1 Integrated Car Crash Repair Demo: 1-click cluster Instructions (OpenShift 4.4, CP4I 2020.2.1)

This lab shows you how to easily create an API Led integration using the IBM Cloud Pak for Integration (CP4I).

We will use the DTE 'Instant provision' environment to give you a 'free' ROKS cluster. We will then use the '1-click install' to install CP4I.

The 1-click install has a feature to set up a lot of what you need for the demo into your cloud pak – we will use this to save time and effort.

We will use the low-code/no-code Integration Designer to create an API which takes a car repair claim request, complete with a photograph of the car, and integrates with a SaaS CRM system and IBM's Watson AI to create a car repair case with all the correct details loaded into the SaaS system and data routed to the correct location based on the image contents; all in a few seconds before returning a response to the customer.

As an 'extension scenario' we check if the car is a convertible/roadster.

If it is, we translate the request into Spanish for our Spanish-speaking partner and create an incident in their ServiceNow SaaS system, complete with car photograph.

Due to time constraints, the 'extension' will only be briefly described – we will ensure we build an end-to-end managed API before discussing the extended scenario.

This lab may look long but don't be put off by the number of pages: Most of them are filled with screenshots and descriptions– there's not that much "work" to actually do – we've created a lot of things for you to use ready-to-go.

Let's get going and create the demo!

2 Pre-Reqs before you start: An eMail server and client

You will need a mail server (SMTP server) as this is required by IBM API Connect.

IBM API Connect uses email to send invitations to onboard user accounts and to reset passwords etc. We will need to send emails when we register new API consumers to send them account activation details.

Normally we'd use our enterprise's SMTP (email) server to send the emails and then receive them into the appropriate accounts using mail client software and inboxes.

If you do have an SMTP email server and accounts you'd like to use, that's fine: But as eMail servers can be used either intentionally or unintentionally to send excess mail, we're going to suggest you use https://mailtrap.io for both.

Mailtrap.io is a free cloud service that allows us to 'pretend to send' emails to anybody via SMTP – but they never actually go anywhere. It's quick and simple to set up a mailtrap account to use at https://mailtrap.io

(Note that this is just a suggestion to give us email capability for this lab – the use of mailtrap.io in conjunction with CP4I is in no way a formal recommendation and you can use whatever SMTP servers and mail client you are comfortable with)

What happens is that mailtrap.io provides both an SMTP email server and an associated "inbox". It keeps all the emails that we "send" and displays them it in an inbox so that we can see it. This way we don't need a mail server, email clients, multiple mail addresses and accounts to worry about.

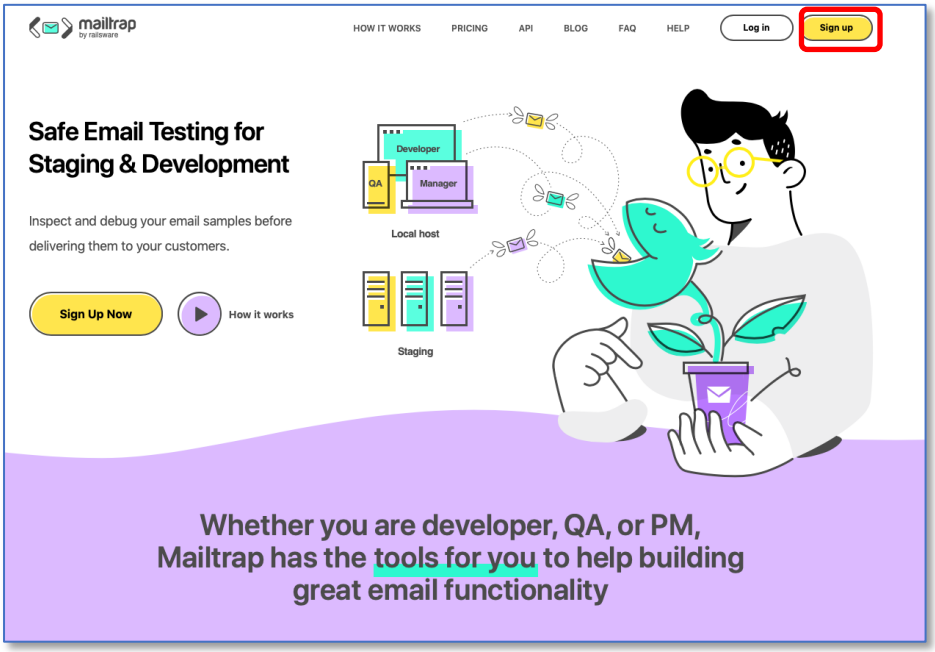
If you have a mail/SMTP server you'd like to use, go ahead and configure it here. If not, we'd suggest using mailtrap.io for this lab demo.

This gives you both a mail server and an inbox bound into one. Any mail you send via their server gets 'trapped' and put into an inbox, allowing you to read it. This is exactly what we want to do!

