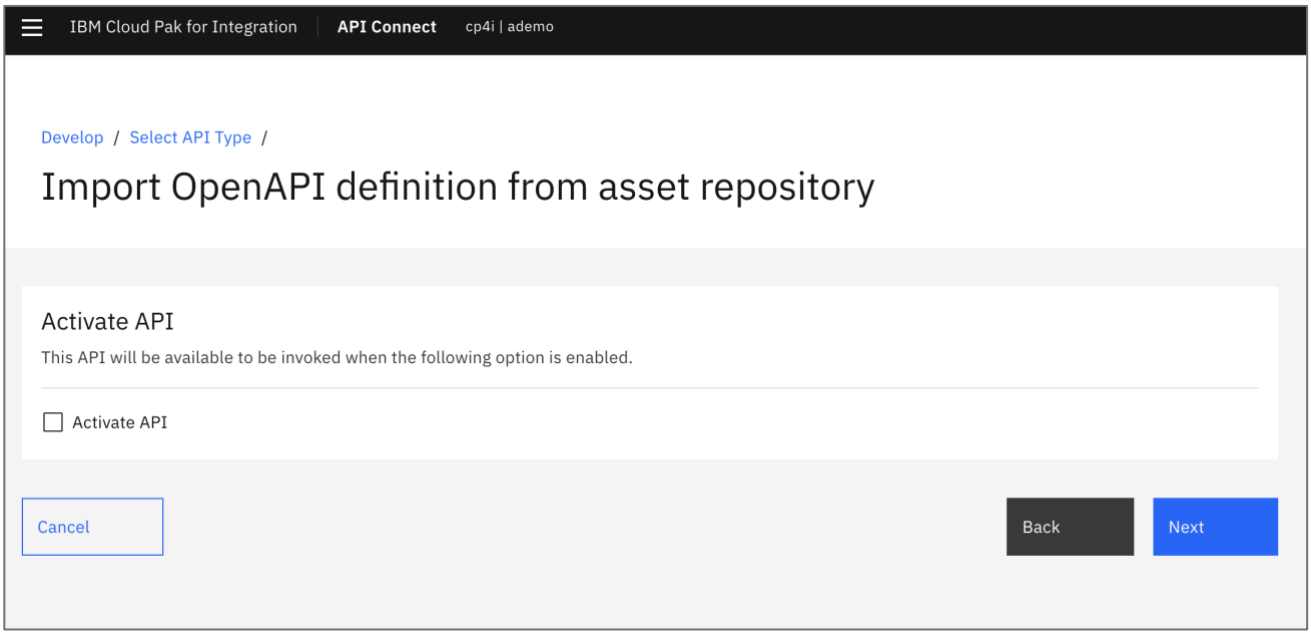


Once our asset is validated, we can click ‘Next’ to proceed with the import.




In the next window, you can configure to Activate API option which creates a draft Product, adds the API to the Product, and publishes the Product to the Sandbox Catalog so that the API is available to be called. We want to only publish to our demo catalog, so will not select this option.

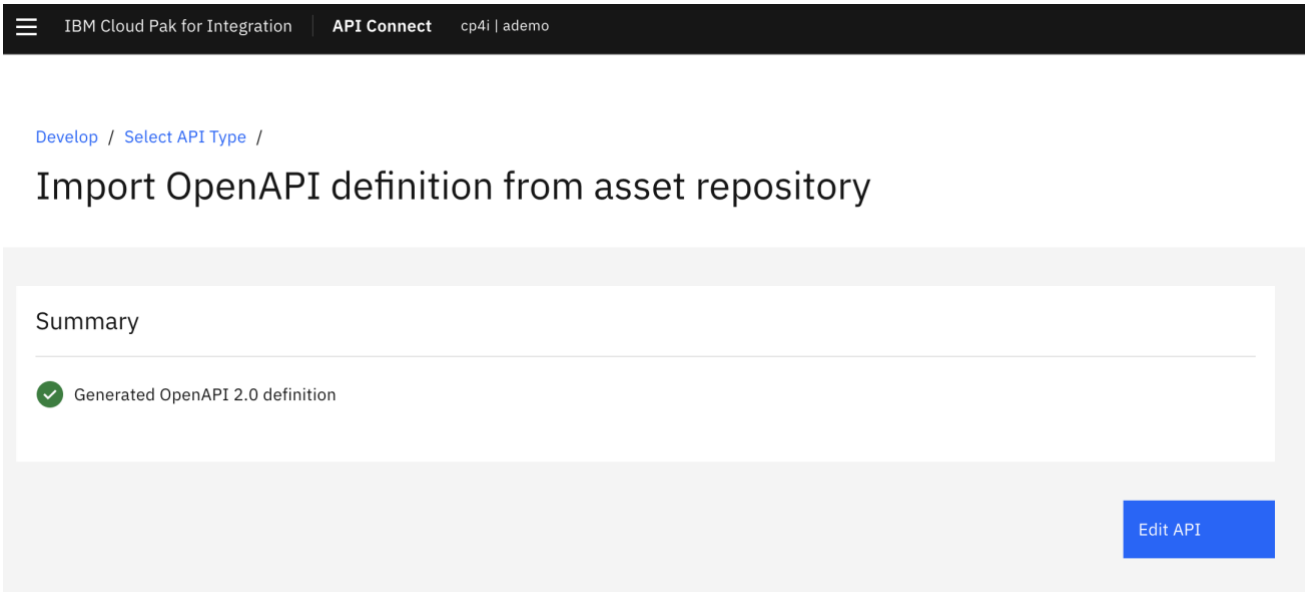
Click ‘Next’.




## 7.2 Reviewing and editing our Facade API Flow

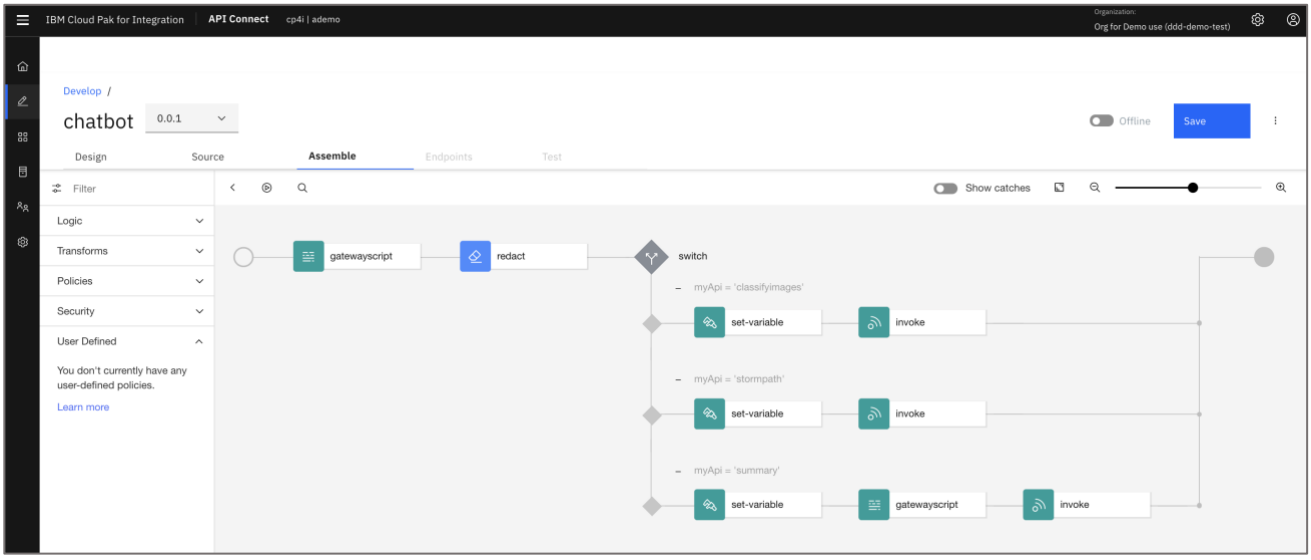
The Import API Summary panel indicates that the YAML file is loaded and valid.

Click ‘Edit API’.

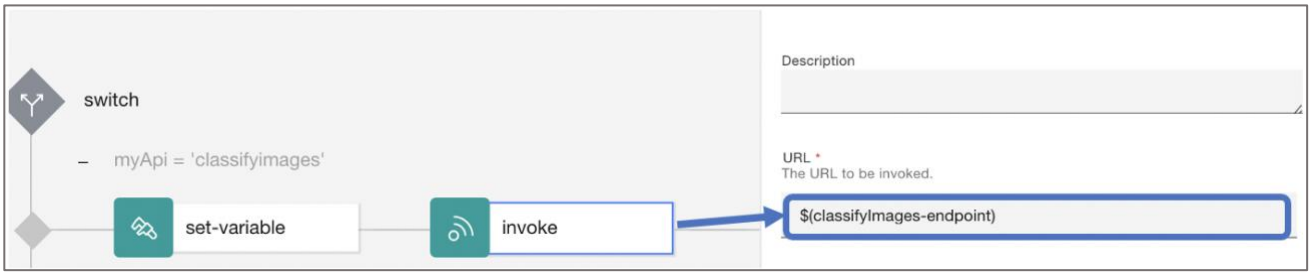



You can switch to the ‘Assemble’ tab to view the implementation details.

This is a simple facade API secured by an API key which routes the incoming requests to our integration APIs based on request content. But you can always enrich the flow additional error handling or security constructs depending on your requirements. For the purpose of this demo, we will stick to our simple façade API ☺

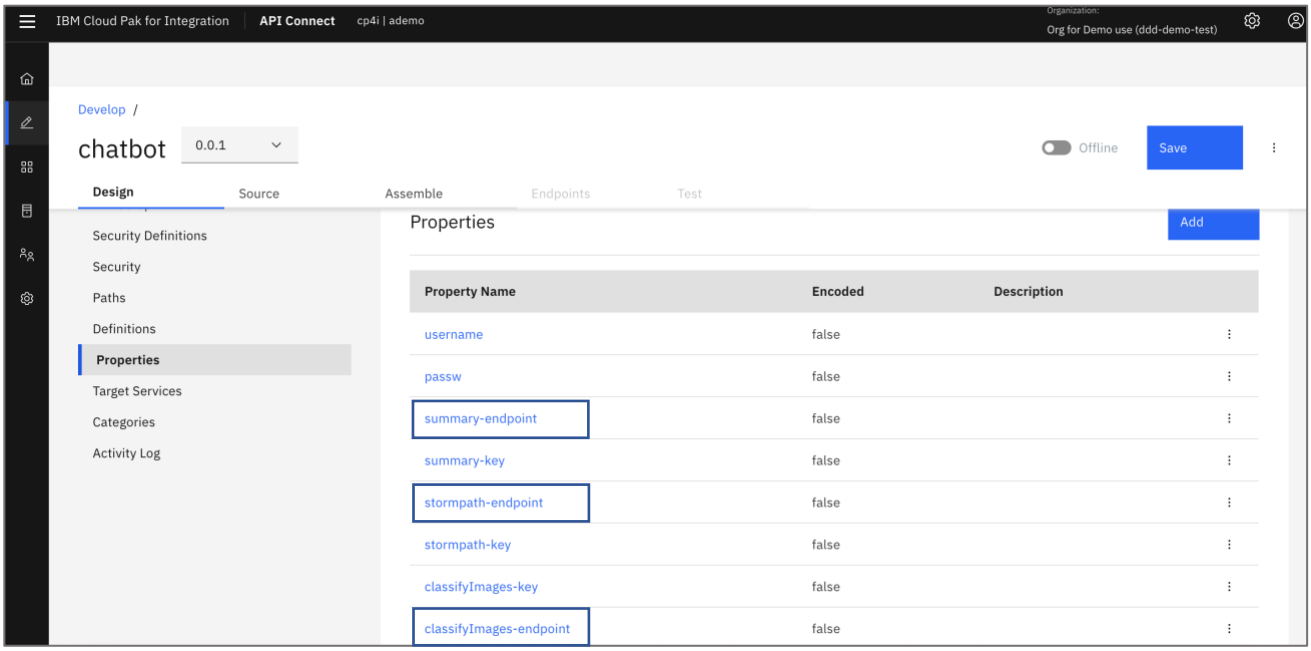



Each invoke policy uses property values to specify the URL for our target integration APIs. (such as classifyImages-endpoint)

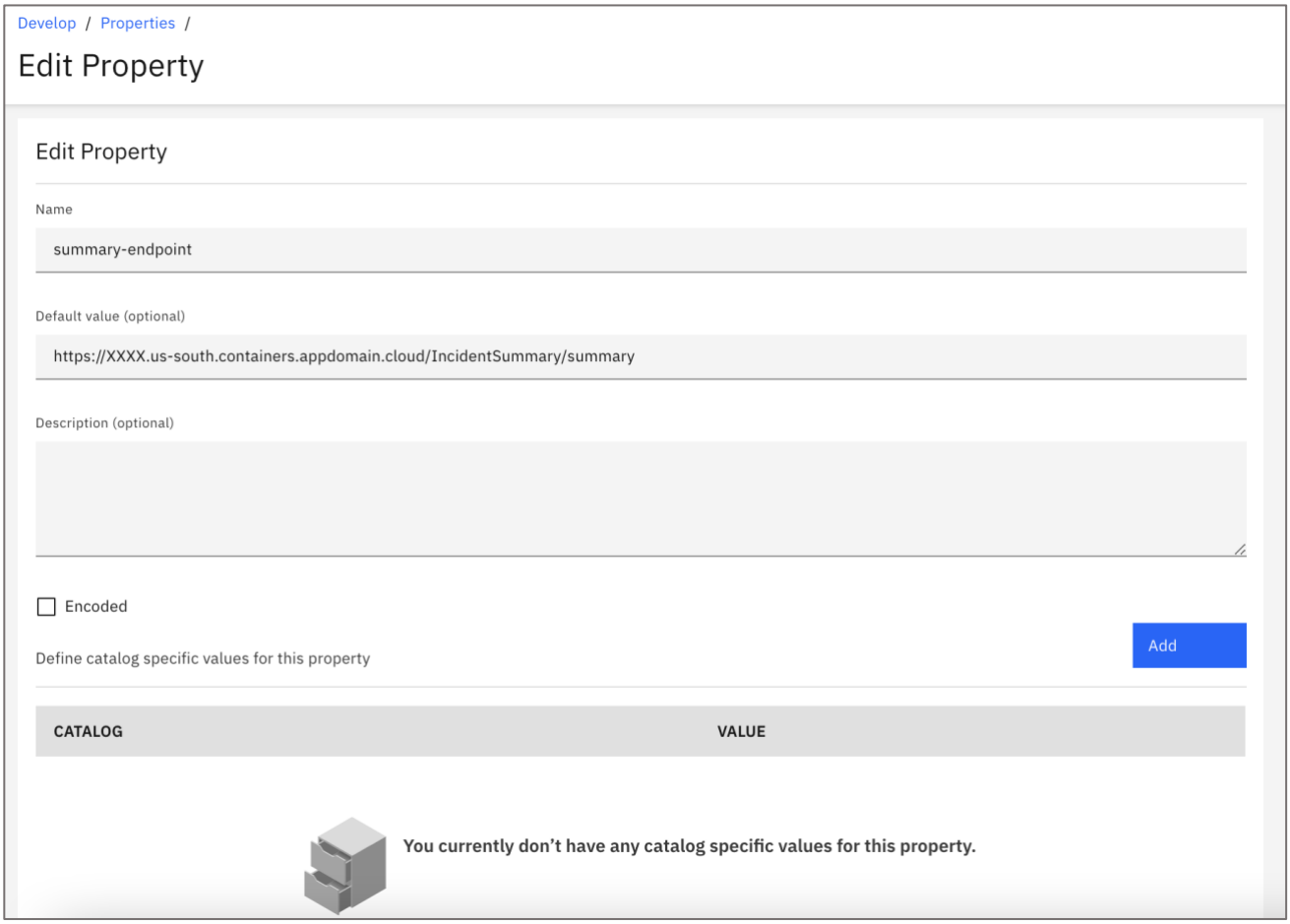



Next, we need to update these endpoint properties to point to our deployed integration APIs.

Switch to ‘Design’ tab and select ‘Properties’.




First, click ‘summary-endpoint’ property and click ‘Add’ to define a catalog specific value.




Select our demo catalog and type the endpoint value of the ‘IncidentSummary API’.


Make sure you use the correct endpoint value here.

It should be in the following format:

http://<incidentsummary-hostname>/IncidentSummary/summary

You can check the value of the hostname from App Connect Dashboard.

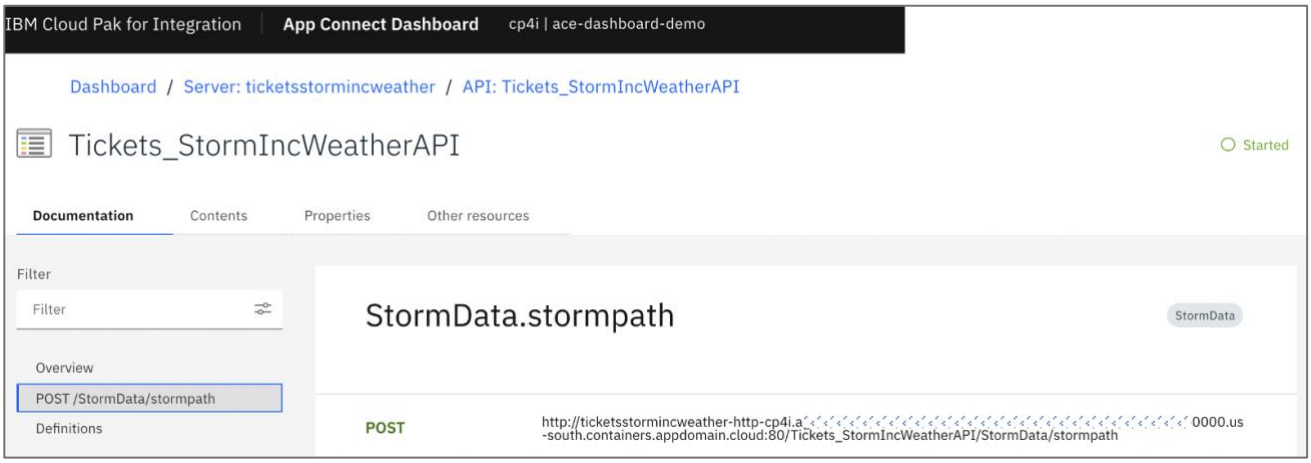

Next, click ‘stormpath-endpoint’ property and click ‘Add’ to define a catalog specific value. Select our demo catalog and type the endpoint value of the ‘Tickets_StormIncWeatherAPI’.

Make sure you use the correct endpoint value here.

It should be in the following format:

http://<stormpath-hostname>/Tickets_StormIncWeatherAPI/StormData/stormpath

You can check the value of the hostname from App Connect Dashboard.




Lastly, click ‘classifyImages-endpoint’ property and click ‘Add’ to define a catalog specific value. Select our demo catalog and type the endpoint value of the ‘classifyImagesV4 API’.

Make sure you use the correct endpoint value here.

It should be in the following format:

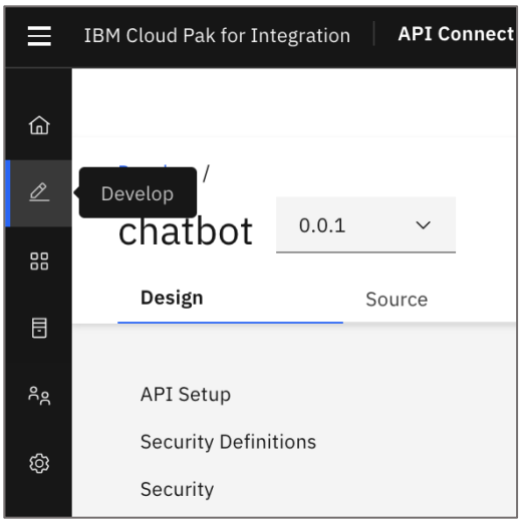
http://<classifyImages-hostname>/ classifyImagesV4/images/classify

You can check the value of the hostname from App Connect Dashboard.

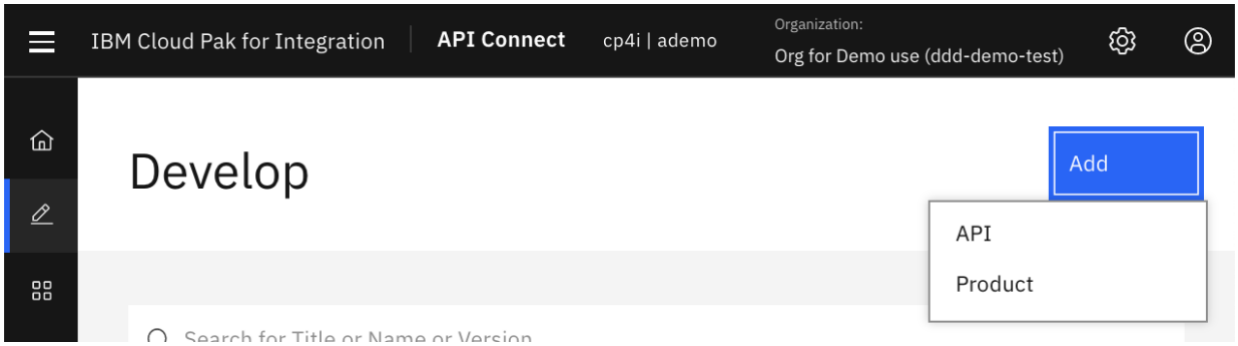

We finished editing our API, lets create a product! A product is a way of grouping together APIs. Consumers subscribe to products rather than individual APIs.