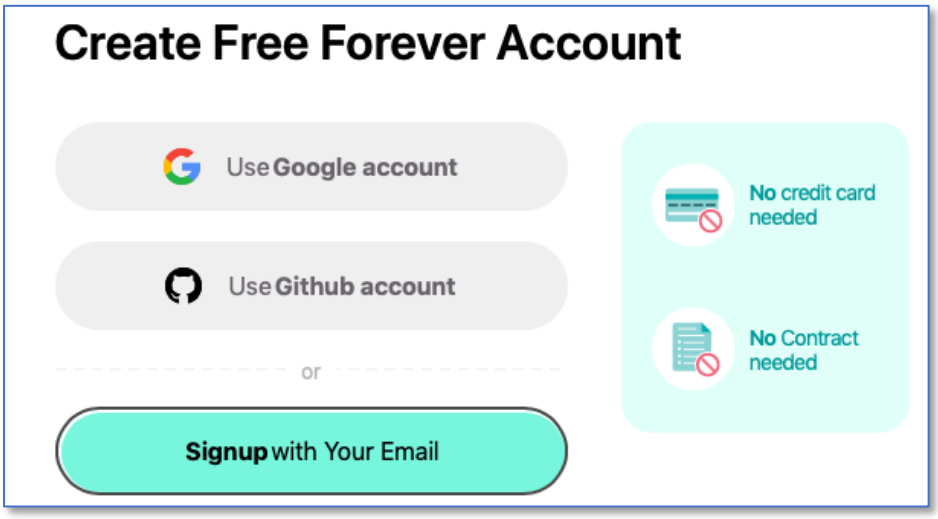
Note: If you have used a ROKS demo environment before and already have an email server to use or an existing mailtrap.io account , you can re-use this – simply enter the details in the fields when prompted.

2.1 Setting up the mailtrap.io email server

Go to http://mailtrap.io
and sign up for an account – it's instant and free.

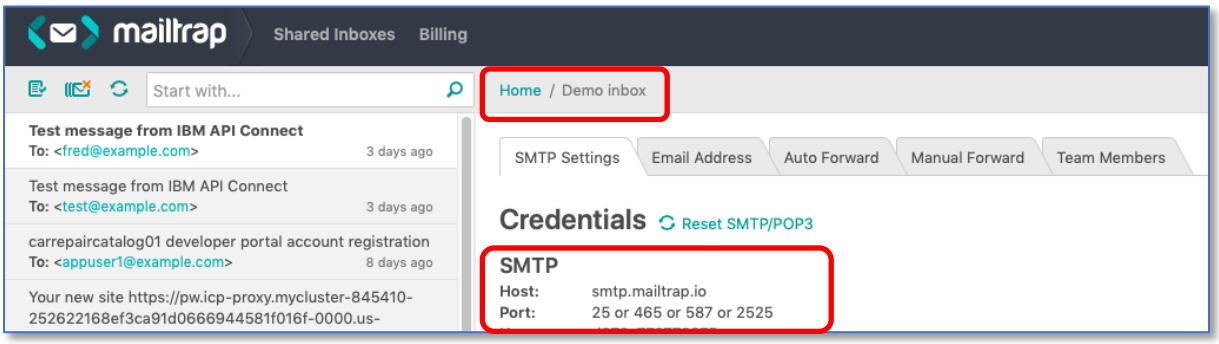


Click 'Sign up"
And 'Sign up with your email' – you can use a disposable account e.g. Gmail if you wish, or sign in with github or google.



When you have your account setup – you'll get an Inbox. Mailtrap.io acts as both a server and an inbox – really handy for what we want to do. If we sent an email to anyone at any address using the mailtrap server, it will show up in the inbox!

Click into your mailtrap.io inbox and click ‘SMTP Settings’ on the tabs – you’ll see all of the SMTP Credentials you’ll need to setup the CP4I eMail server when you install it.
(Note, use the SMTP credentials, not the POP3 ones)



You’ll need the `user Id` and `Password` from mailtrap when running the 1-click install.

The 1-click install will default the rest of the mailtrap.io settings for you.

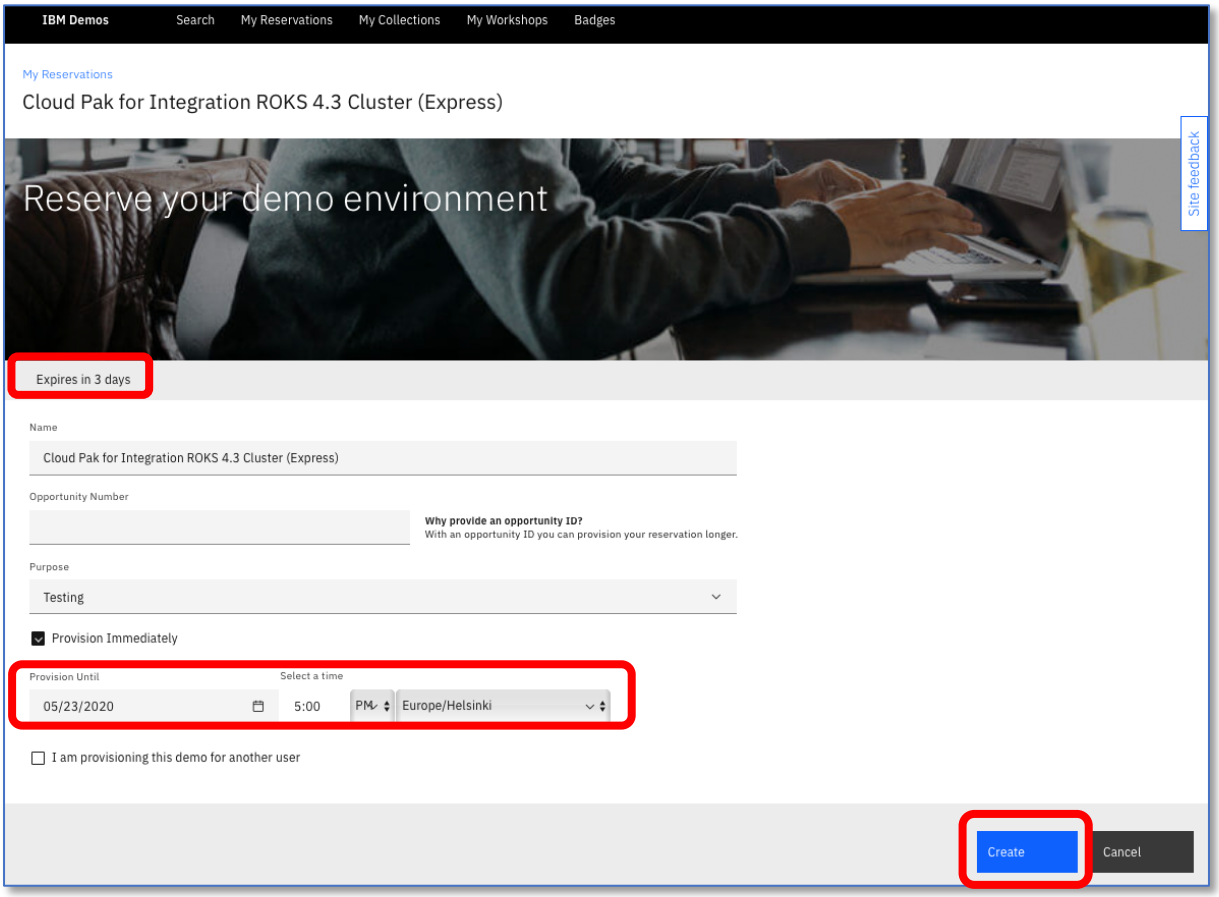
3 Installing your Demo Environment of CP4I

Create your ROKS cluster as a ‘Cloud Pak for IntegrationROKS 4.4 cluster’ following the instructions on the tile. Note that for CP4I Update 2020.2.1, you will need OpenShift 4.4 or later. OpenShift 4.3 is not supported for this update.

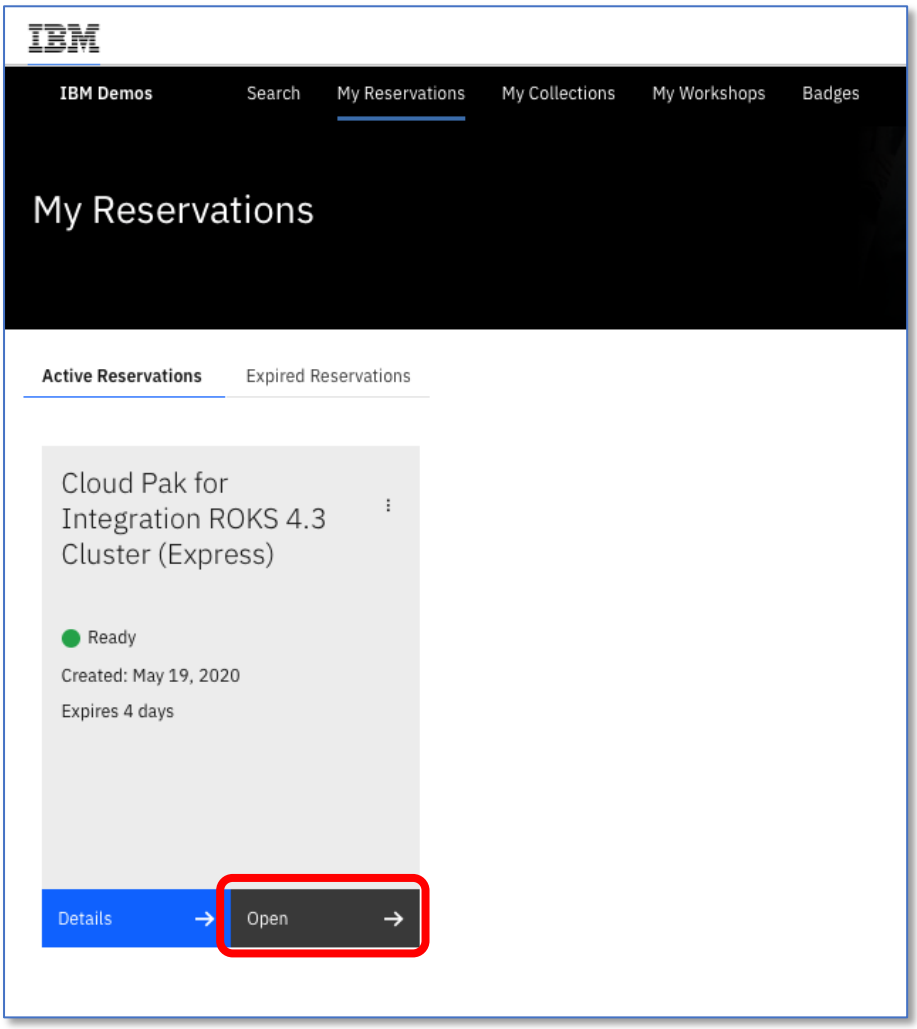
If you already have a ROKS cluster, or you want to provision one separately, that’s OK – the 1-click install will work with any ROKS cluster – just make sure it meets the minimum requirements for CP4I and that it’s version 4.4 or later.

Make sure you set your ‘Provision Until’ date to be longer than “2 hours from now.” which is the default. When you change the date, the ‘Expires in 2 hours’ message will change when you click out of the date field.