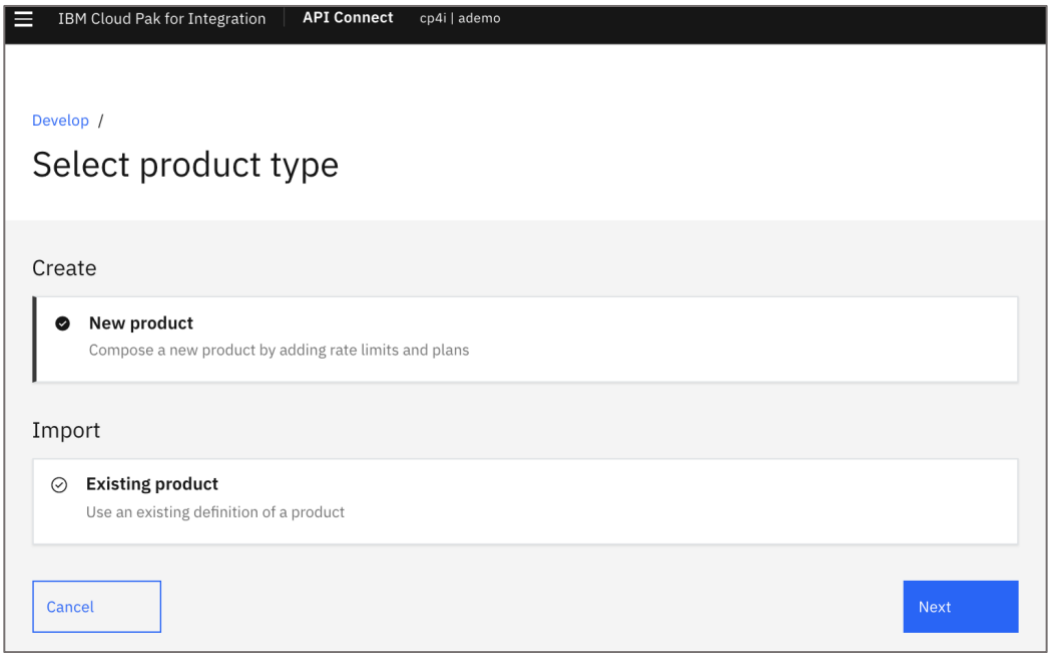
## 7.3 Publishing our Facade API Flow

Click the ‘Develop’ menu on the left.

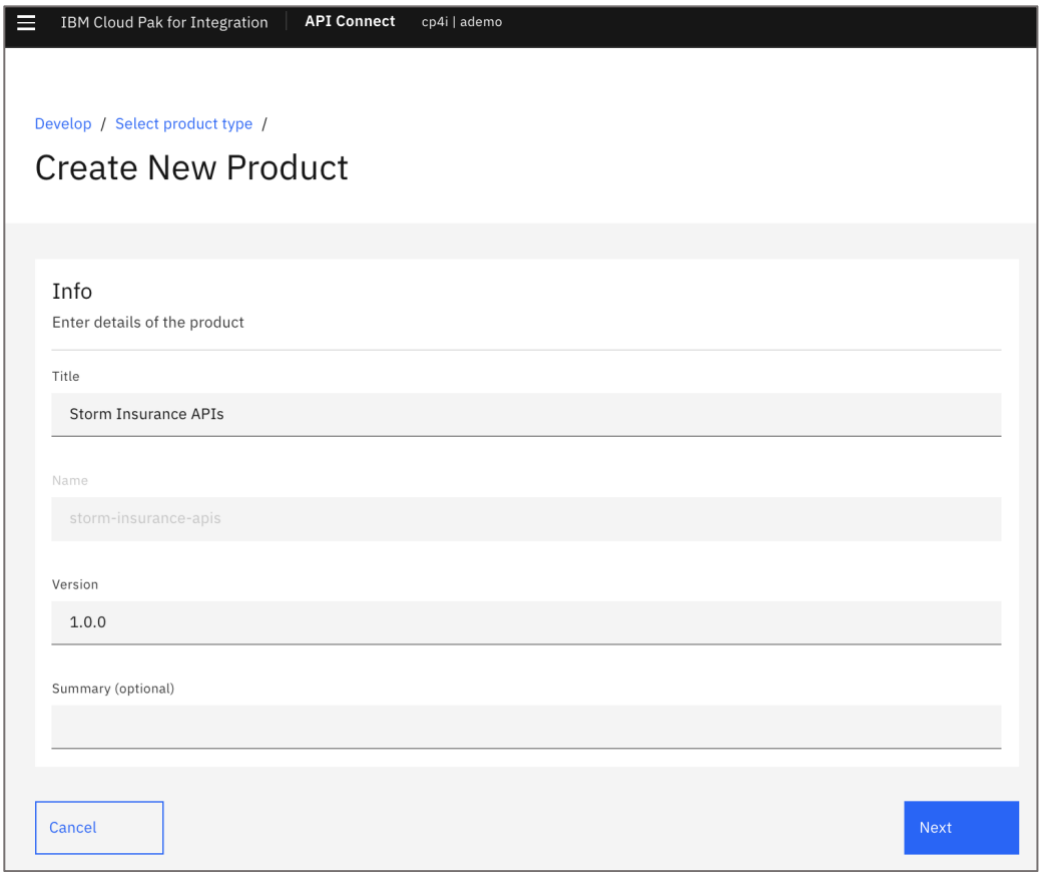



Click ‘Add’ and select ‘Product’.

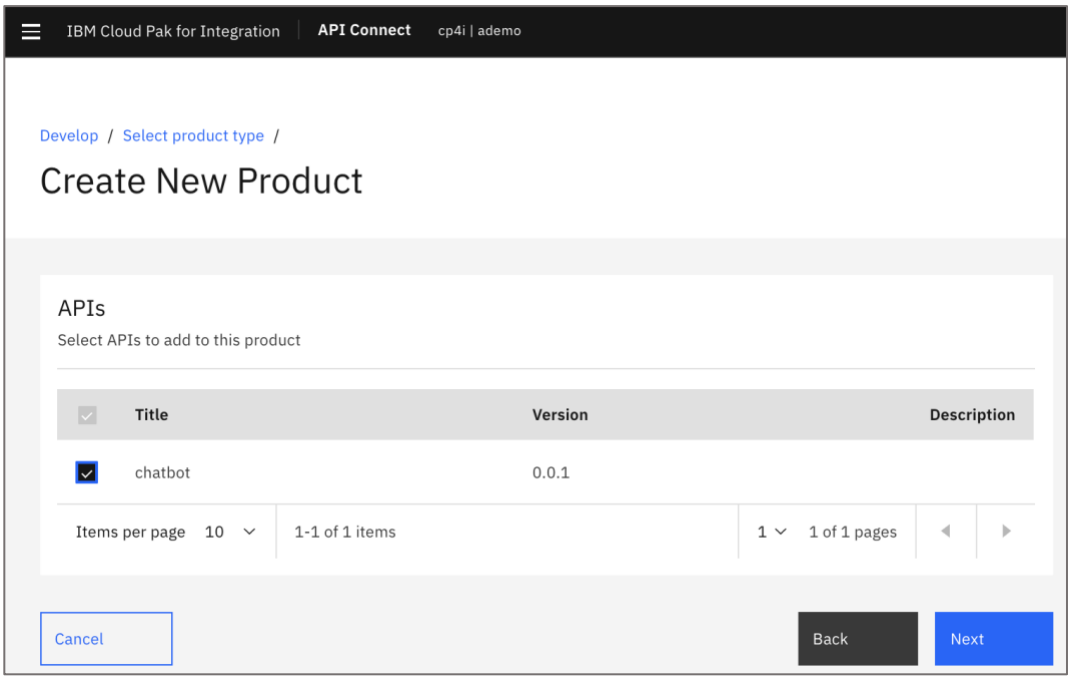


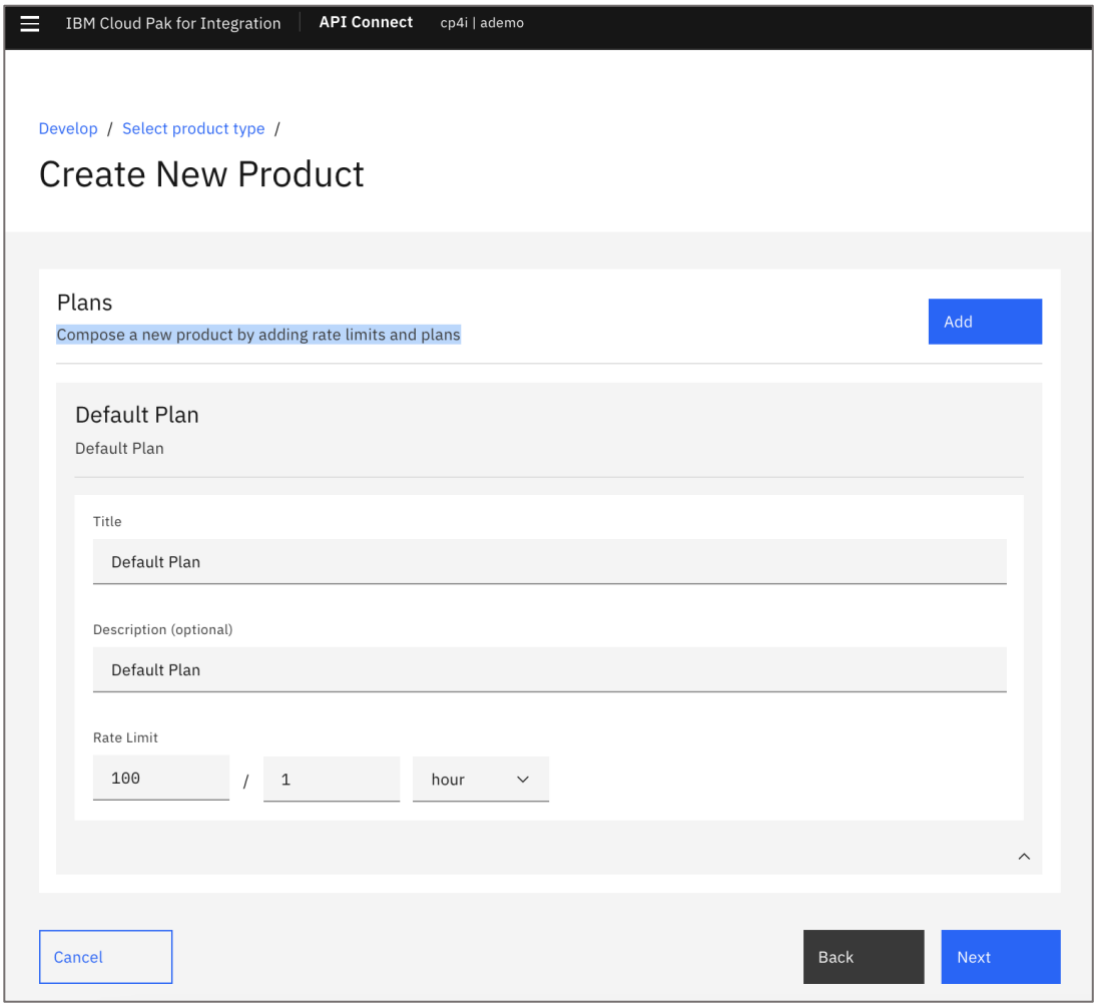

Enter a product name such as ‘Storm Insurance APIs’ and a version ‘1.0.0’ will do.



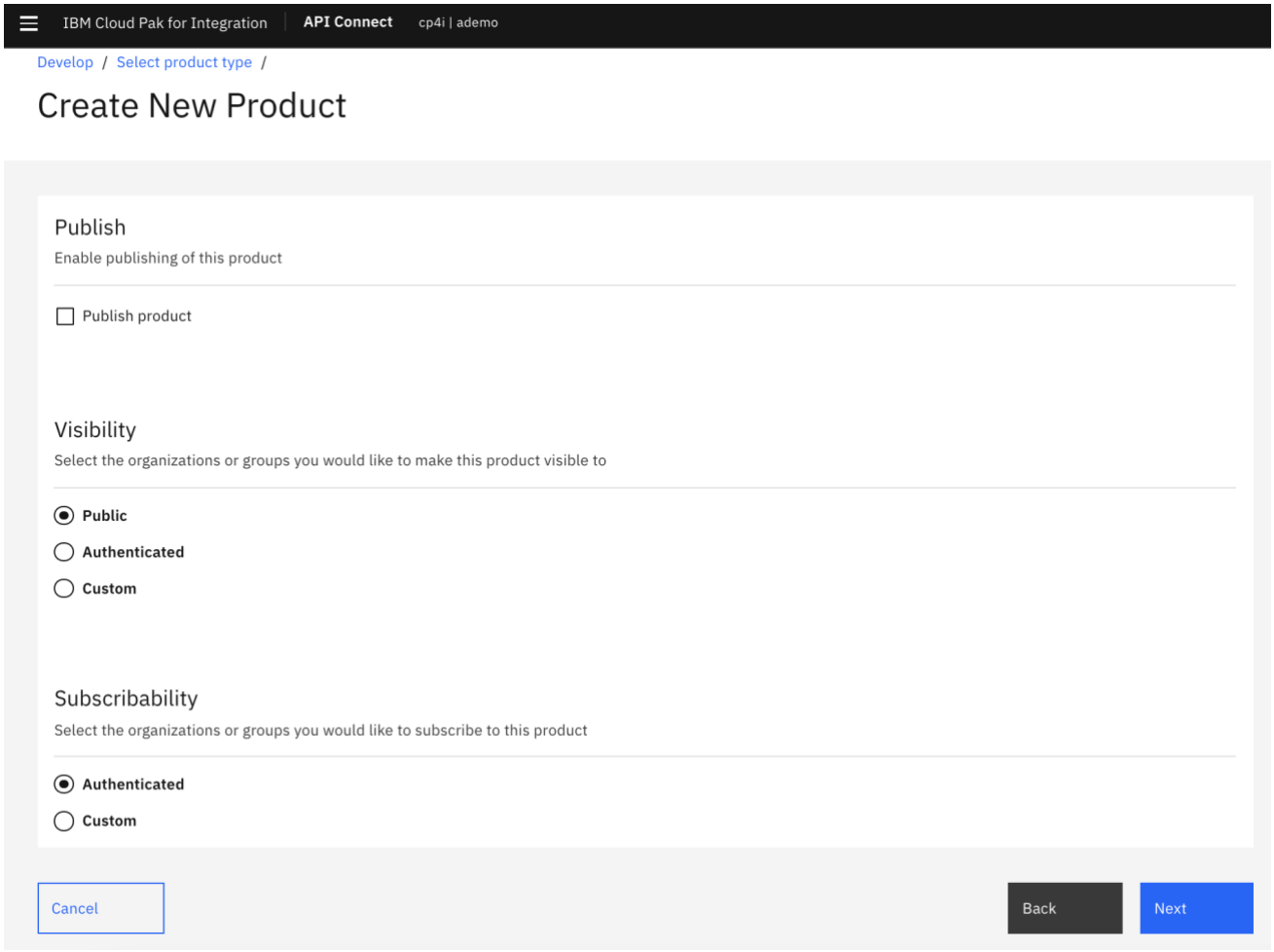

Select ‘chatbot’ API to add to the product and click ‘Next’



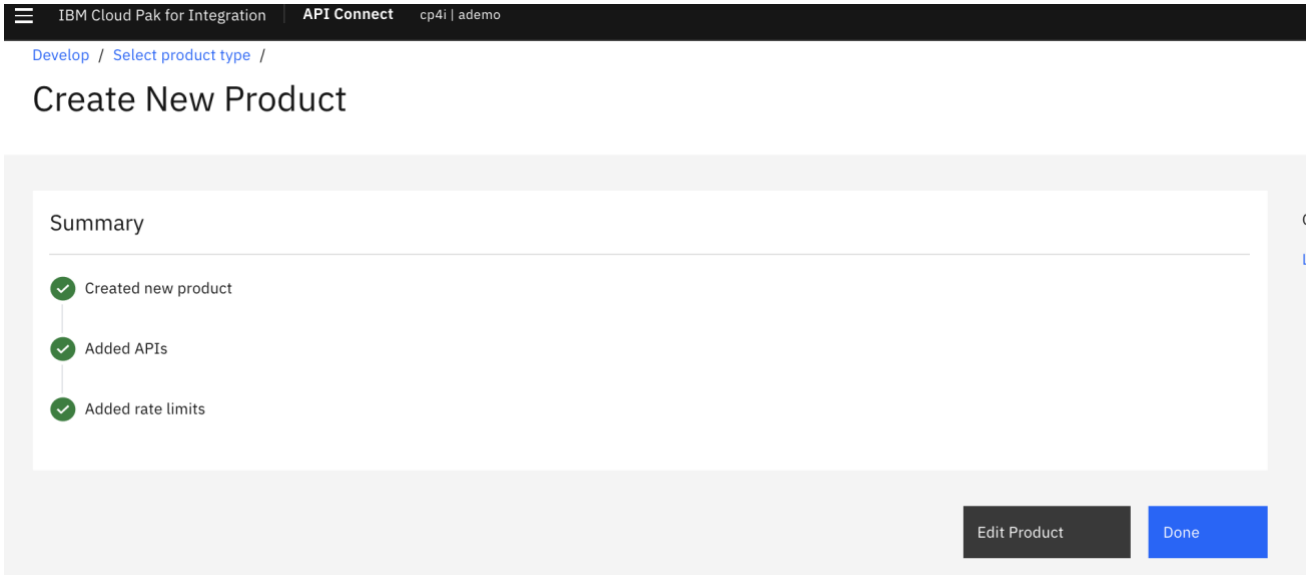

You can add multiple rate limits and plans. But for now, Default plan will do.



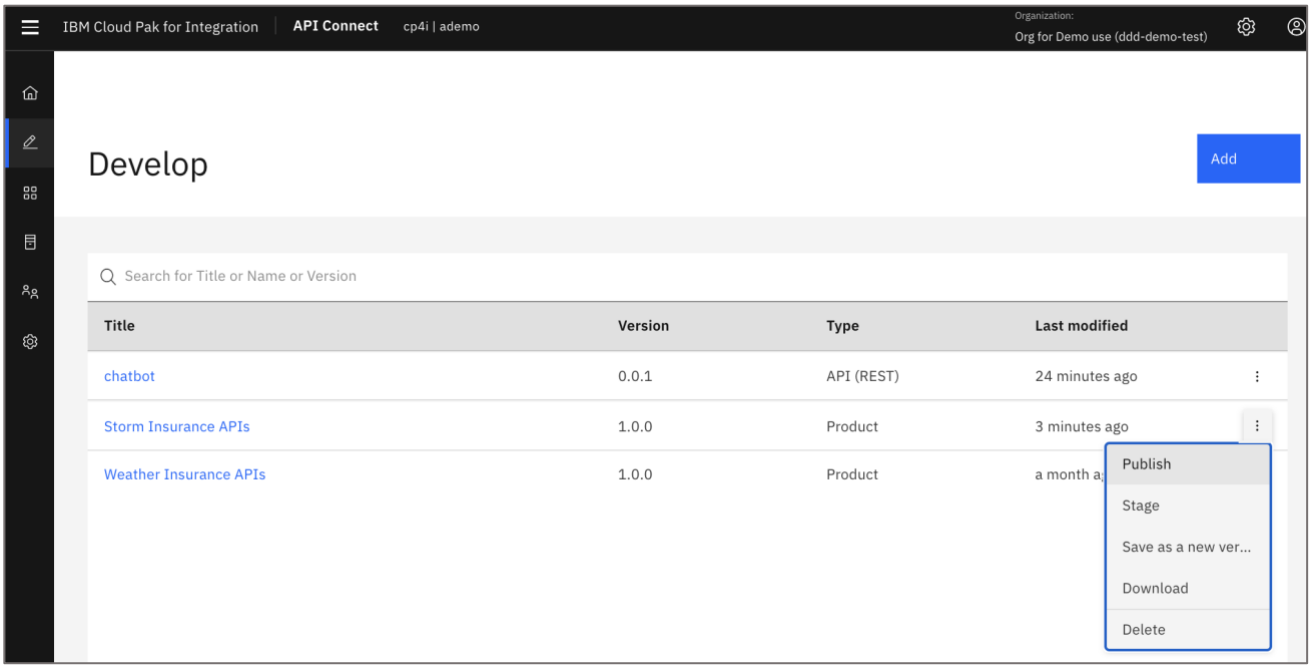

Finally configure the ‘visibility and subscribability’ settings. You can leave the default settings. We will leave the ‘Publish product’ checkbox empty as we want to publish to our demo catalog later.



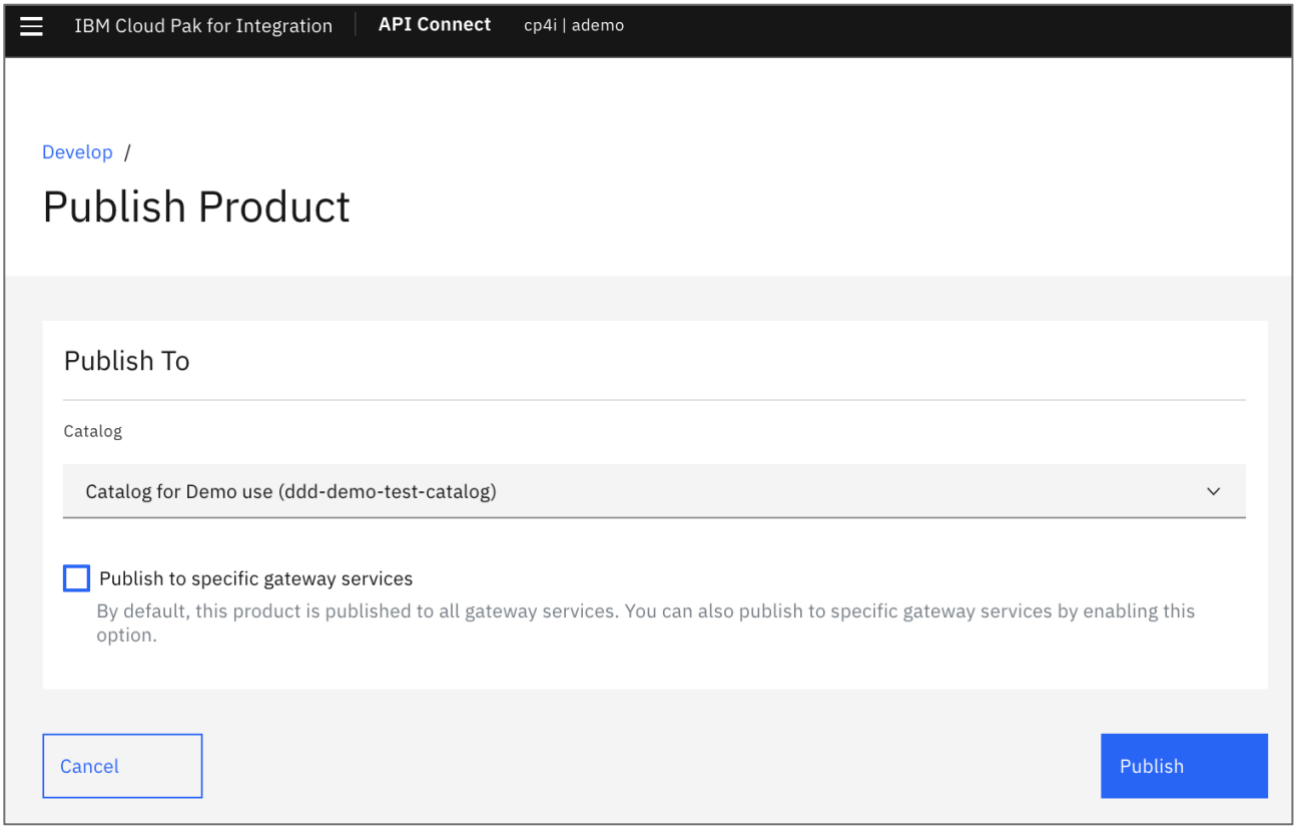

Click ‘Done’. You will be redirected to the ‘Develop’ page. Now, we are ready to publish to the Portal!




Now click on the three-dot overflow menu by the ‘Storm Insurance APIs’ product and click ‘publish’.

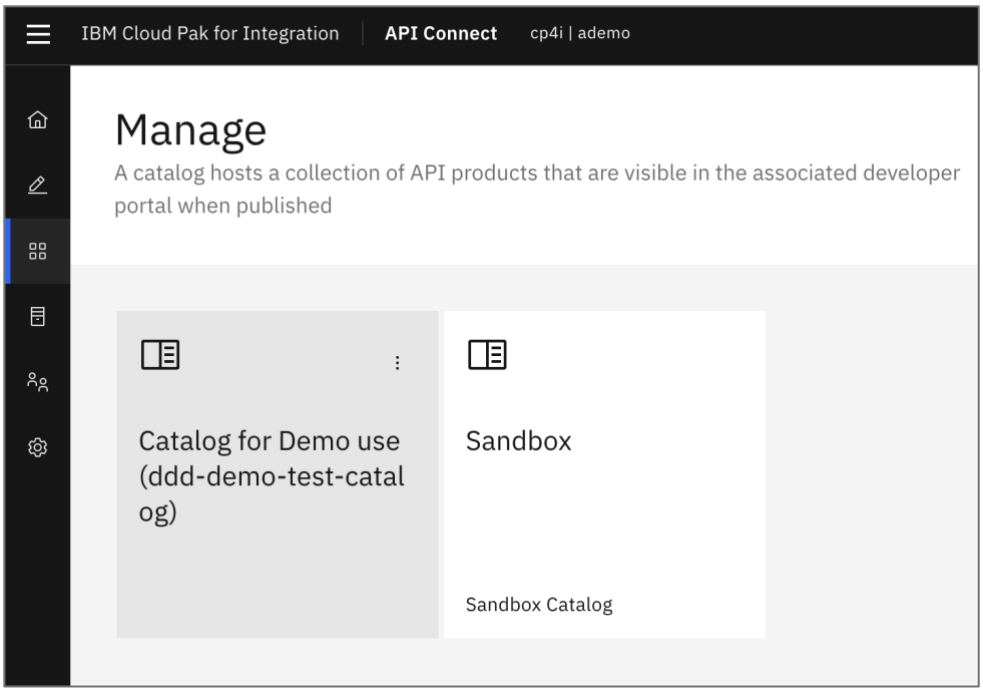



You'll be prompted for a catalog to publish to – select 'Catalog for Demo use'. We only have one gateway installed so we can leave the checkbox blank – click 'Publish'



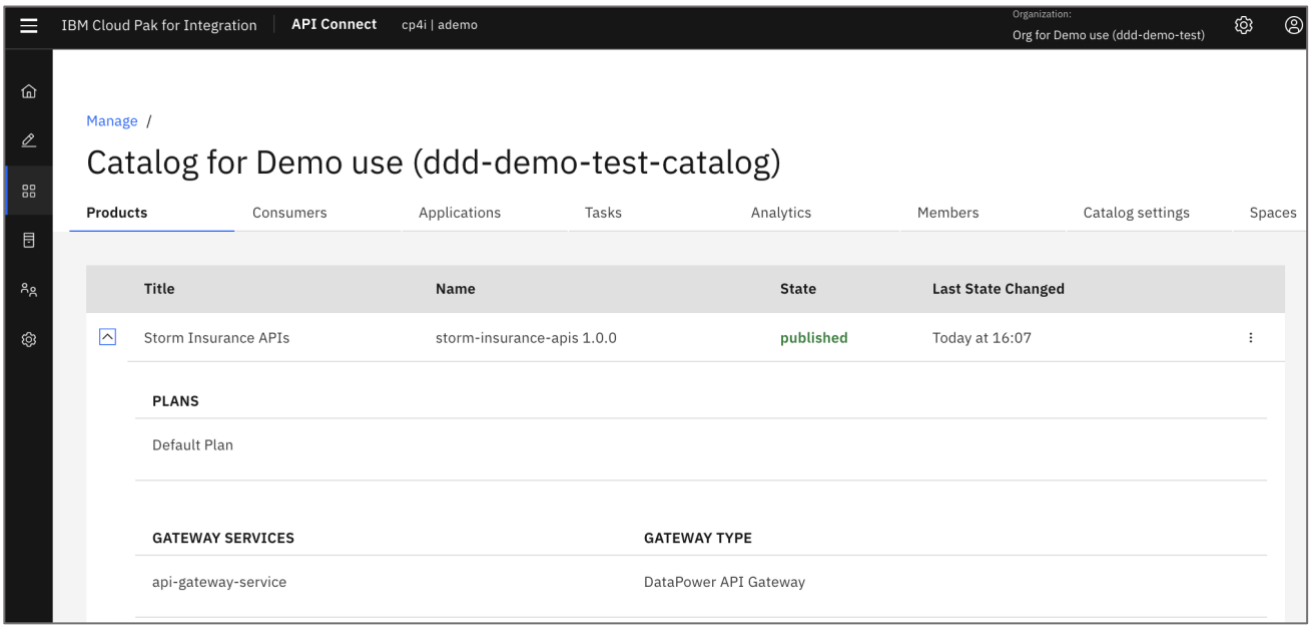

You will see a notification once the publish finishes. (in seconds)

If you now go back to your catalog and look for products, you can see the status is 'published' (go to the 'Manage' menu and then click on the Catalog for Demo use)




You can see our Default plan added into the product. You can also see that we've published our API to a secure DataPower Gateway.

The gateway has been configured as an APIC gateway service and bound to the catalog as part of the 1-click demo installation for this lab.

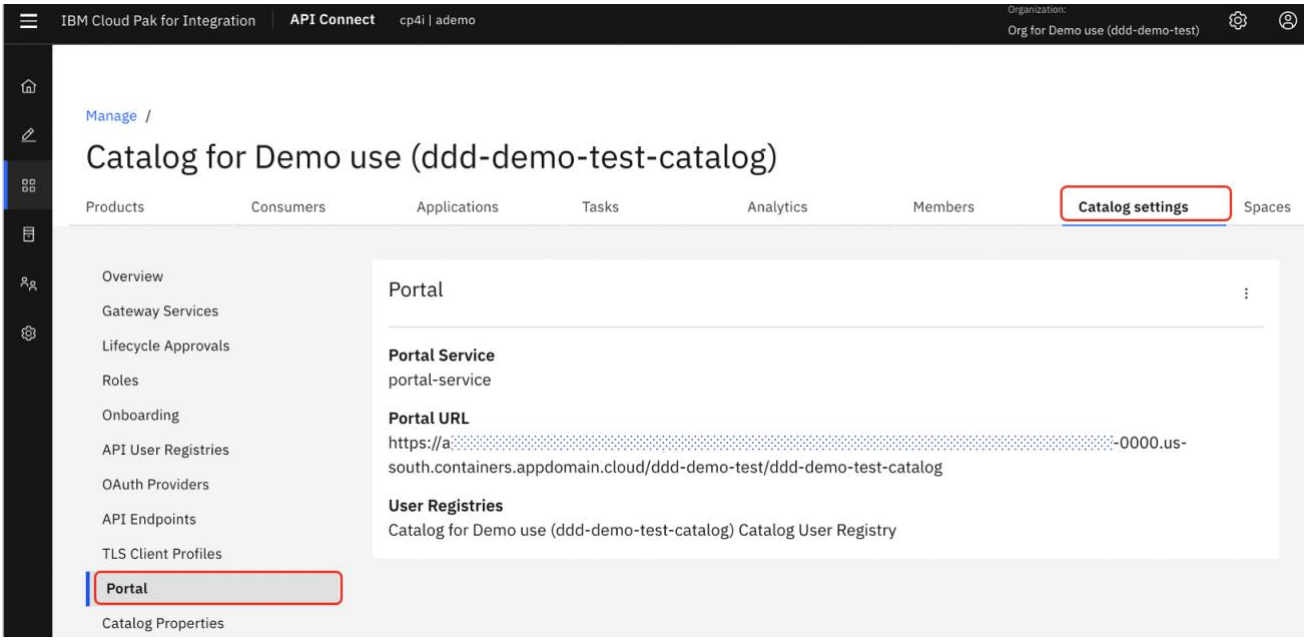


## 7.4 Discovering and consuming our API

Now that we've published our API, we need to make sure that our API consumers can discover it and use it.

Our Portal will allow customers to view the APIs, sign up and subscribe to plans in a self-service manner, test the APIs, download the OpenAPI / Swagger documents and more.

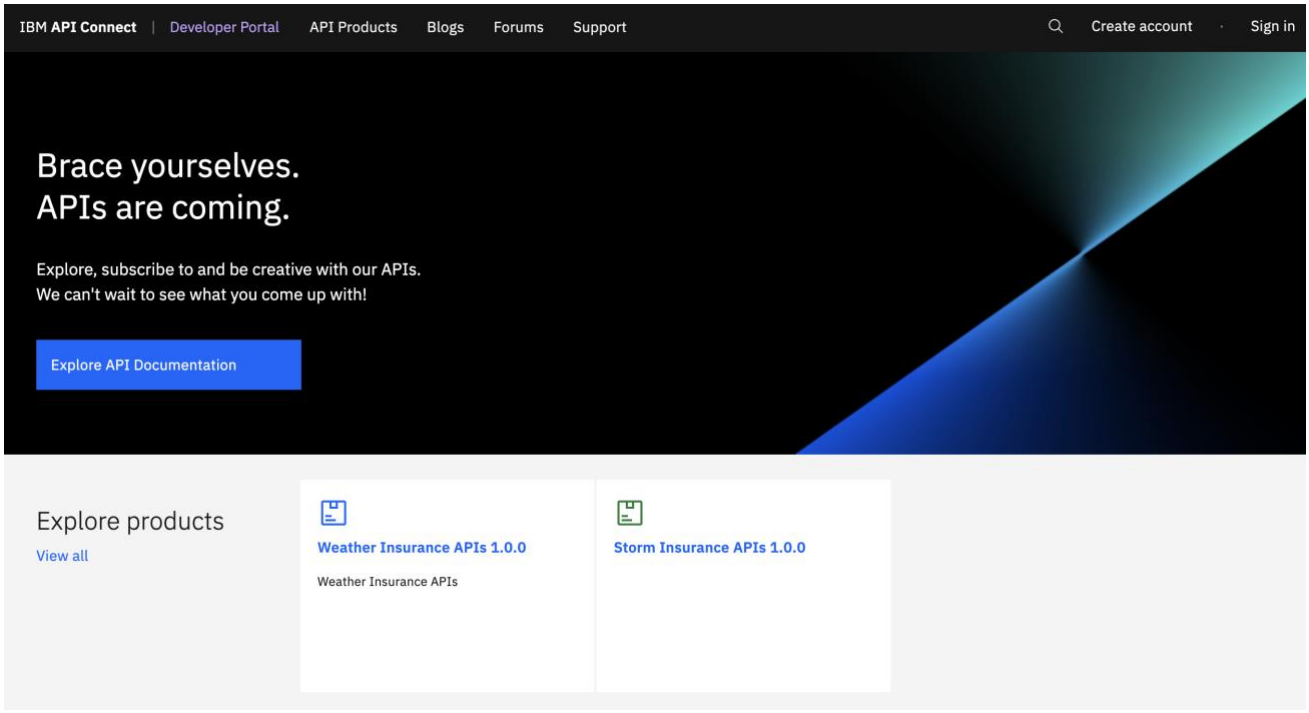
Click 'Catalog settings' and 'Portal'.



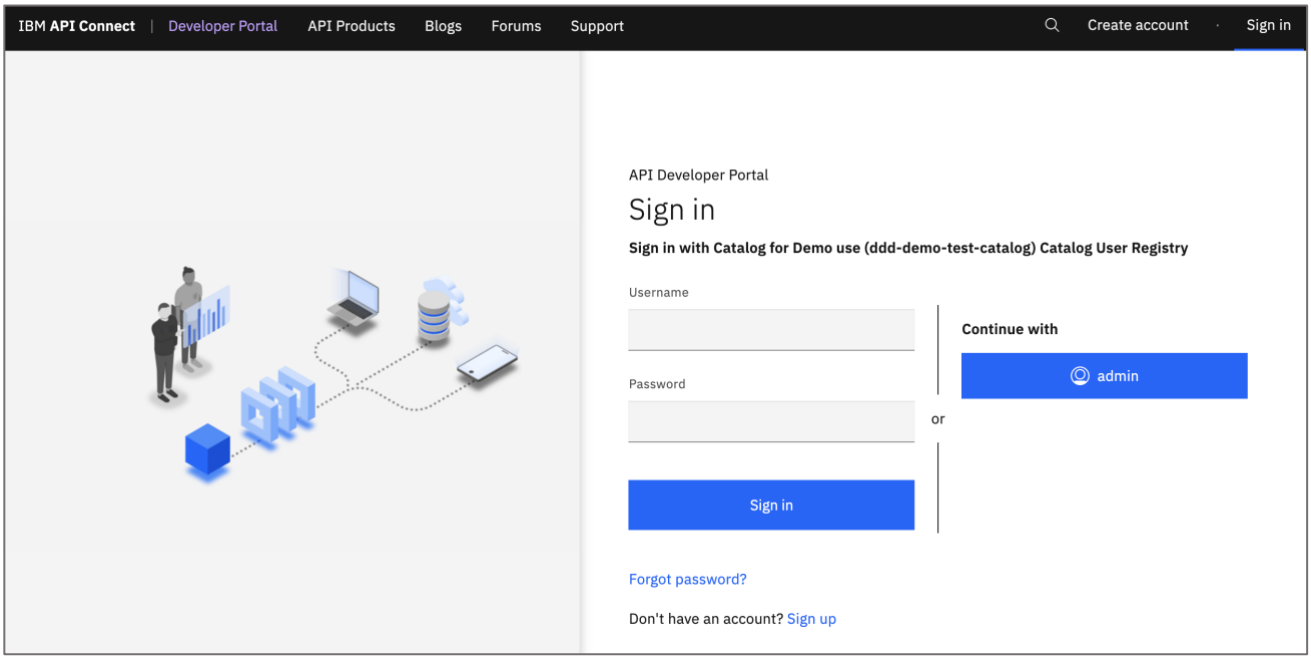

You can see that a portal service has been created for you as part of the 1-click demo installation.

You can directly access to the Portal URL from your browser. Notice that 'Storm Insurance APIs' are already visible as we set the visibility as 'public'.

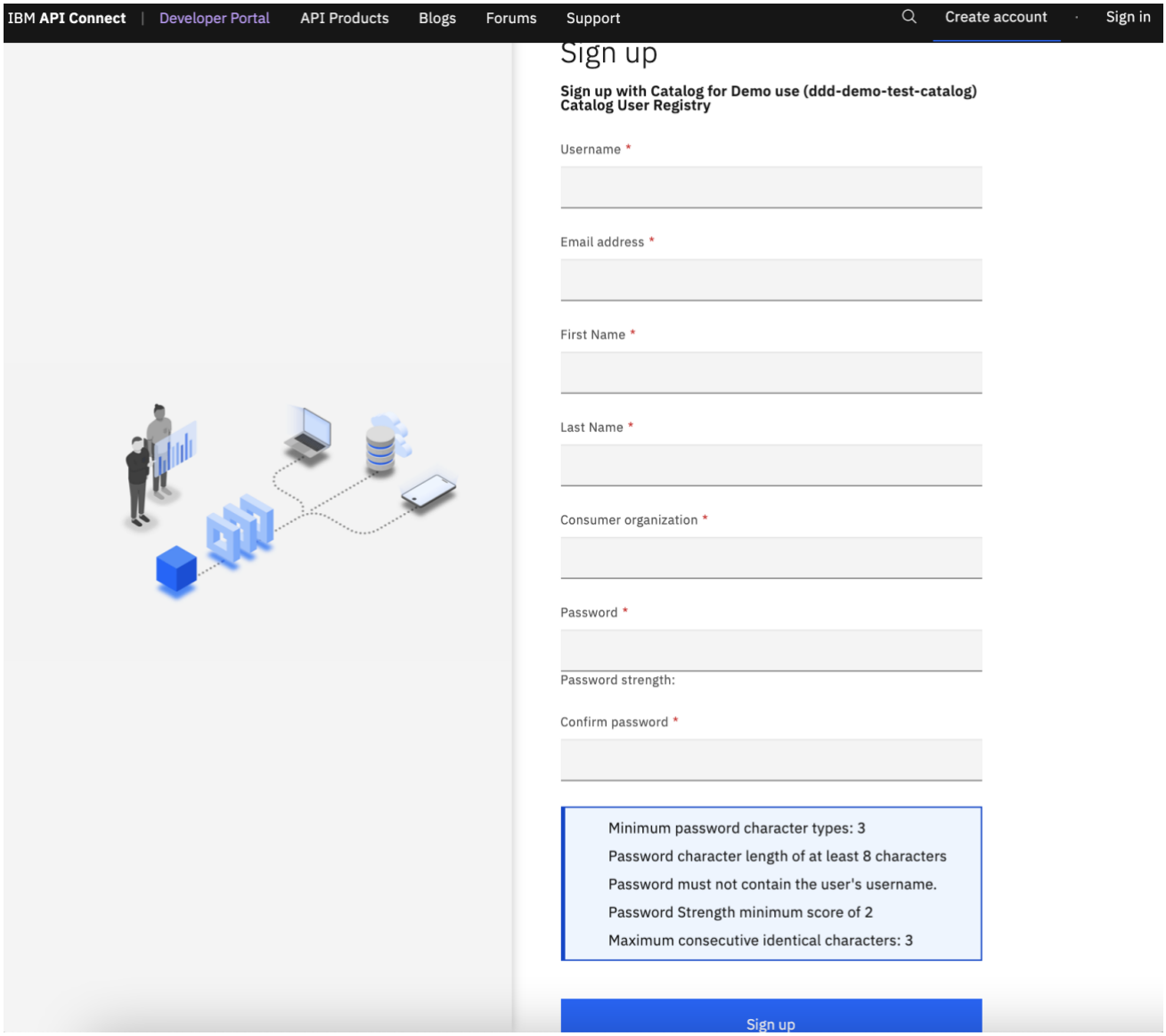
We're going to need to register a consumer and get an API key – luckily, we can do that self-service! Click 'Sign in'.