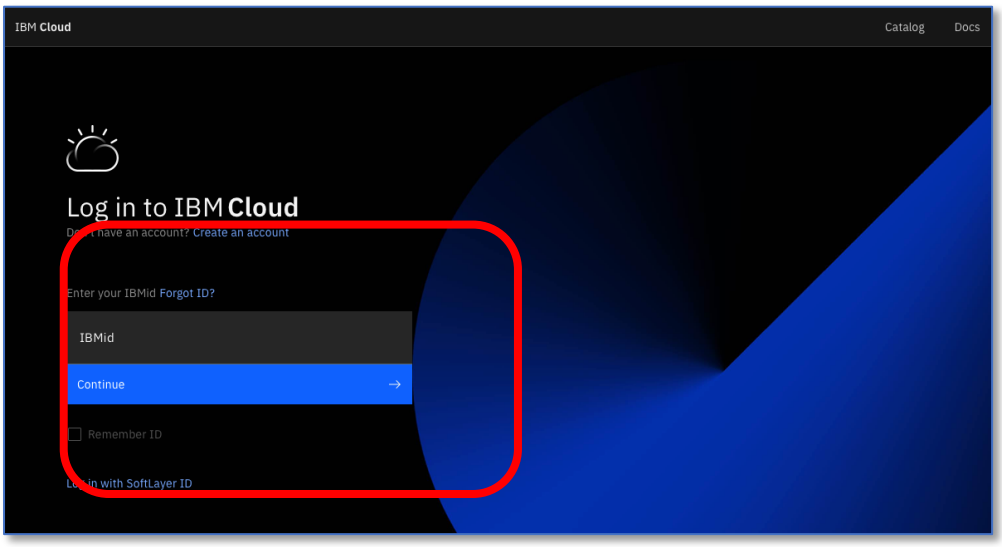
(Hint: You can provision for a decent amount of time even without an Opportunity Number)



When you've requested your instance, it'll provision quickly (usually less than a minute), and you'll see the following: Click 'Open'

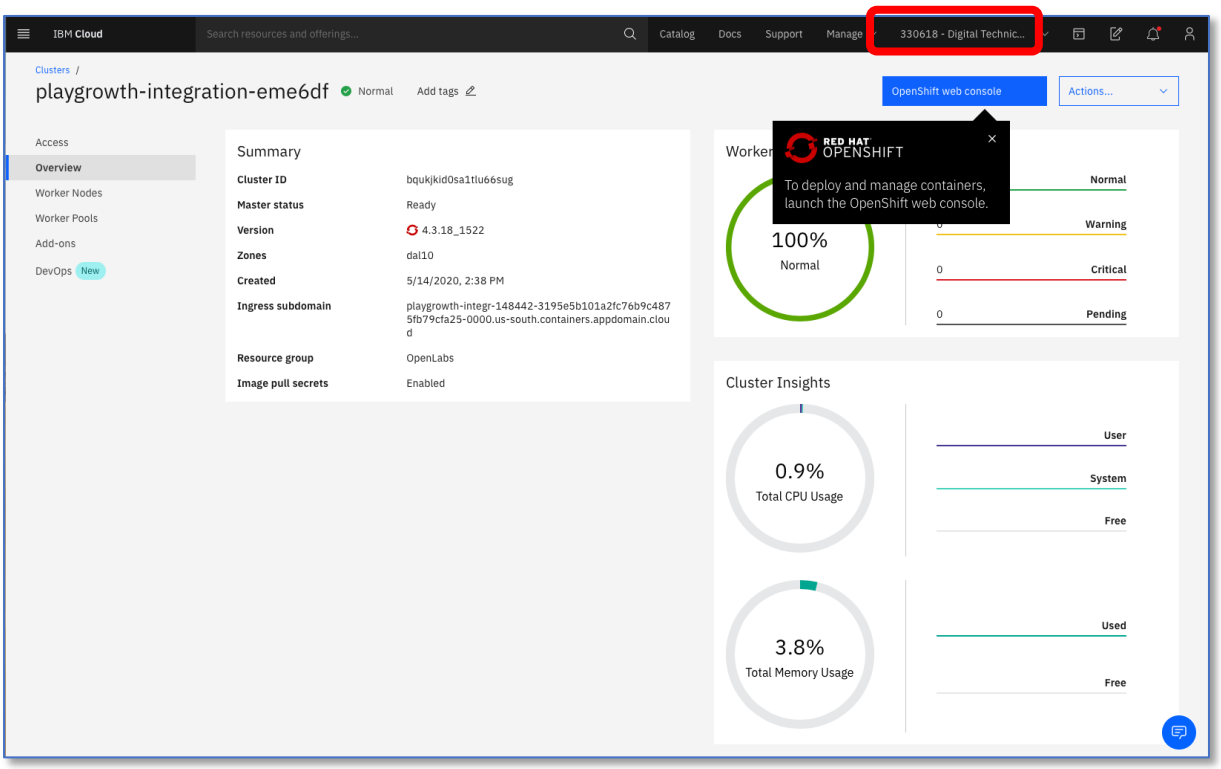


You'll then be prompted to log in with your IBM cloud account as below: Use your usual IBM Cloud credentials (IBM ID)



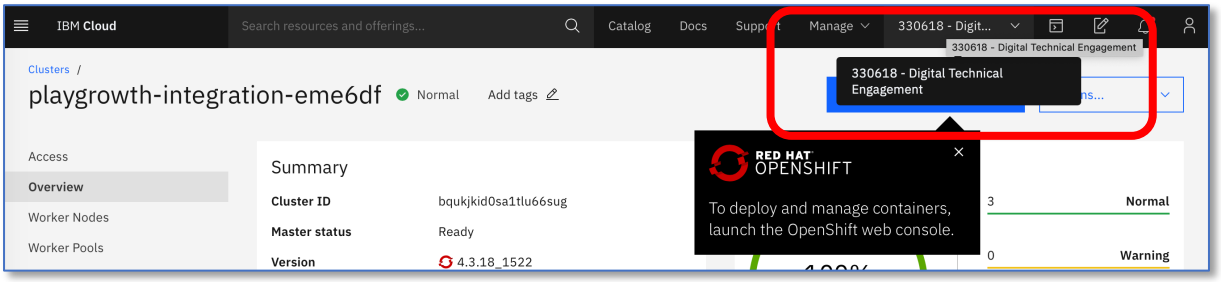
When you’re logged in, you’ll be taken to the RedHat OpenShift dashboard.

At this point, you are logged in as yourself (your IBM Cloud account), but you’ve been made a temporary member of the IBM DTE Account so that you can have complementary access to one of their DTE ROKS clusters.



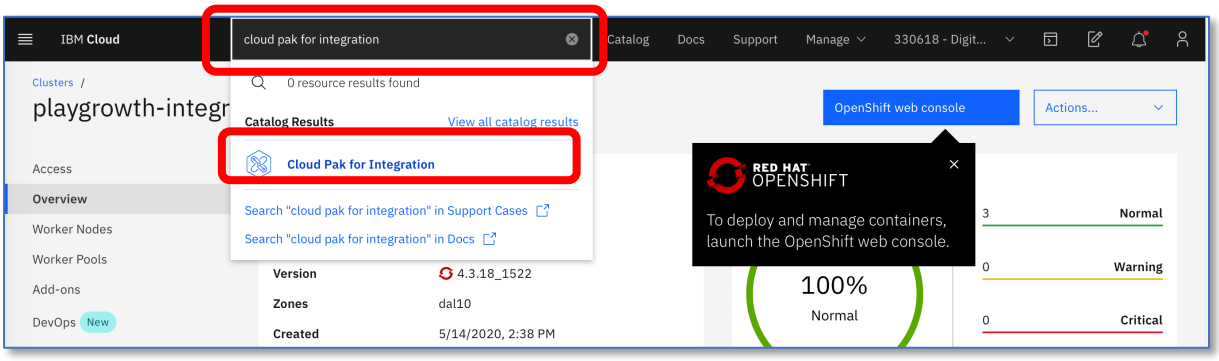
Near the top right, if you click on the ‘accounts’ to the right of “Manage’ you’ll see the DTE account.

IMPORTANT: DO NOT CHANGE THE ACCOUNT e.g. back to your own – if you do, change back to the DTE account before carrying on with these instructions.



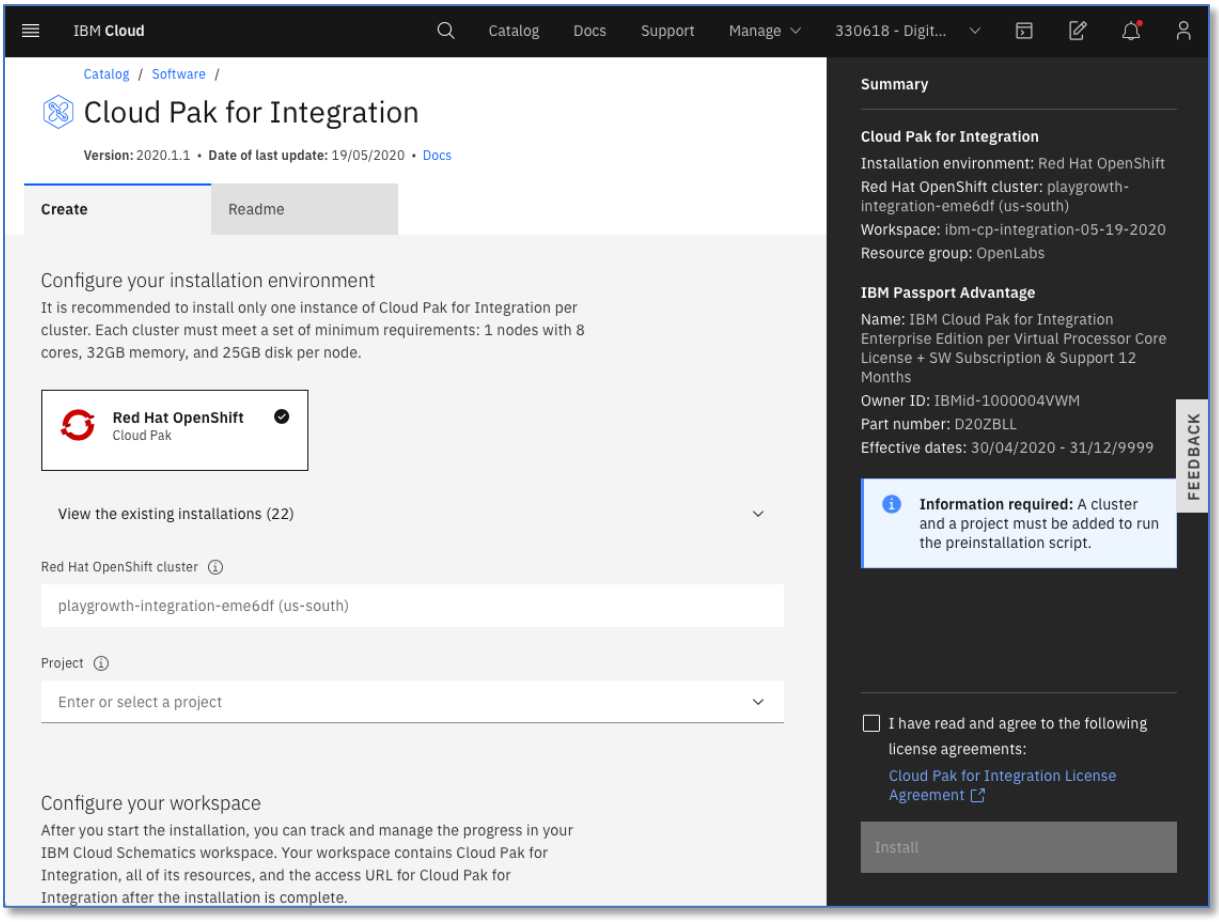
Next, Click in the Catalog Search bar and search for ‘Cloud Pak for Integration’ – when it appears in the drop down, click on it.