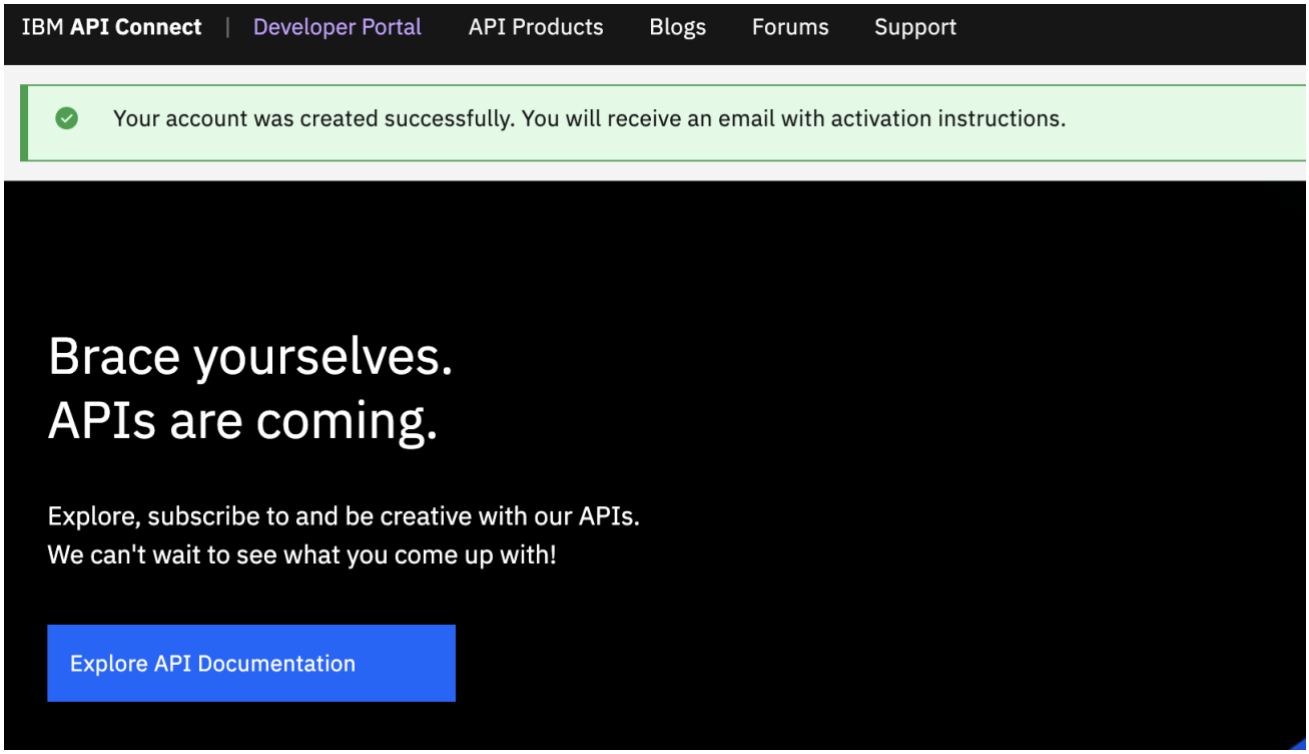

Click ‘Sign up’ to create a new account. (if you don’t have already one)




Fill in your details and then click ‘sign up’.

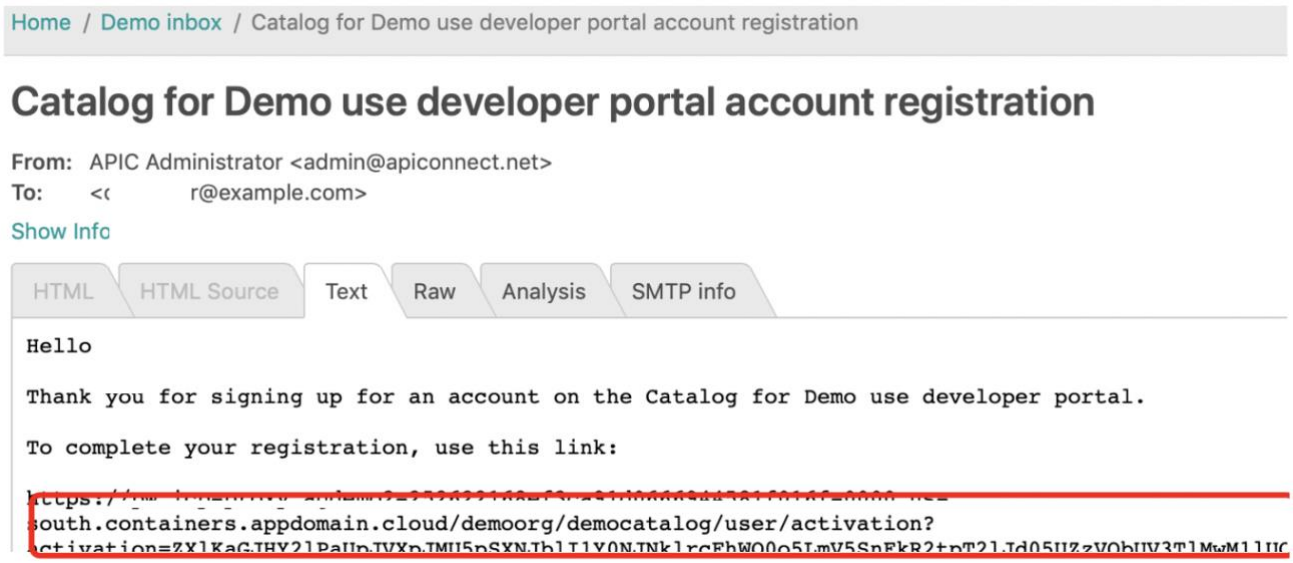



Tip: If you're using mailtrap.io as your mail server, you can use any email address. Use 'chatbot@example.com' to be safe – example.com is guaranteed to not be a real domain.

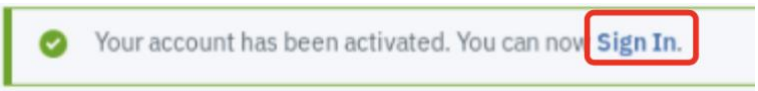



We'll need to get the email: You'll find it in your email page in your mailtrap.io. account.

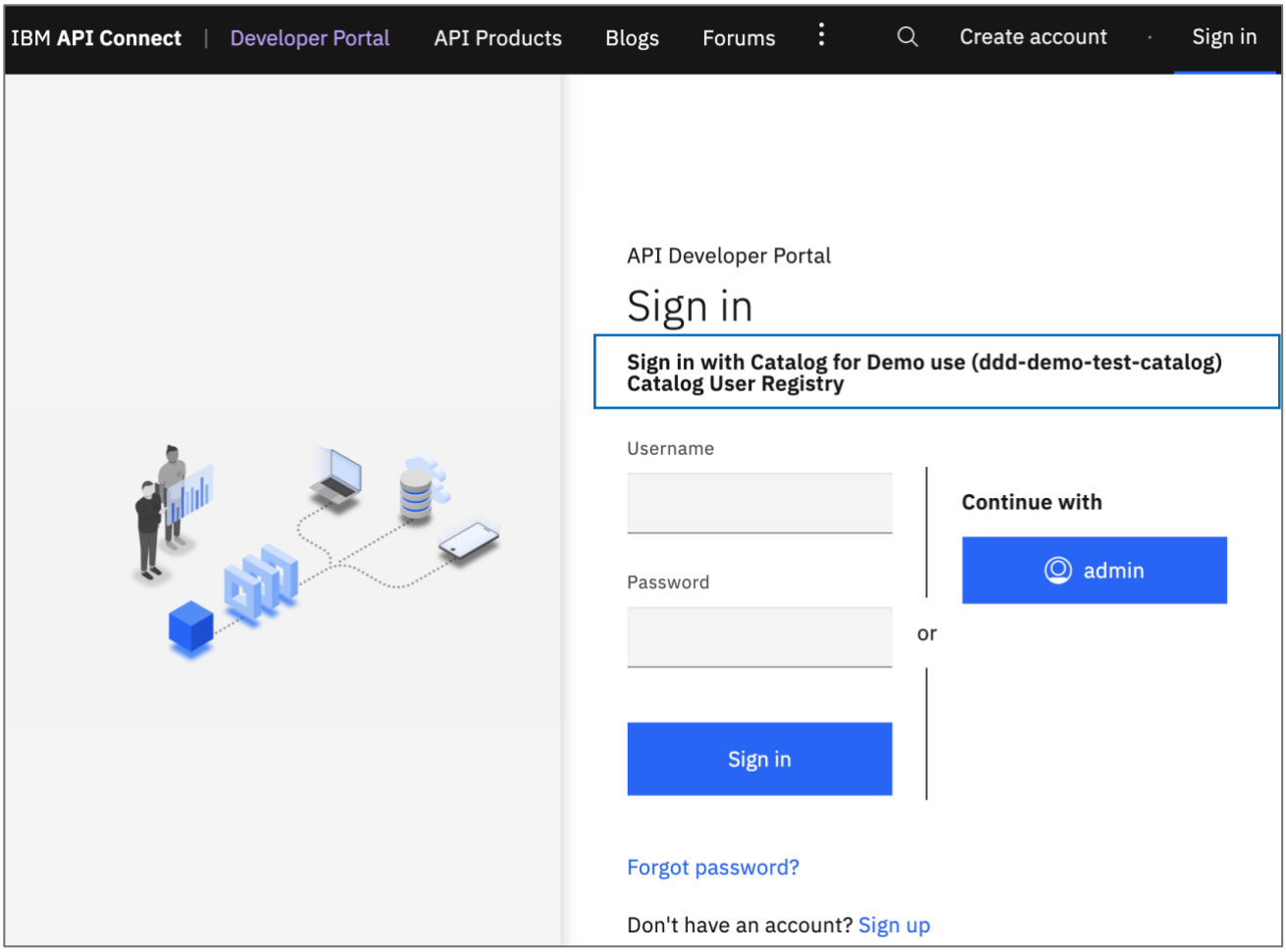
API Connect thinks you are now a new consumer user and has sent you an email to welcome you.



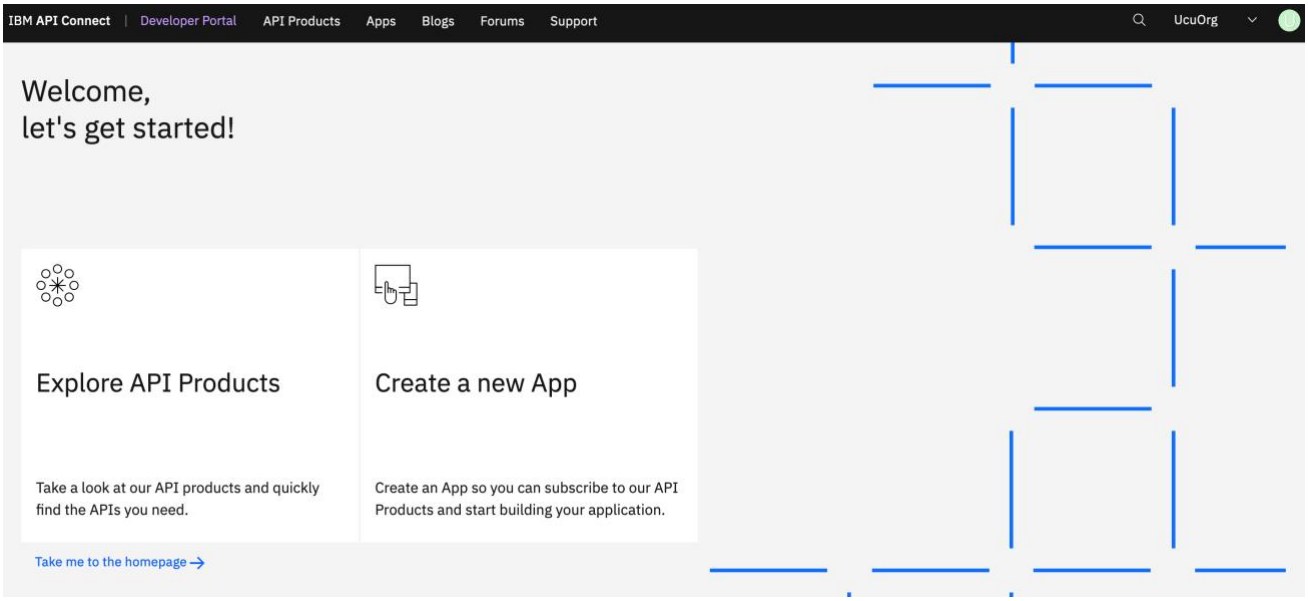

Copy and paste the link into the browser in the lab desktop machine. You should eventually get the portal with the notice:



Make sure you're using demo catalog user registry and sign in with your credentials you just created.



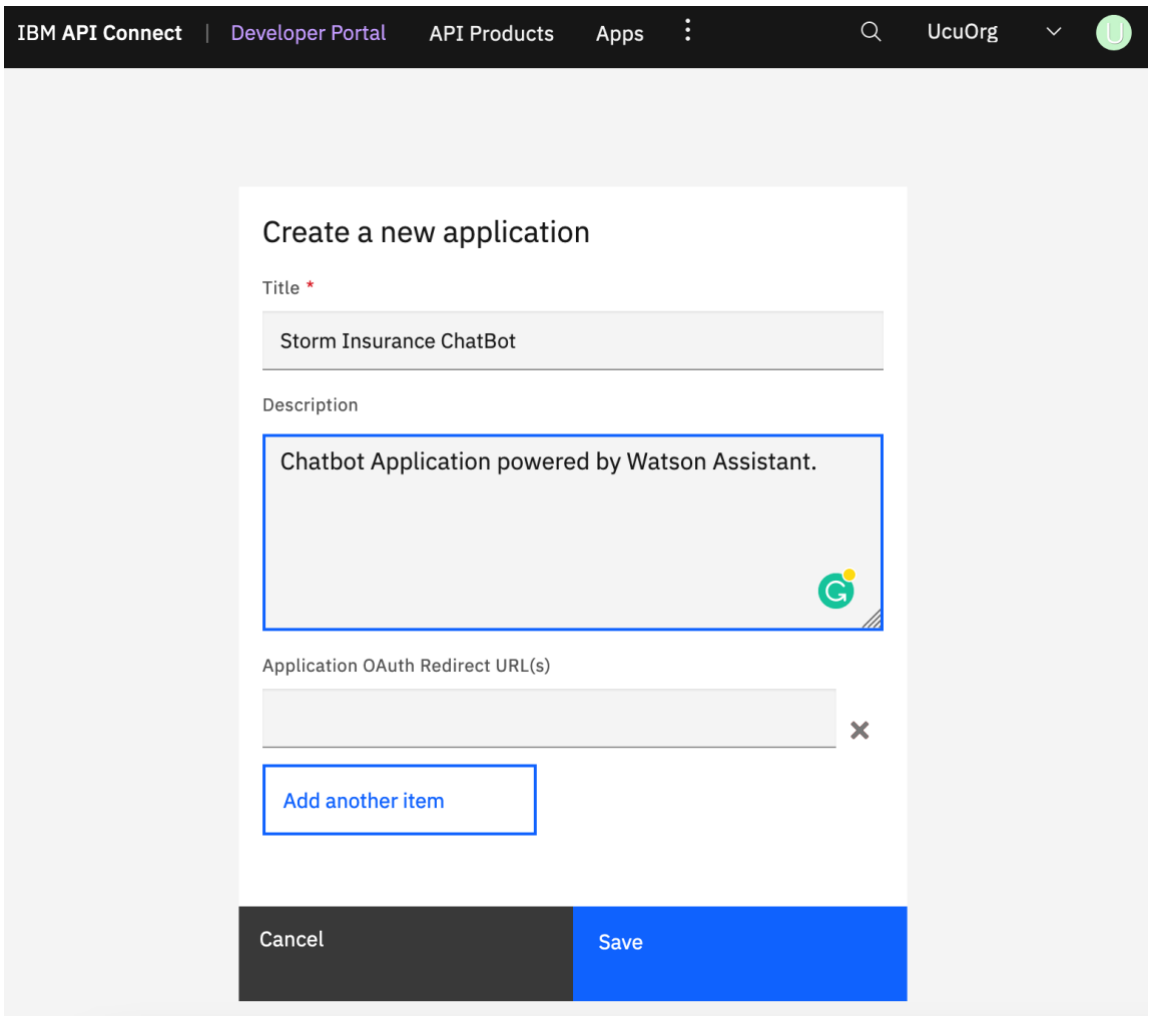

You'll get the following home screen:



We're going to create a new application: This will give us an API key so we can call our APIs.

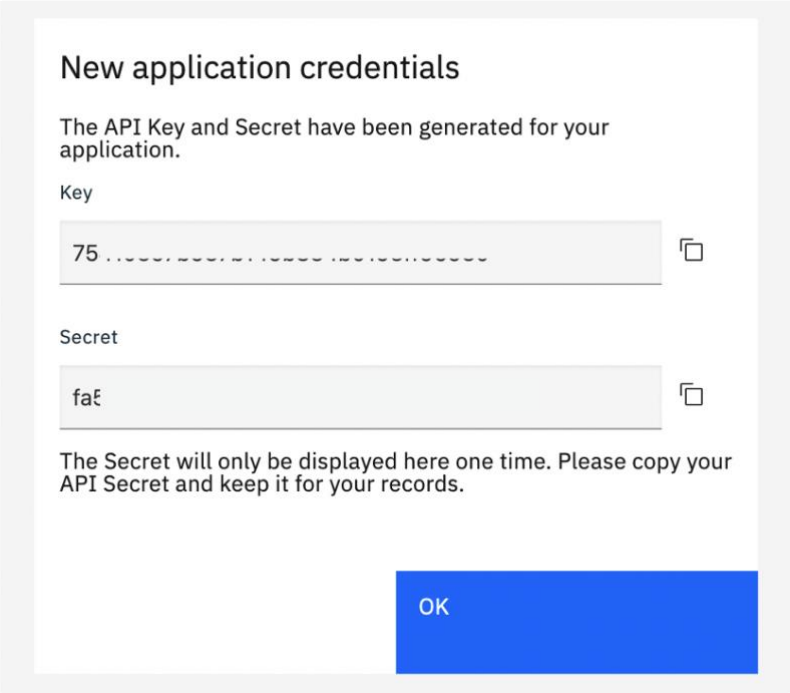
Click on 'Create a new App'.

Give our App a title e.g Storm Insurance Chatbot and click 'Submit':



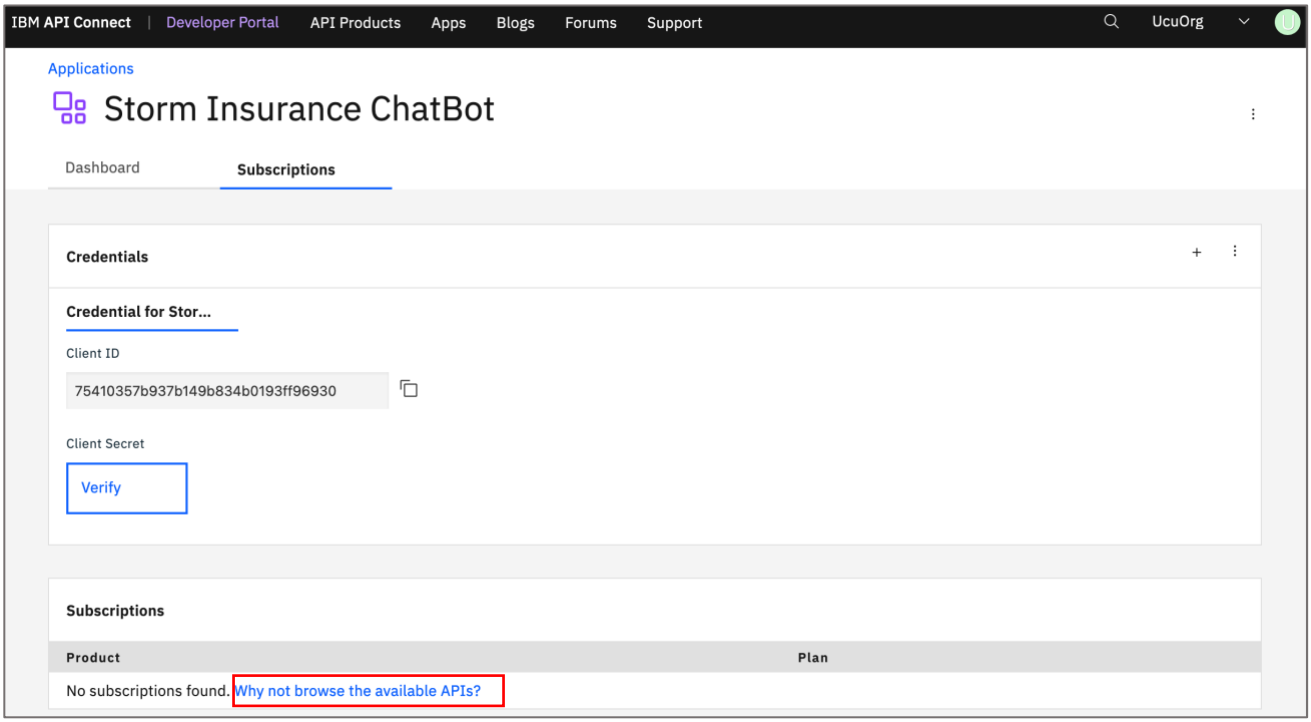

This gives us an API key and secret. You'll only ever be able to see the secret once here. For this lab, we haven't asked for secrets, so you won't need to remember it.

Click 'Copy' ( )to get your API key. Copy it somewhere safe then click 'Ok'

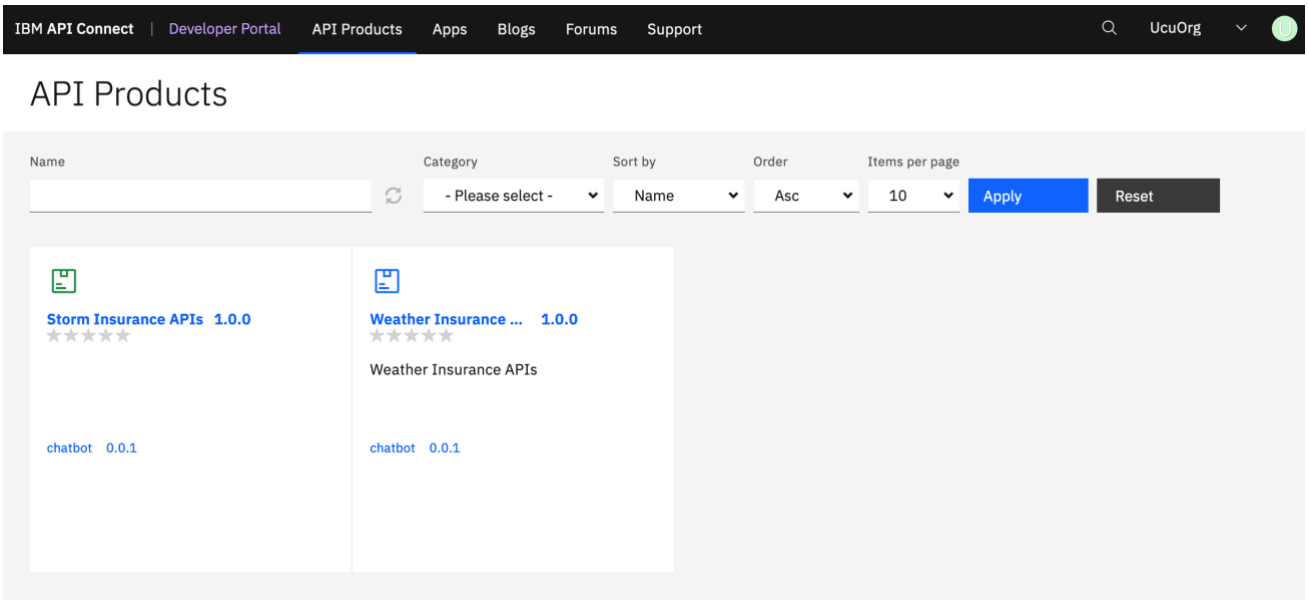



You'll now see the details for your application. Dashboard tab shows you the stats of your application. At the moment, we don't have an API calls, so no stats...

We've not also subscribed to any APIs yet – click on 'Why not browse the available APIs?'

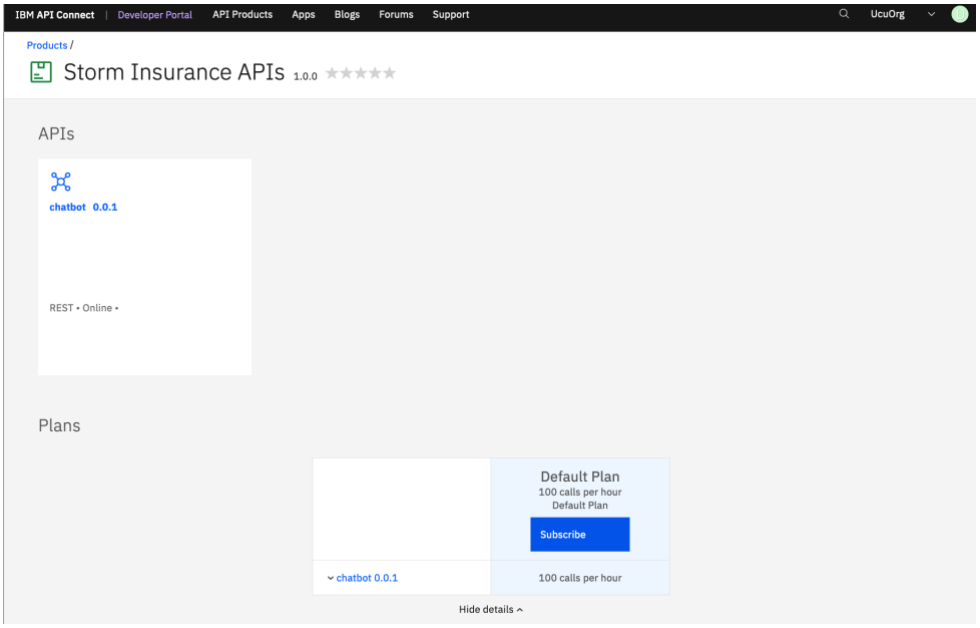



Click on the 'Storm Insurance APIs' product.

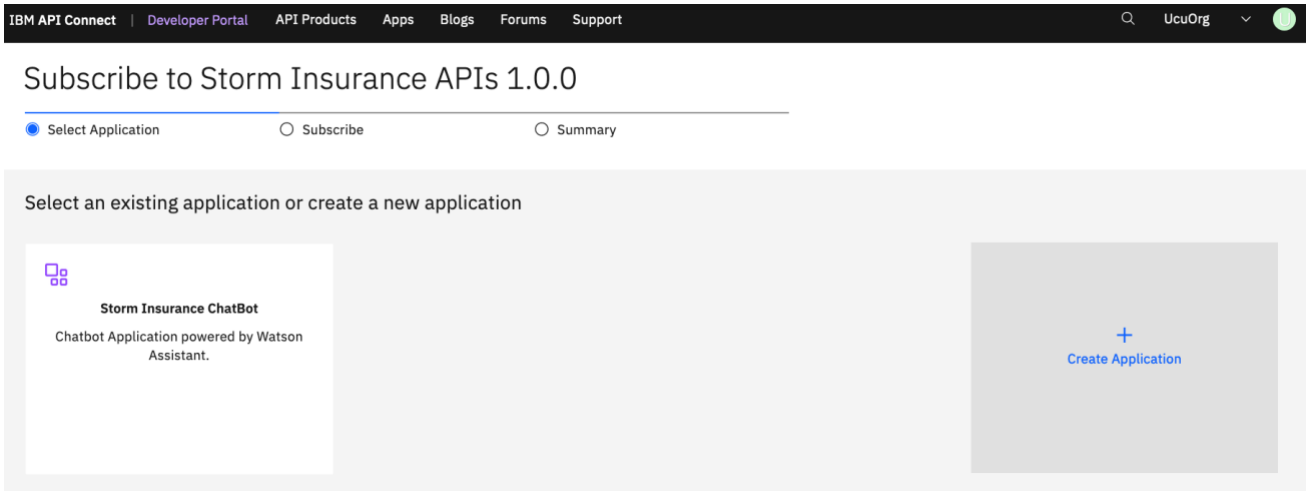



You can now see the plans:

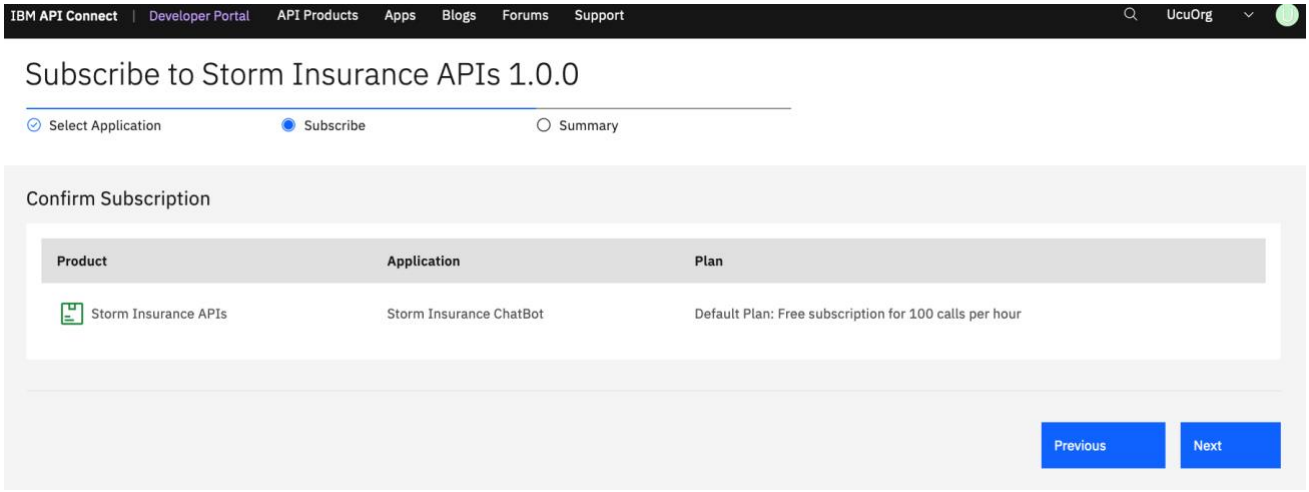
We have only Default plan. Click on ‘subscribe’.




Select the ‘Storm Insurance ChatBot’ application we have just created.



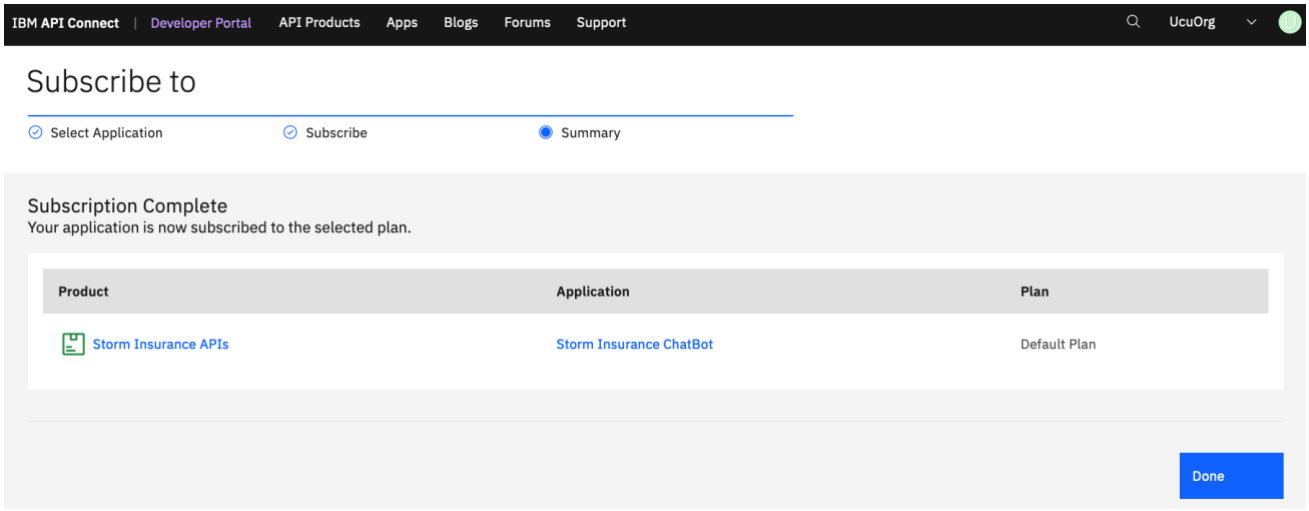

We now need to confirm our subscription – click ‘Next’.



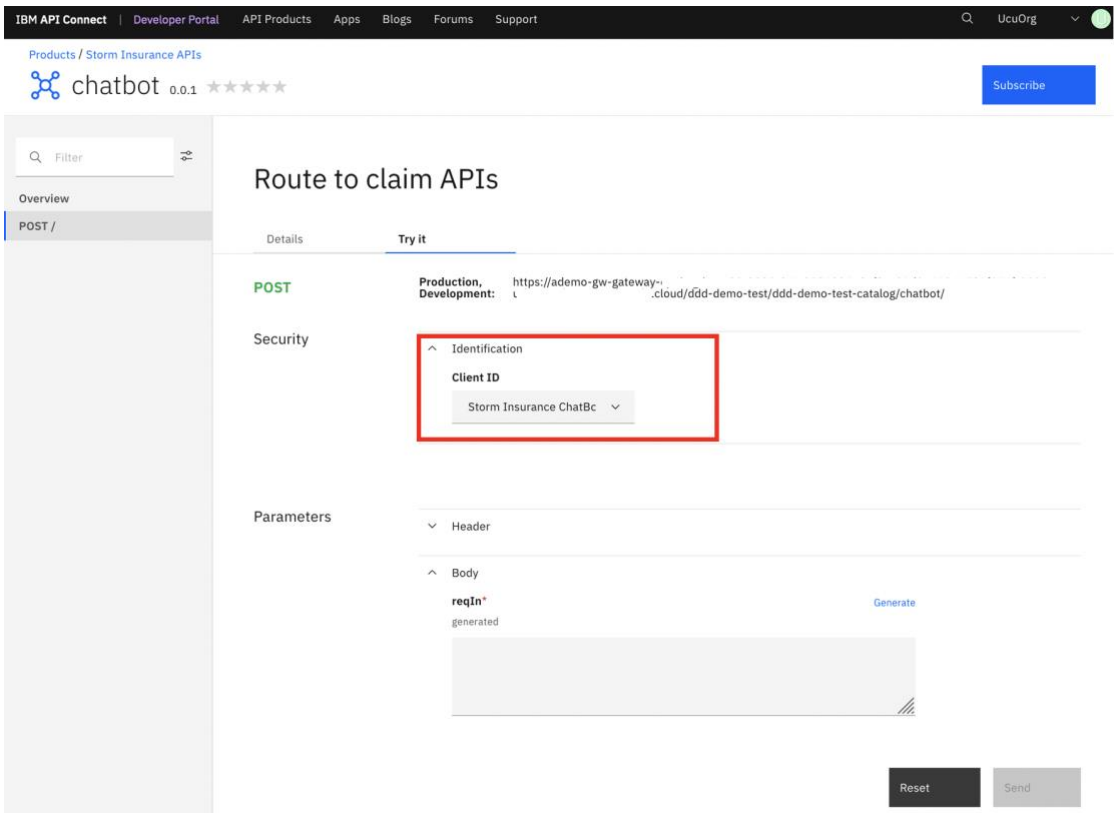

Now our application subscribed to the ‘Default plan’ of the ‘Storm Insurance APIs’ product.

What does this mean?

This means, ChatBot application can make 100 calls per hour for free.




We're now back at the product screen – click on the API itself, not the plan. Click on POST – note the portal has everything you need to call your API – if you scroll down, it's even generated clients in various languages for you (that's how we created our test clients in curl in our scripts for this lab).




You can go ahead and test your API.

Notice that API key will be automatically filled by the 'Storm Insurance Chatbot' application's client id.

You can use the below request examples. Remember to use the incident number returned from the first call in the subsequent calls.

**1) Route to ticketsstormincweather API**

```
{
  "apiName": "stormpath",
  "postCode": "70510:US",
  "date": "2018-07-13",
  "name": "John ACE"
}
```

**2) Route to classifyimages API**

```
{
   "apiName": "classifyImages",
   "image_url":" https://raw.githubusercontent.com/IBM/cp4i-demos/main/ace-weather-
chatbot/images/car.jpg",
   "incident_id": "REPLACE_INCIDENTNUMBER",
   "image_name":"My damaged car using a URL"
}
```

**3) Route to incidentsummary API**

```
{
  "incidentId": "REPLACE_INCIDENTNUMBER",
  "apiName": "summary"
}
```

Now we are ready to consume our API from Watson Assistant chatbot!

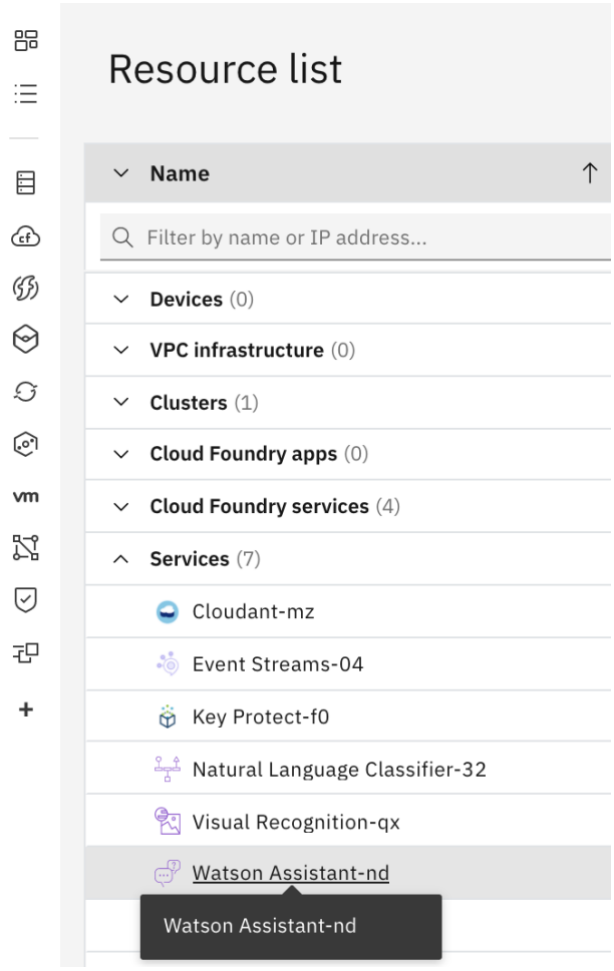
# 8 Create your Watson Assistant chatbot

Now it is time to create our chatbot which will consume the APIs we've created before.

Login to your cloud account. (sign in at https://cloud.ibm.com Use your IBM ID, or if you're an IBMer, you may go through a Single Sign-on process).

Once you're in, you'll be presented with the cloud dashboard showing which services you have provisioned.

From the 'Resource summary' tile click on the 'Services', you'll find your Watson Assistant service that you've created before.