(If you prefer, you can go to the Catalog and browse for the Cloud Pak for Integration that way)



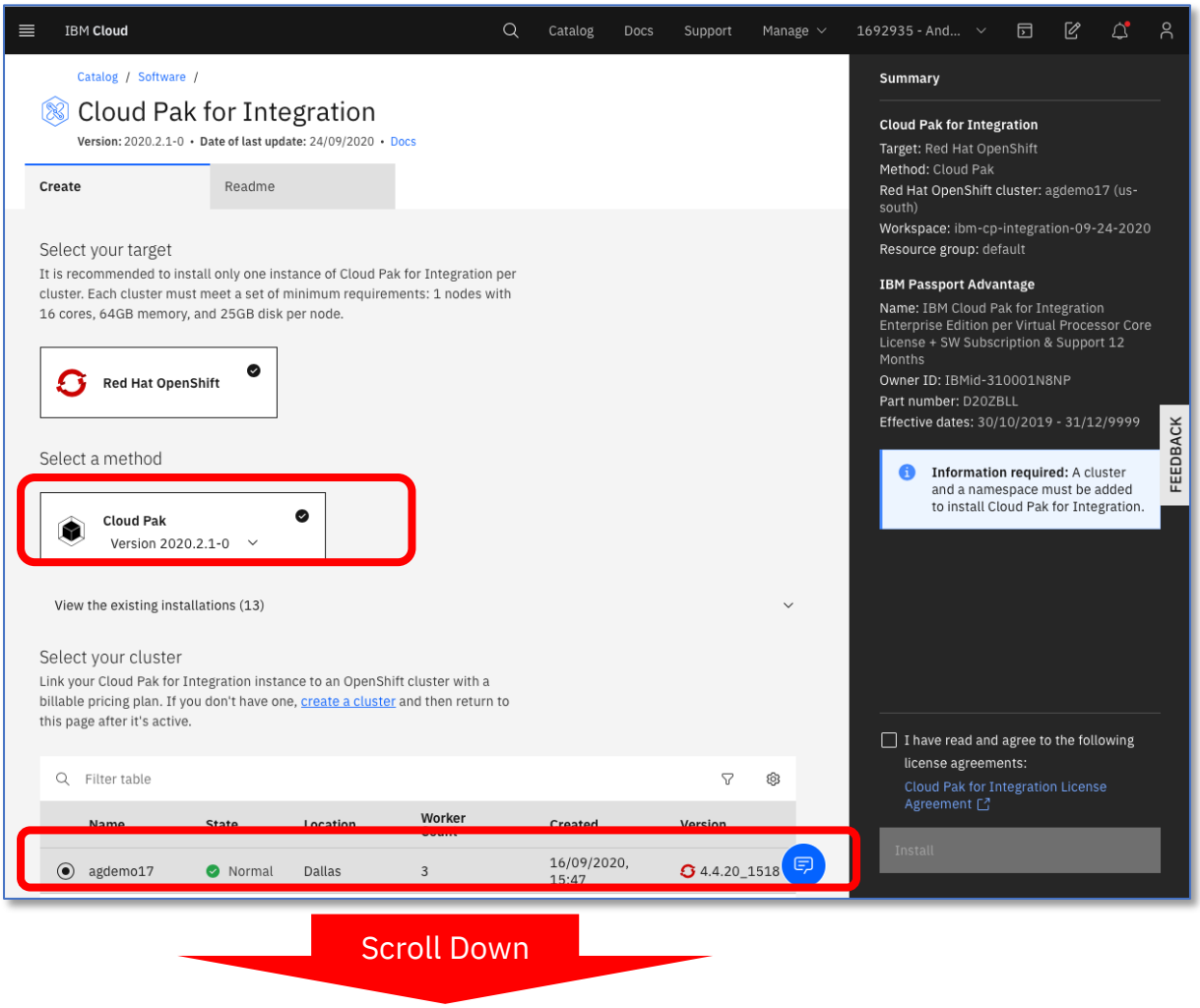
You’ll be taken to the Cloud Pak for integration catalog entry and you’ll see this: This is the ‘1-click’ install for the Cloud Pak for Integration!