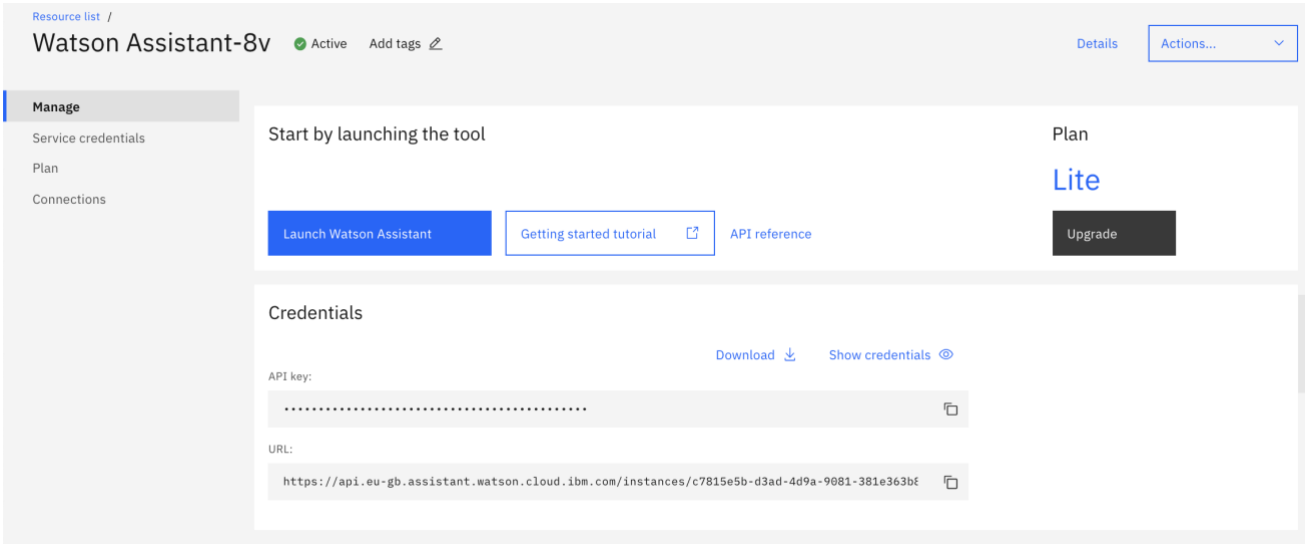



An assistant helps your customers complete tasks and get information faster. It may clarify requests, search for answers from a knowledge base, and can also direct your customer to a human if needed. In our demo,

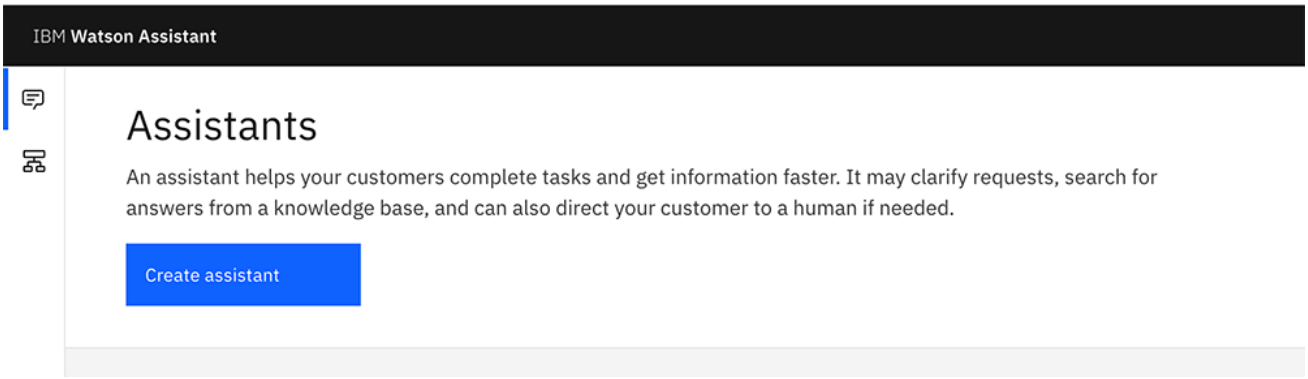
- it will search for answers to queries like 'can customer claim a storm insurance?' via validating the storm occurrence with IBM Weather Service.

- it will analyse the image uploaded by customer to get an estimate value for the damage via connecting to Watson Visual Recognition service.
- And moreover, it will automate the claim creation process.

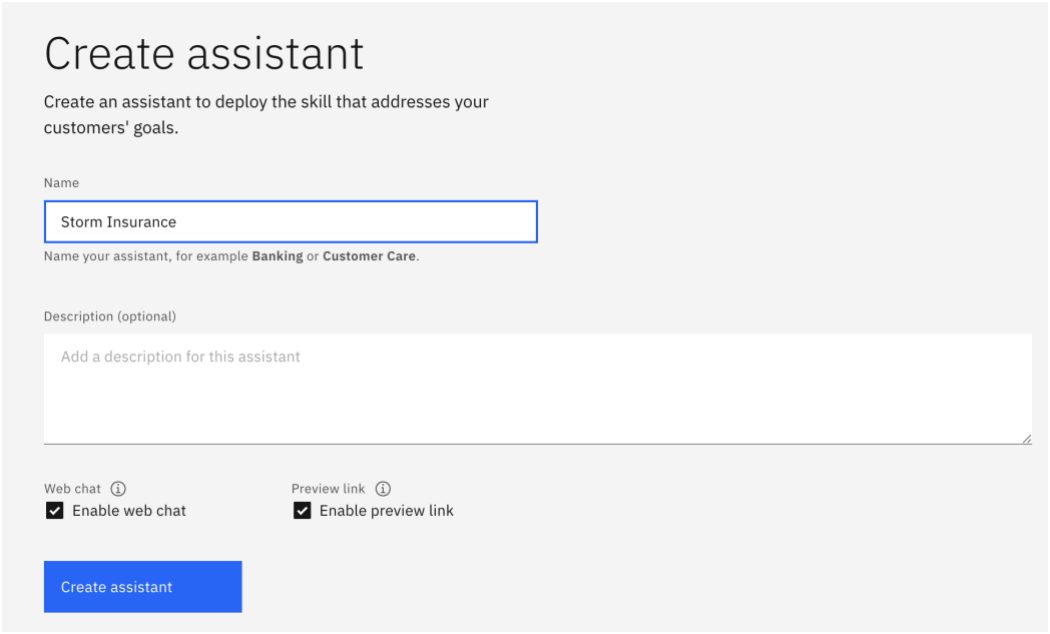
Click on your new service and you'll see the 'Manage' tab. Click 'Launch Watson Assistant'.




Click 'Create assistant'.




Name the Watson Assistance instance 'Storm Insurance' and click 'Create assistant'.

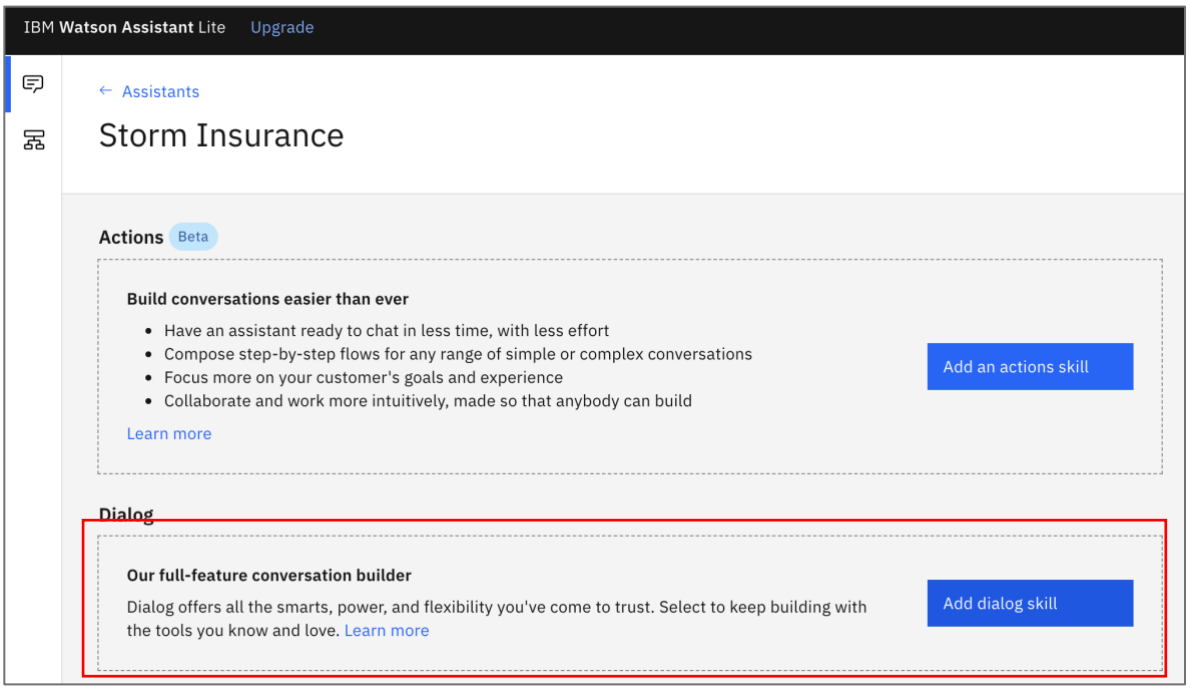



Next, we are going to add the dialog skill that is hosted on our Git repo (https://github.com/IBM/cp4i-demos/tree/main/ace-weather-chatbot/assets/watson-assistant-skills).

A dialog skill is a container for the artifacts that define the flow of a conversation that your assistant can have with your customers.

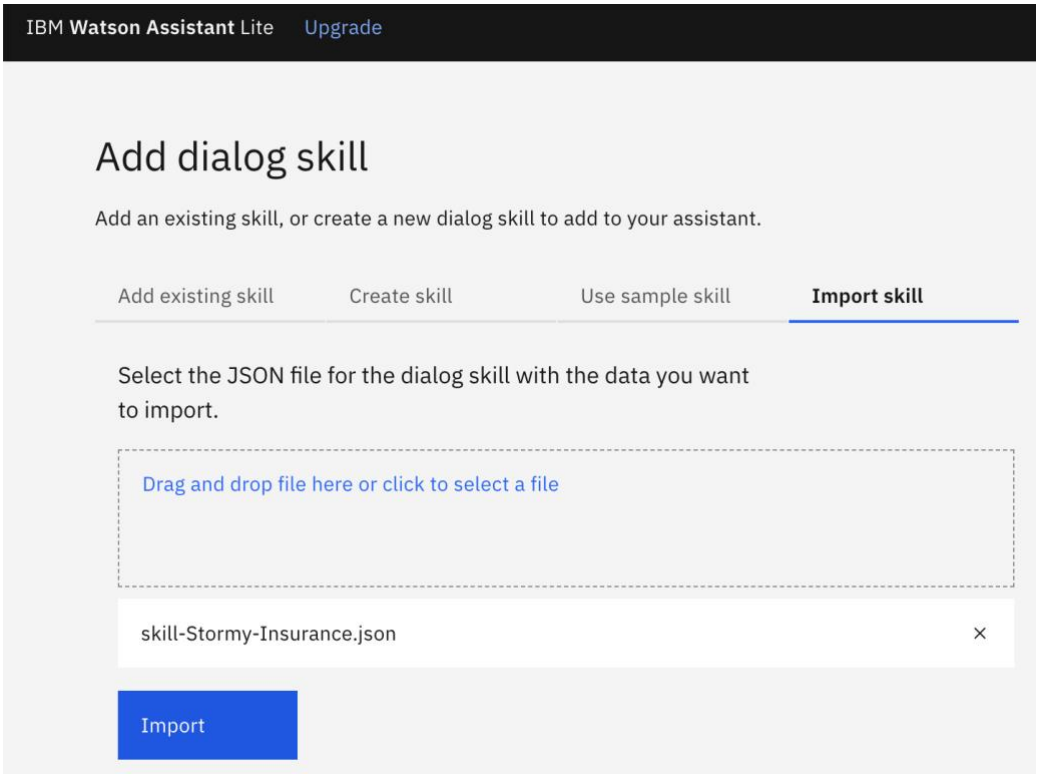
Download or clone the dialog skill (skill-Stormy-Insurance.json).

Click ‘Add dialog skill’.

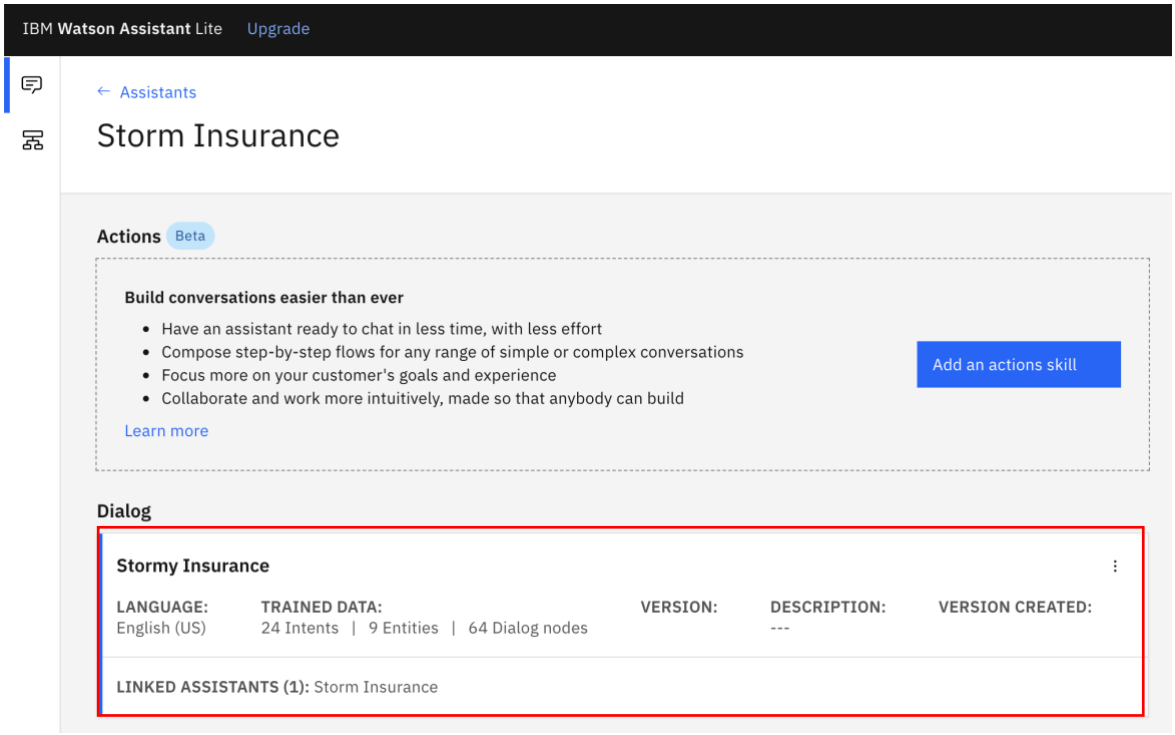


Click ‘Import skill’ tab.

Select ‘skill-Stormy-Insurance.json’ file that you’ve downloaded from Git repo and click ‘Import’




You will see that dialog is added to your assistant.

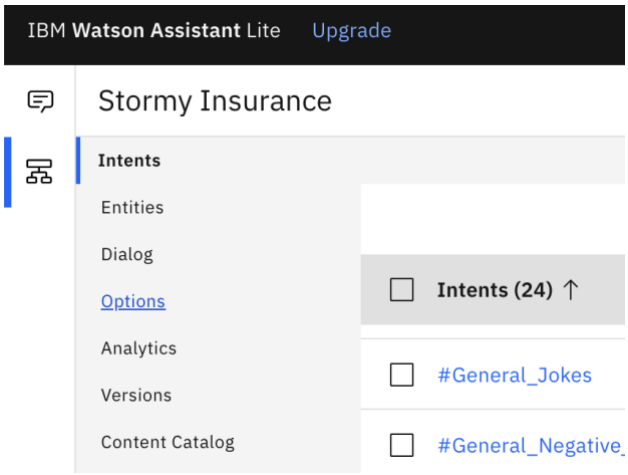



Now that you've created your Watson Assistant-enabled chatbot, you need to connect it to a data source. The following section shows you how to do that by adding webhooks to Watson Assistant that query for dynamic data.

Our insurance chatbot uses our facade API to communicate with the integration flows.

Click on your 'Storm Insurance' dialog tile to open the dialog

Click on Options on the left.

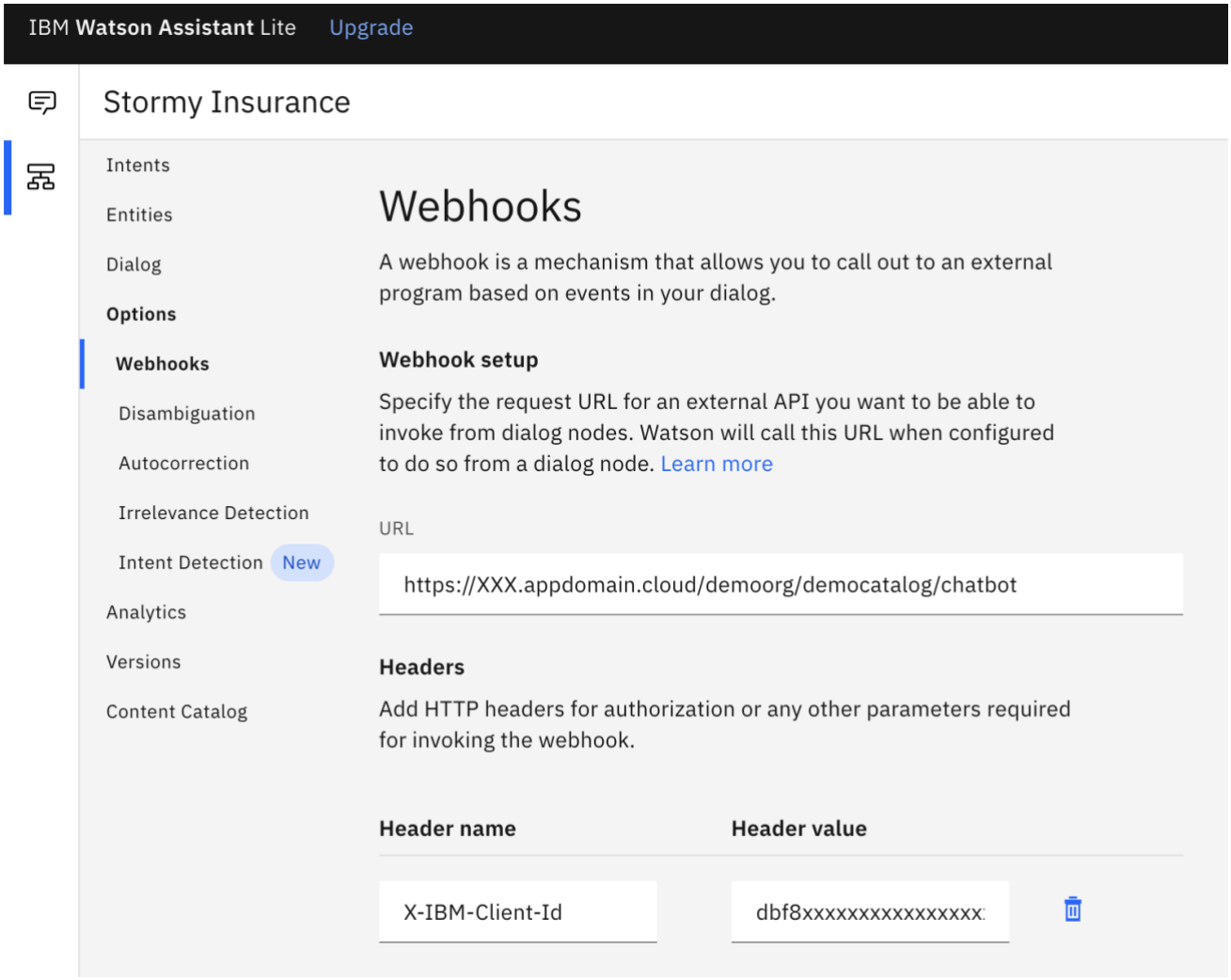



Under Options, click 'Webhooks'.

A webhook is a mechanism that allows you to call out to an external program based on something happening in your program. When used in a dialog skill, a webhook is triggered when the assistant processes a node that has a webhook enabled. The webhook collects data that you specify or that you collect from the user during the conversation and saves in context variables.

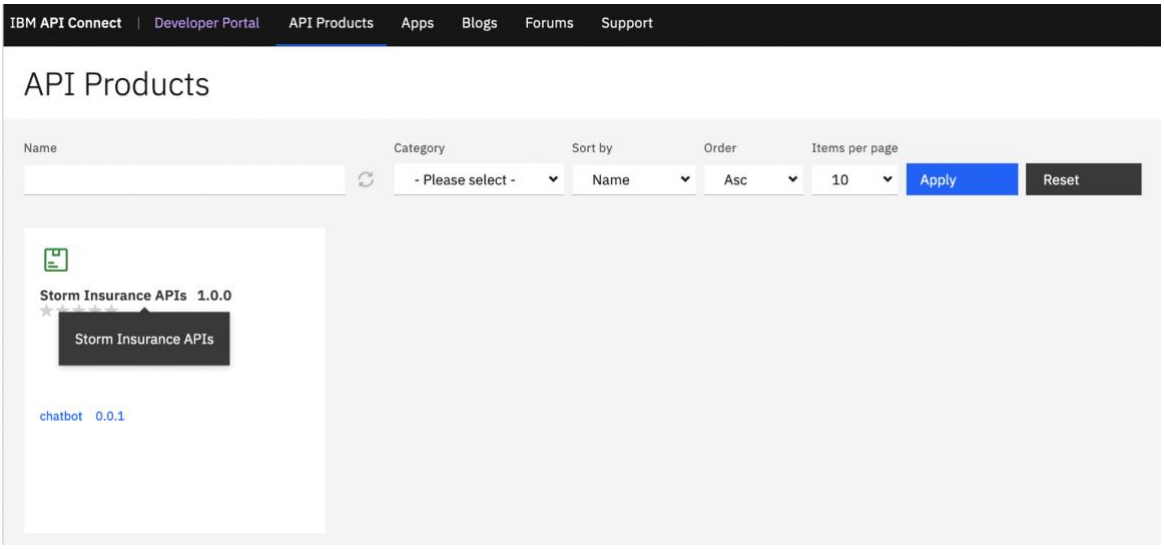
In the URL text box, type the endpoint value of your chatbot API (that you've exposed via API Connect)

In the 'X-IBM-Client-Id' Header value text box, type the Client ID (API key) value of your chatbot API (that you've exposed via API Connect).

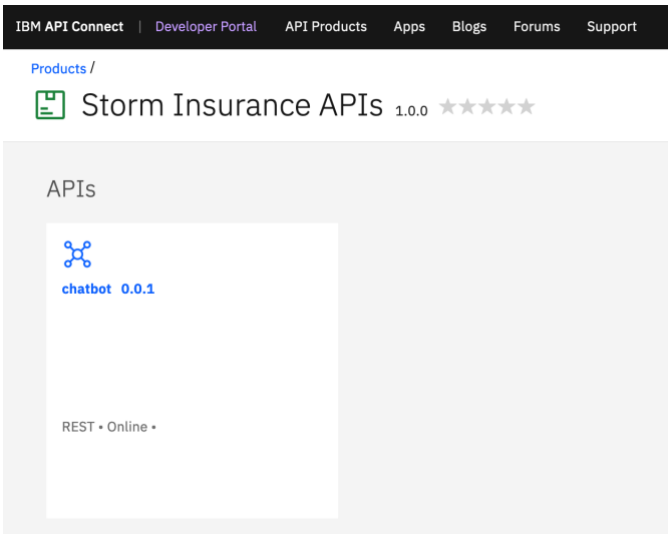



Note: If you didn't take a note of these values you can follow the steps below to access

From the API Connect Portal dashboard, click on the tile for the Storm Insurance APIs product.

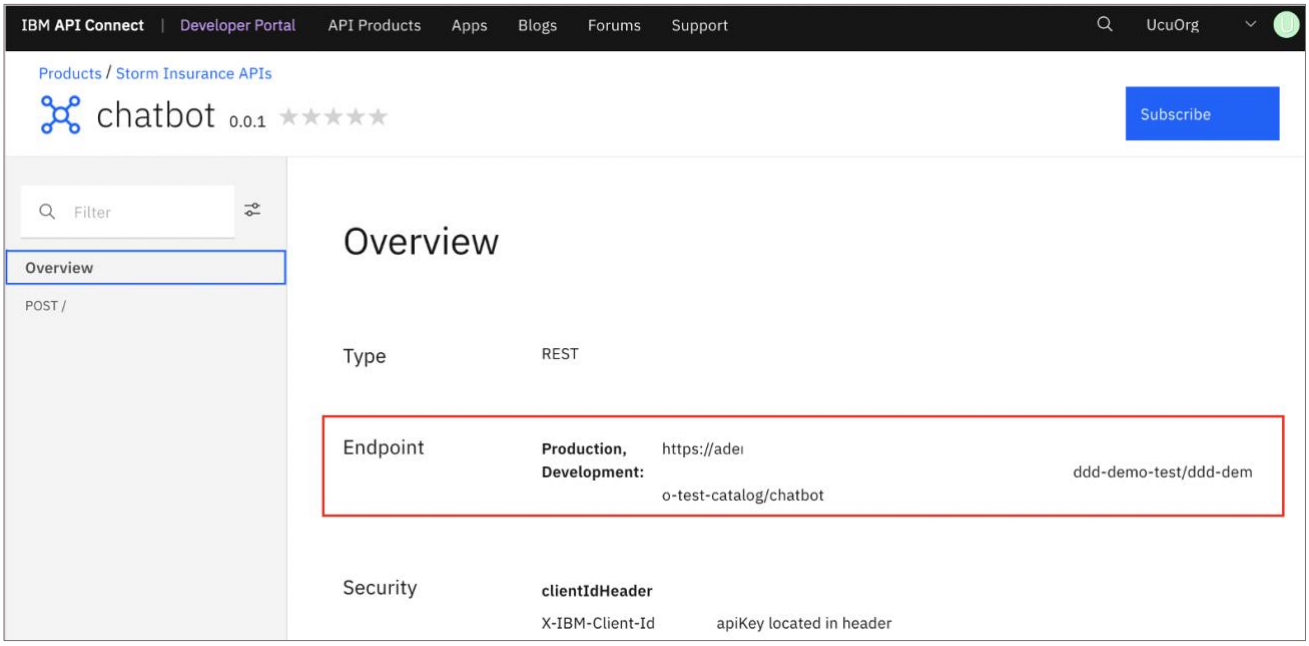



Click on the tile for the chatbot API.




From the 'Overview' copy the Endpoint URL

This is used as your webhook `URL`

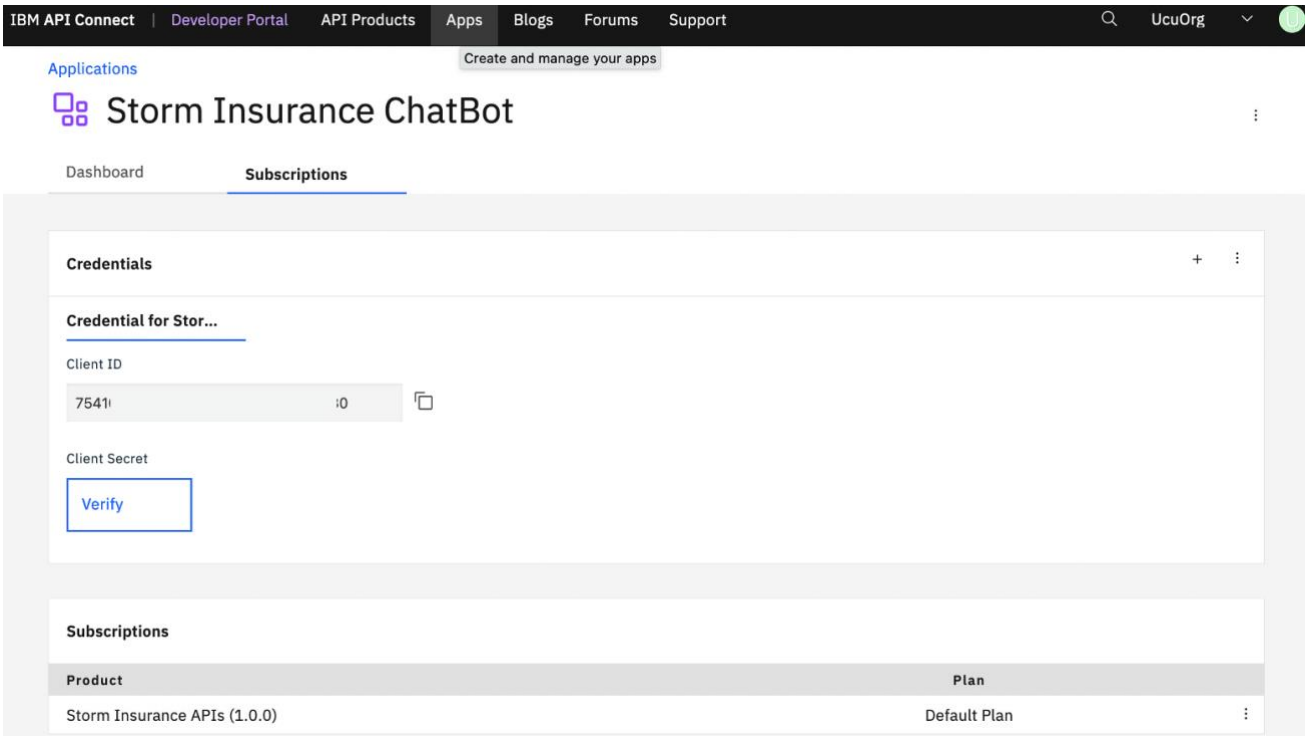



Next, we need to get the value to use as `'X-IBM-Client-Id'` HTTP Header value.

From the API Connect Portal dashboard, click 'Apps'.

Click on the tile for the 'Storm Insurance APIs' product and navigate 'Subscriptions' tab.

The 'Client ID' is the value to use as `'X-IBM-Client-Id'` HTTP Header value.



# 9 Node Chat Application

The application deals with text and image responses only from the Watson assistant. This project was bootstrapped with Create React App. This project deploys a react application that connects to a Watson API.

## 9.1 Installation

0. Clone this repo (https://github.com/IBM/cp4i-demos/tree/main/ace-weather-chatbot/assets/watson-assistant)

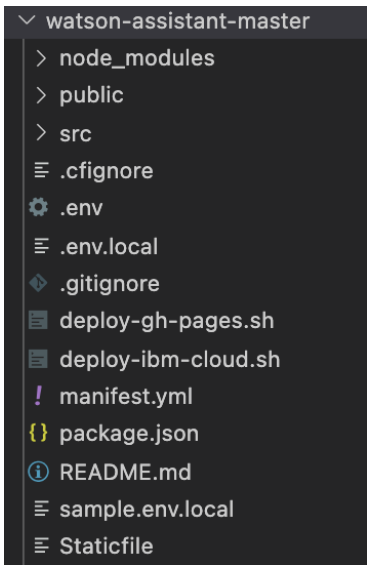
1. install 'yarn': follow the instructions for your OS

    a. Mac-specific: if you do not have it already, you will need to install xcode. This article gives detailed instructions.

2. Install the dependencies: yarn install

## 9.2 Configuration

Once you cloned the repository and installed the yarn packages, you will see a similar folder structure.




You can now configure the project with your preferences and demo environment details.

### 9.2.1 Set access credentials in `.env.local`

The chatbot has the following immediate dependencies;

- The API Credentials specific to your Watson Assistant
- The credentials to access an image processing API, which is provided by an App Connect API Flow

The credentials for these are stored in the `.env.local` file, which you will need to create, in the same directory that you cloned this repo to. We have supplied a sample `sample.env.local` that you should edit and rename to `.env.local`.

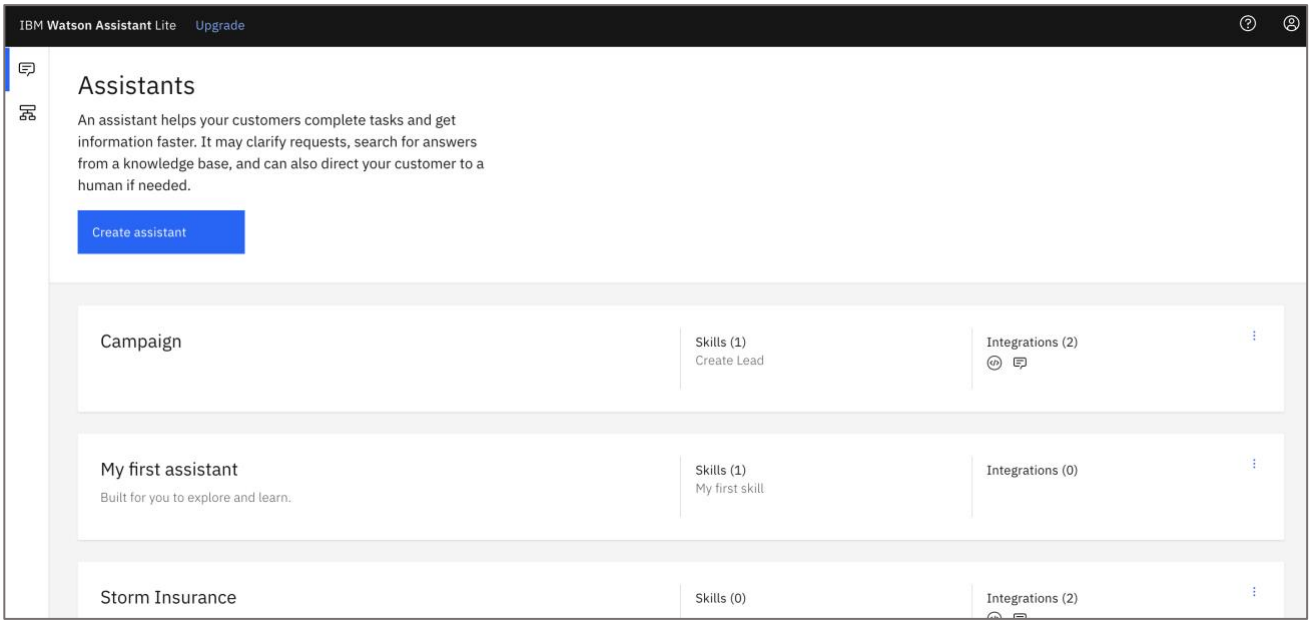
```
REACT_APP_ASSISTANT_URL="https://..."
REACT_APP_ASSISTANT_API_KEY="xxxYYYzzz"
REACT_APP_IMAGE_PROCESS_URL="https://..."
REACT_APP_IMAGE_PROCESS_API_KEY="xxxYYYzzz"
```

**For your Watson assistant**

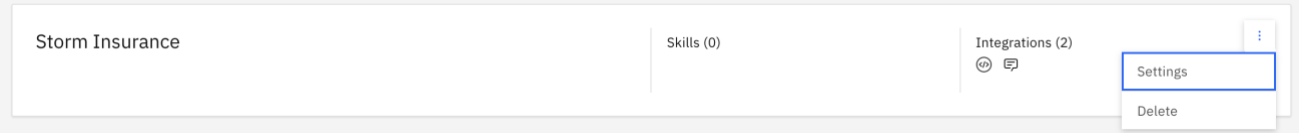
- The API URL for your Watson Assistant: `REACT_APP_ASSISTANT_URL`
- The API key for accessing your Watson Assistant instance: `REACT_APP_ASSISTANT_API_KEY`

To get these;

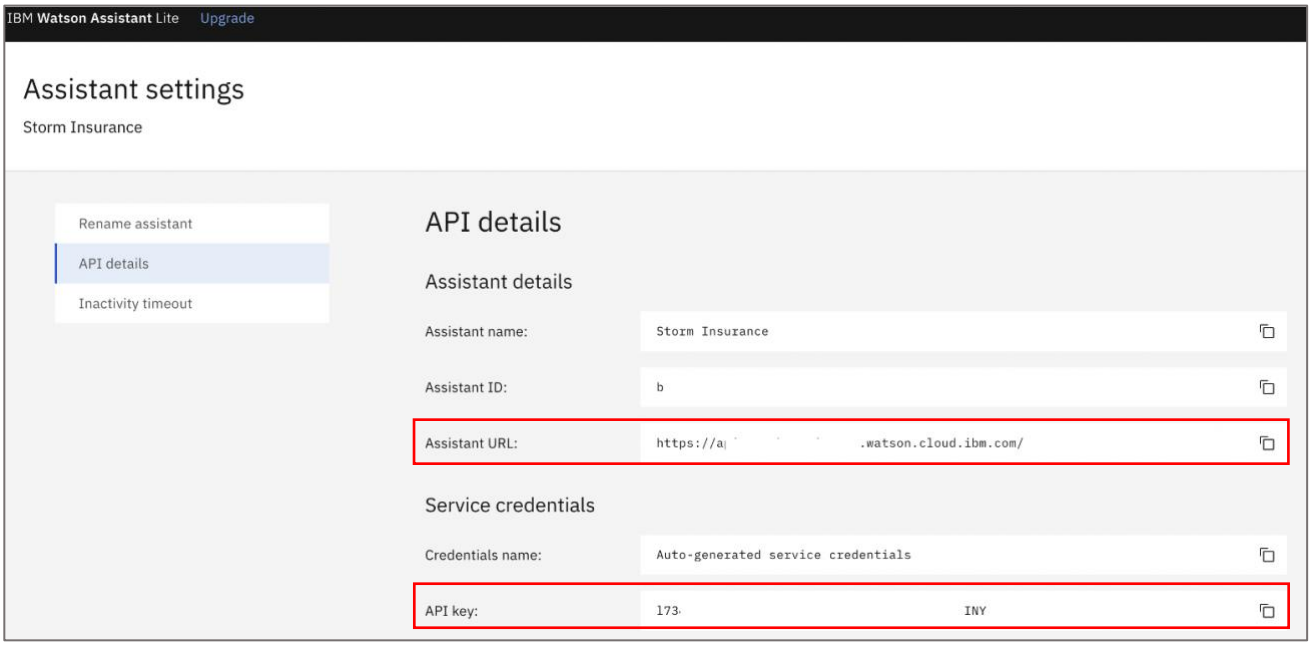
Browse to your list of assistants on assistant.watson.cloud.ibm.com




Choose the assistant for this demo; from its overflow menu (:) select ‘Settings’.




Select the 'API details' tab, where you will find "Assistant URL" and "API Key".

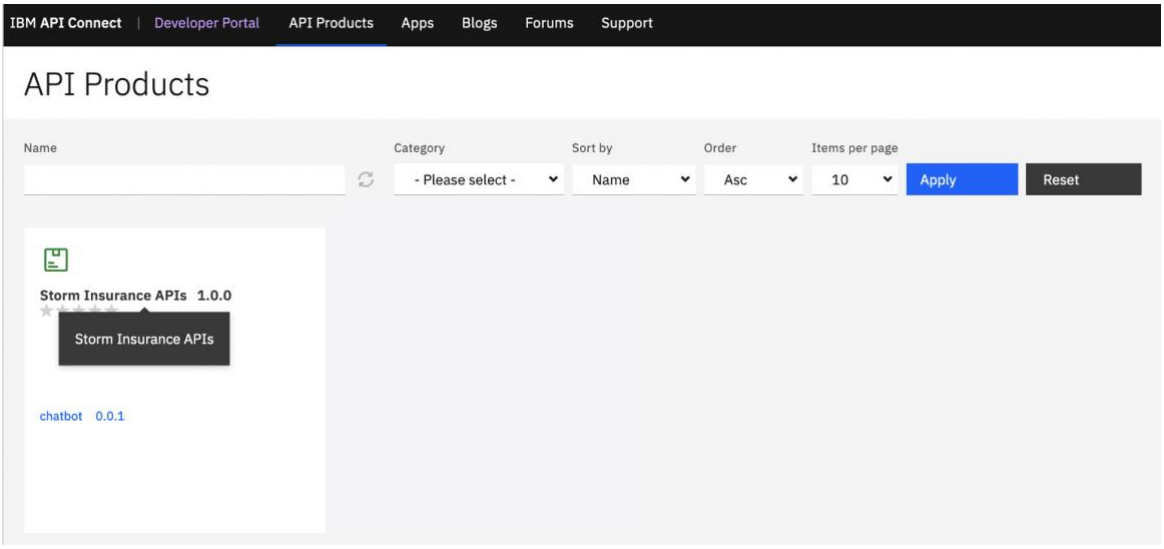



**For your App Connect Image Processing flow**

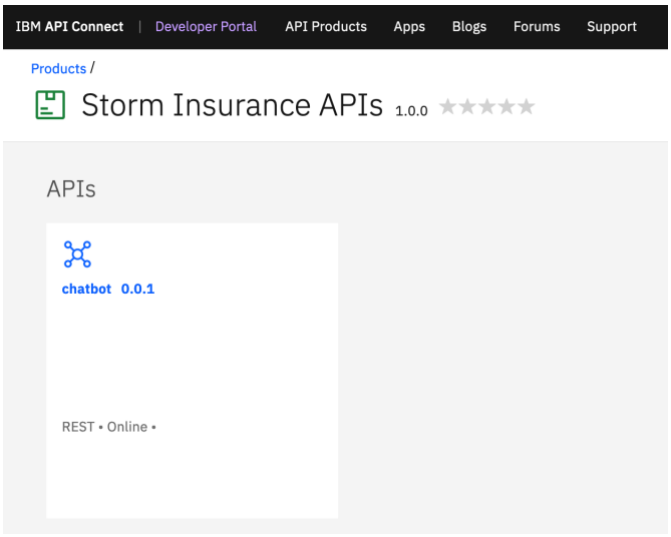
- The URL for your image processing API: `REACT_APP_IMAGE_PROCESS_URL`
- The API Key for accessing your image processing API: `REACT_APP_IMAGE_PROCESS_API_KEY`

To get these;

From the API Connect Portal dashboard, click on the tile for the Storm Insurance APIs product.

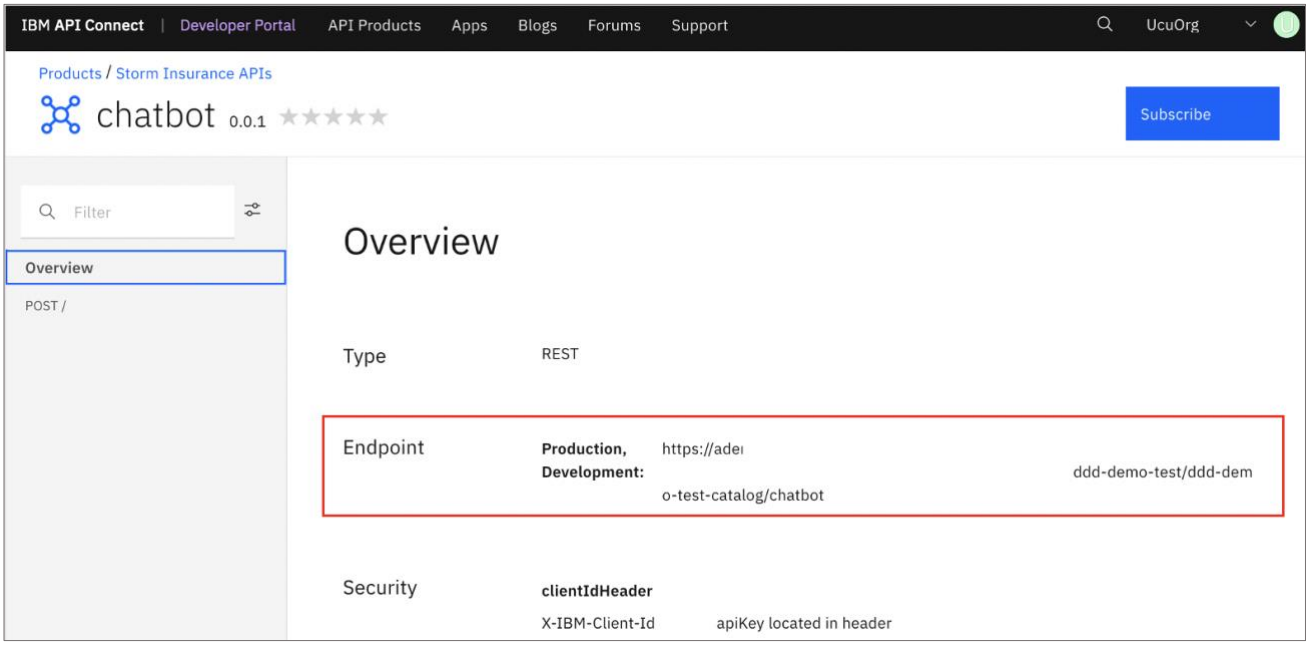



Click on the tile for the chatbot API.