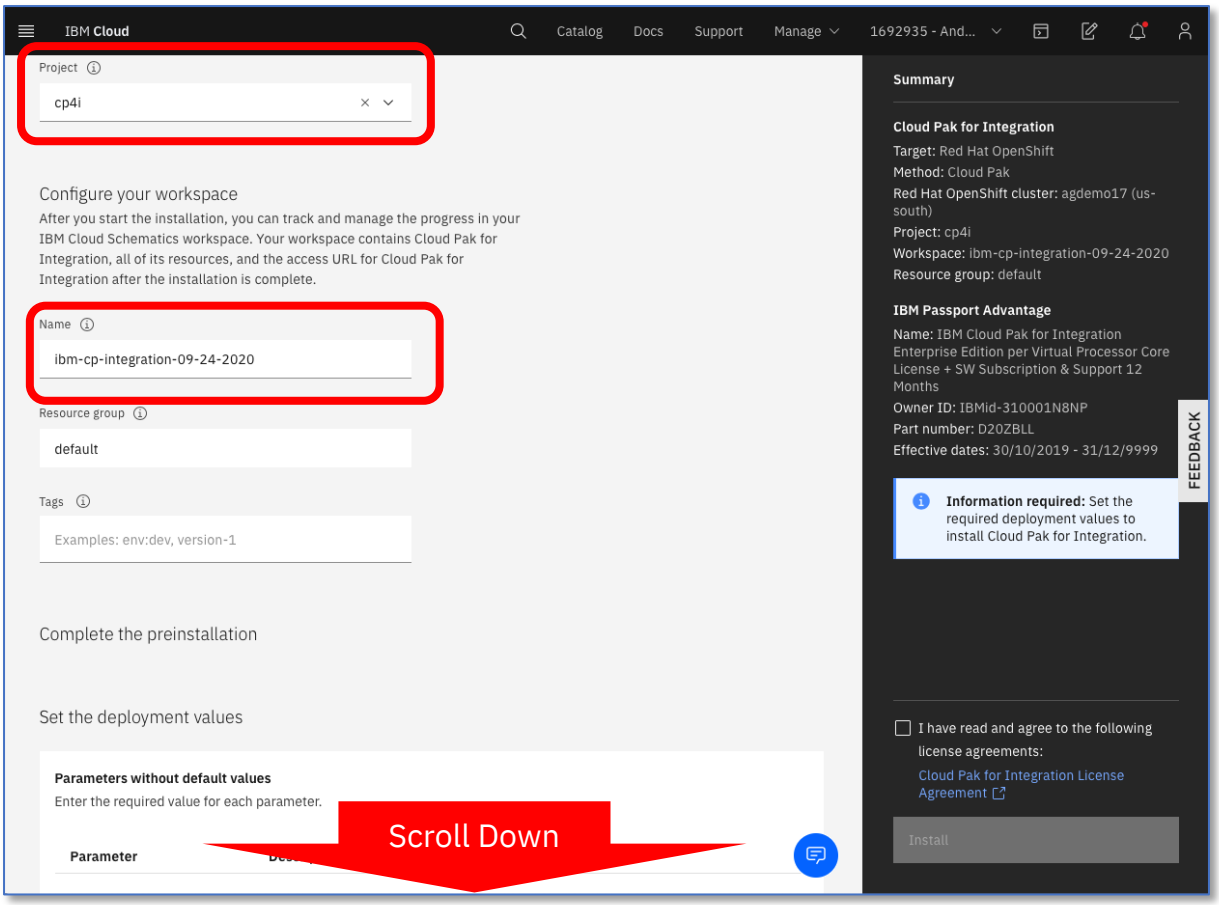
(you may need to wait a few seconds for the ‘Project’ field to load)



There are some steps we need to follow to tell it how we want to install

(**IMPORTANT**: Be careful here, especially when entering the password – at the moment, the password validity is NOT checked by the IBM Cloud before you click 'start' so make sure it's valid otherwise your install will not work and you'll need to do a lot of cleaning up to start again…)