From the 'Overview' copy the Endpoint URL

This is used as your `REACT_APP_IMAGE_PROCESS_URL`

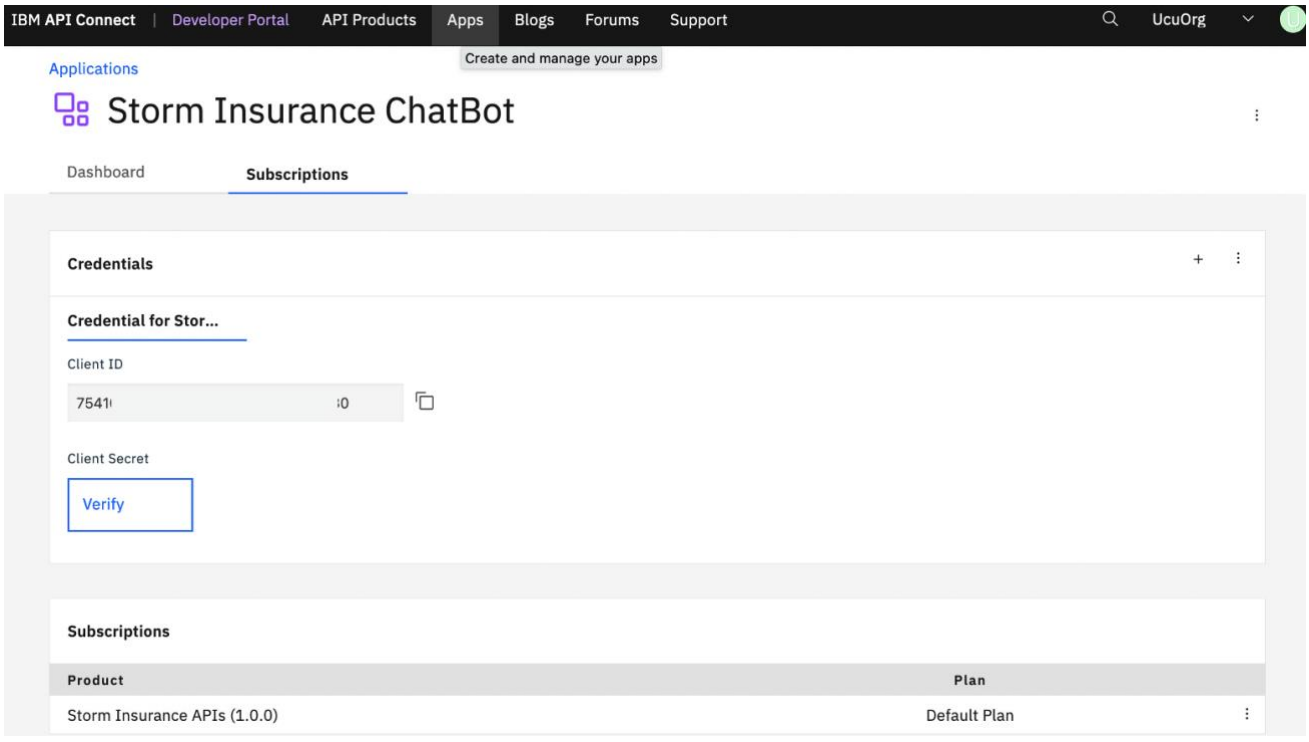



Next, we need to get the value to use as `REACT_APP_IMAGE_PROCESS_API_KEY`.

From the API Connect Portal dashboard, click ‘Apps’.

Click on the tile for the ‘Storm Insurance APIs’ product and navigate ‘Subscriptions’ tab.

The ‘Client ID’ is the value to use as `REACT_APP_IMAGE_PROCESS_API_KEY`



### 9.2.2 Set the value of proxy in `package.json`

You must set the value of proxy in `package.json`. You need to do this to allow your client to communicate with Watson assistant - which does not allow 'Cross Origin' requests. This must be set to the same value as the base part of your `REACT_APP_ASSISTANT_URL`. For example, if your `.env.local` has;

REACT_APP_ASSISTANT_URL="https://api.us-south.assistant.watson.cloud.ibm.com/instances/ffcc1122/..."

then your package.json should contain;

```
  ...
  "proxy": "https://api.us-south.assistant.watson.cloud.ibm.com",
  ...
```

### 9.2.3 File uploads

The `Assistant.js` component is currently configured to accept jpg, png and gif. Change the following line to update that.

```
const imageTypes = ["jpg", "jpeg", "png", "gif"];
```

### 9.2.4 File processing code

The file is processed via code in src/utils/imageApiCall.js. The supplied implementation calls an API, and constructs a Watson Assistant message based on the response.

This code is called from within src/components/Assistant.js using this code;

```
imageApiCall(result.value.data).then(
  (response) => {
    //addUserMessage(response)
    sendUserMessage(response)
      .then((res) => {
        console.log(JSON.stringify(res.data, null, 2))
        setConversation((prevState) => [...prevState, res.data]);
      })
      .catch((err) => {
        console.dir(err);
      });
  }
)
```

The current implementation sends the result of 'imageApiCall' to Watson Assistant (sendUserMessage) but does not display is in the chat dialog. The message can be displayed in the dialog by uncommenting the call to 'addUserMessage'.

### 9.2.5 The dialog

The dialog adds both images the user adds and a text message depending on the number of files uploaded.

- "file: uploads failed"
- "file: upload failed"
- "file: uploads successful"
- "file: upload successful"

Comment out the 'addUserMessage' that follows these strings in Assistant.js to avoid displaying this string.

```
addUserMessage(msg);
```

### 9.2.6 Assistant response to files

The same text messages are sent to the Waston Assistant using 'sendUserMessage(msg)'.

- "file: uploads failed"
- "file: upload failed"
- "file: uploads successful"
- "file: upload successful"

These need to be configured as intents in the Watson assistant with responses in the dialog.

## 9.3 Running your Node Chat Application

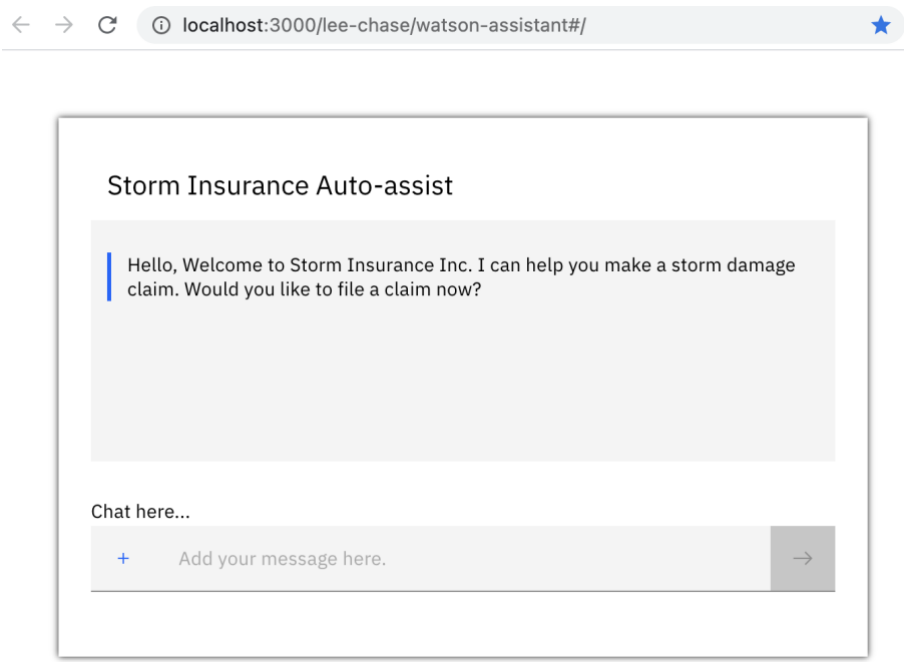
We have installed and configured our application. Now we are ready to run it locally.

From your terminal navigate to your project folder and run the following command.

```
yarn start
```

Runs the app in the development mode.
Open http://localhost:3000 to view it in the browser.

The page will reload if you make edits. You will also see any lint errors in the console.




There are some other scripts in the project directory that you can run to test, build and eject the application

```
yarn test
```

Launches the test runner in the interactive watch mode.
See the section about running tests for more information.

```
yarn build
```

Builds the app for production to the build folder.
It correctly bundles React in production mode and optimizes the build for the best performance.

The build is minified and the filenames include the hashes.
Your app is ready to be deployed!

See the section about deployment for more information.

```
yarn eject
```

*Note: this is a one-way operation. Once you eject, you can't go back!*

If you aren't satisfied with the build tool and configuration choices, you can 'eject' at any time. This command will remove the single build dependency from your project.

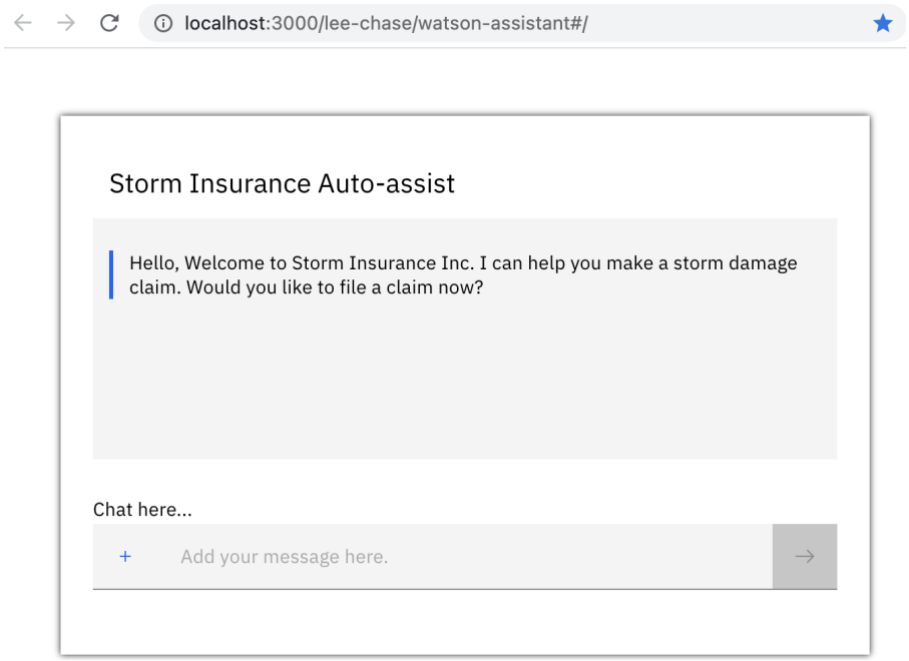
Instead, it will copy all the configuration files and the transitive dependencies (webpack, Babel, ESLint, etc) right into your project so you have full control over them. All of the commands except eject will still work, but they will point to the copied scripts so you can tweak them. At this point you're on your own.

You don’t have to ever use ‘eject’. The curated feature set is suitable for small and middle deployments, and you shouldn’t feel obligated to use this feature. However we understand that this tool wouldn’t be useful if you couldn’t customize it when you are ready for it.

## 9.4 Test everything – your cognitive integrated insurance chatbot!

Let’s test our application end to end.

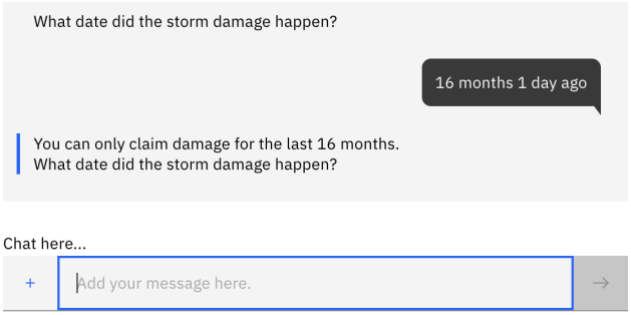
Open http://localhost:3000 to view chatbot in the browser.



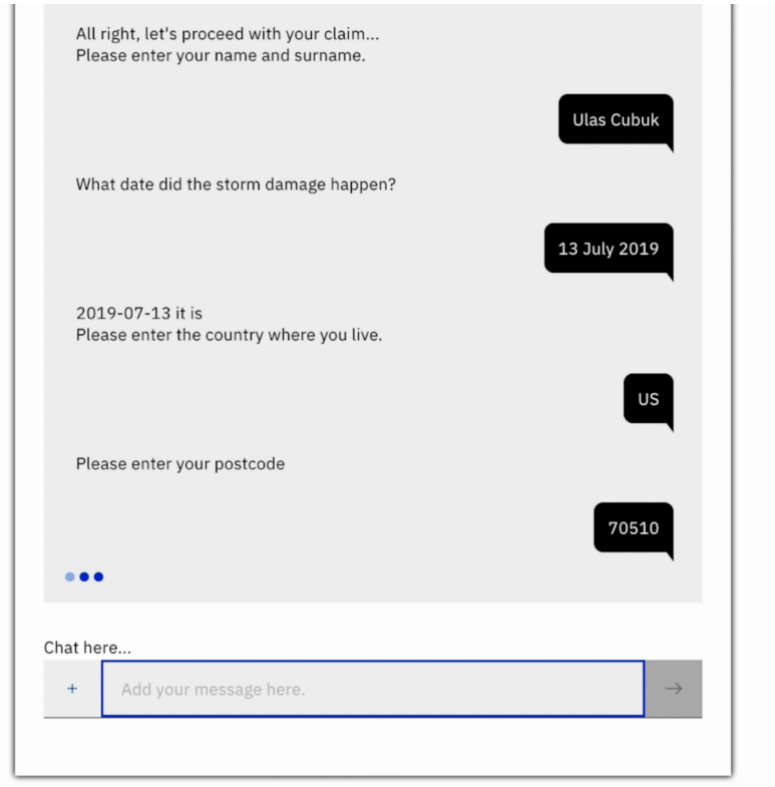

Type ‘yes’ to start your claim process.

Next the chatbot ask questions to collect details about you, your location and the date of the storm.

Please note that for the purpose of this demo we trained the chatbot to accept only US and UK postcodes. Also, we’ve embedded a logic to accept the claims requests that are not older than 16 months. In that case, it will ask you again to enter a valid date or you can refresh your page to start over again.



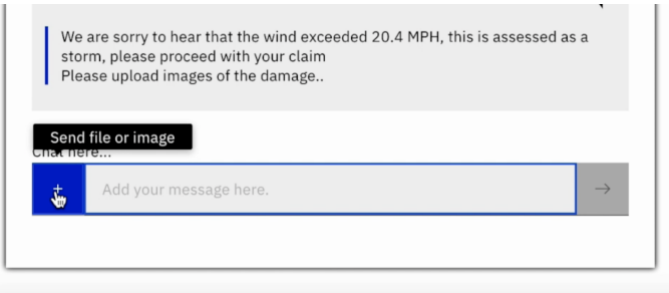

The location and the date given below in the screenshot is an occurrence that matches the criteria of storm in our integration flow. You can use the same or you can try other options.



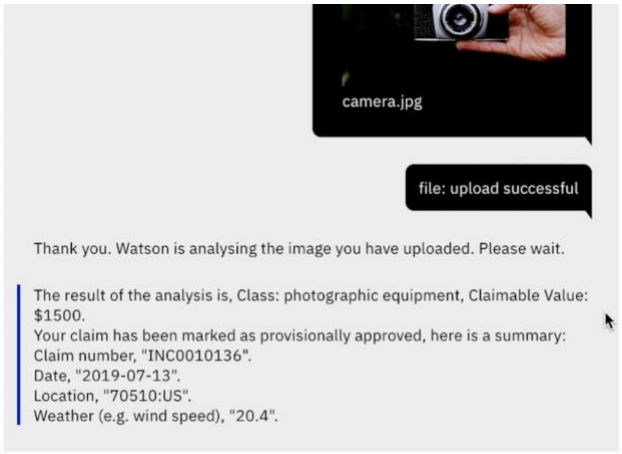

At this point the chatbot uses the power of integration to return back a dynamically retrieved assessment result. In addition to that it will trigger claim process and create a provisional claim.

You can see that your case is confirmed as a storm occurrence.

Now all you need to do is to upload the image of the damage.




Again, your chatbot, powered with integration and Watson Visual Recognition, will analyse the damaged object.



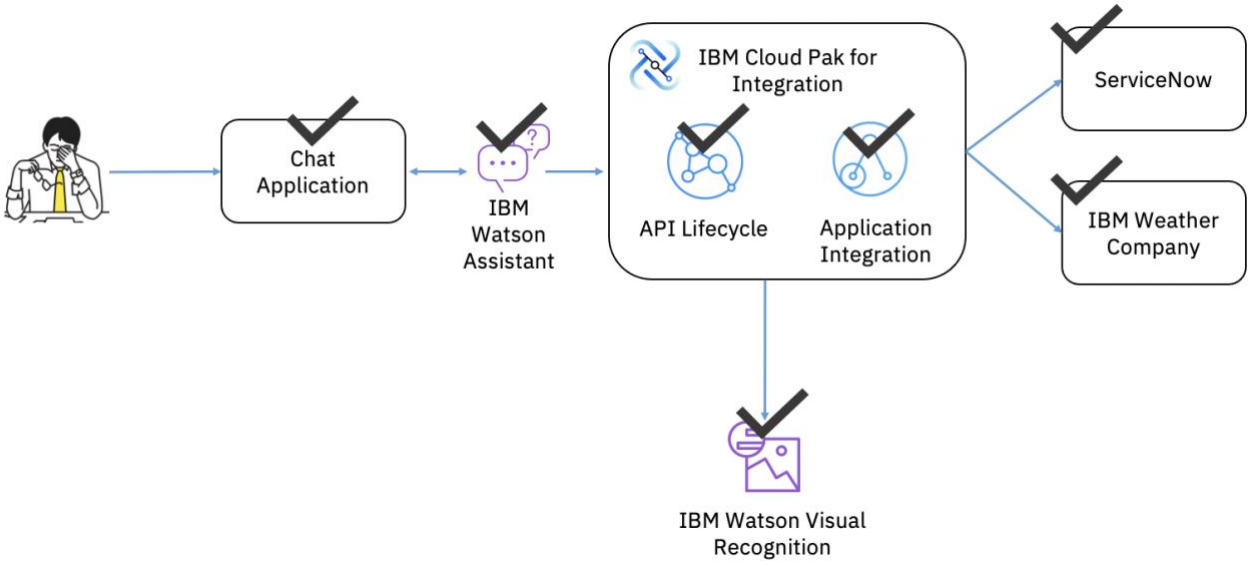

IBM Watson Assistant replies to the user the analyses result along with the provisional claim details. The chat application displays the answer to the user.

There you go! No need to worry about the issues like; how to make an insurance claim? what do I need to submit for an insurance claim? who will assess the damage? when should I make an insurance claim?... With couple of clicks your claim process has been initiated.

# 10 Summary

Well done, you've completed the lab!

Let's review what we have done...




Created a series of SaaS endpoints to do the lab with.

Created secure managed connections to each of these endpoints using the CP4I connectors.

Created a facade API and API integration flows to process storm damage claims.

Tested the connections from within the tooling, building your integration interactively.

Deployed your integration flows as a highly available, scalable resilient Kubernetes deployment of containers and pods onto CP4I runtime on OpenShift

CP4i secured credentials using a Kubernetes secret to abstract credentials from the integration flow

Configured API Connect with a Developer Organization and a Catalog, a secure Gateway and a Portal.

Created an API definition in API connect to securely route the requests to App Connect. Added an API Key security policy to keep your API secure and added a rate-limit policy to manage your API at 100 calls/minute. Published your API to a self-service portal.

Signed up as a new consumer of your API. Registered as a new consumer, used the portal self-service features including the interactive tester.

Created Watson Assistant chatbot and a Node application.