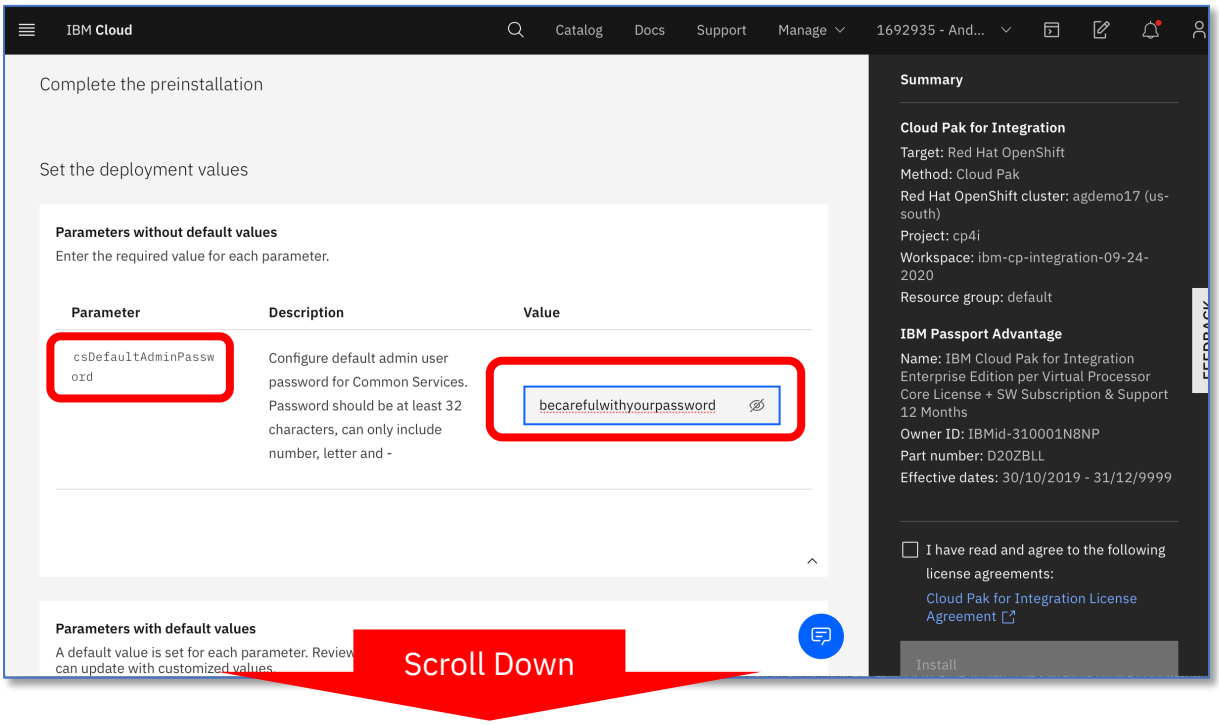


Enter a project name – it's up to you, but we used '`cp4i`'. You'll need to remember this as it will be a namespace later on

(The project name is the OpenShift project name – a Kubernetes namespace. Don't worry if you're not familiar with these terms – just use 'cp4i' and you'll be fine)

You may see 'Complete the preinstallation' – this is no longer required, although the heading may still be there.

Scroll Down to ‘Set the Deployment Values’ and ‘Parameters without default values’



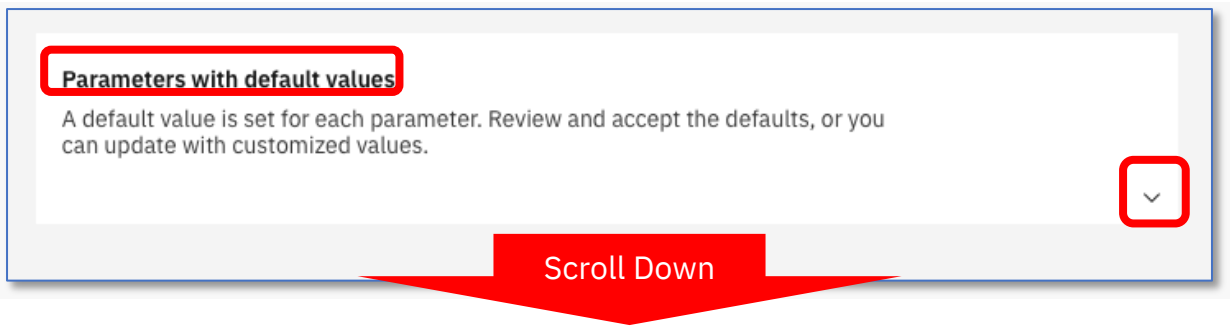
Enter the password in the field in the ‘Value’ field as shown above.

VERY IMPORTANT

- Read the rules for the password. If your password does not match the rules, it won’t be checked – until it’s too late and the install scripts will fail.
- Be careful of your typing: There is no ‘re-enter your password’ field. Use the ‘eye’ to show your password and make sure it’s correct.
- There is no ‘reset your password’ – write it down and store it safely.
- Make sure your password is long enough.
  32 characters is longer than you think!
  (In the example below, 32 characters is ‘n’ in ‘long’ and that’s counting the dashes too.)
- Password generators/managers are unlikely to generate a suitable long enough password – use your own or add extra characters to theirs.

Hint: `use-a-sentence-of-words-as-a-long-password`

Scroll down to ‘Parameters with default values’ and click on the ‘twisty’ to open them:



Scroll down until you see the fields in the screenshot below:

IMPORTANT: Make sure you set the ‘demoPreparation’ to ‘true’ You can’t go back and re-run it if you forget….

Parameters with default values

A default value is set for each parameter. Review and accept the defaults, or you can update with customized values.

Parameter	Description	Value
csDefaultAdminUser	Configure default admin username for Common Services	admin
navReplicaCount	The number of replica pods to run.	3
demoPreparation	Note: Not for Production use. Ensure the consumers of this service are aware that this instance is purely for non-production demo purposes. We recommend the name of this service indicates that the nature of these constraints. Installs ACE, ACE designer, APIC, Event Streams, MQ, Tracing, PostgreSQL and Asset Repo.	Set to 'true' true
demoAPICEmailAddress	The email address APIC uses to notify of portal configuration. The email will be sent via the configured Mail Server	your@email.address
demoAPICMailServerHost	Host name of the mail server	smtp.mailtrap.io
demoAPICMailServerPort	Port number of the mail server	2525
demoAPICMailServerUsername	Username for the mail server	<your-username>
demoAPICMailServerPassword	Password for the mail server	••••••••••
useFastStorageClass	Change the default storage class to cp4i-block-performance, normally on ROKS ibmc-block-gold will be the default. cp4i-block-performance has higher IOPS, especially for smaller PVC sizes, but will also cost more.	true

Fill in the following fields:
`csDefaultAdminUser`: leave this as ‘admin’
`navReplicaCount`: leave this at ‘3’

`demoPreparation`: Set to ‘true’

`demoAPICEmailAddress`: if you’re using mailtrap.io, use any email address. Use ‘`apicadmin@example.com`’ to be safe – `example.com` is guaranteed to not be a real domain. If you’re using your own mail server, use an email address that you can receive and read.

`demoAPICMailServerPort`: if you’re using mailtrap, leave this as 2525. If your’re using your own server, this is the SMTP port

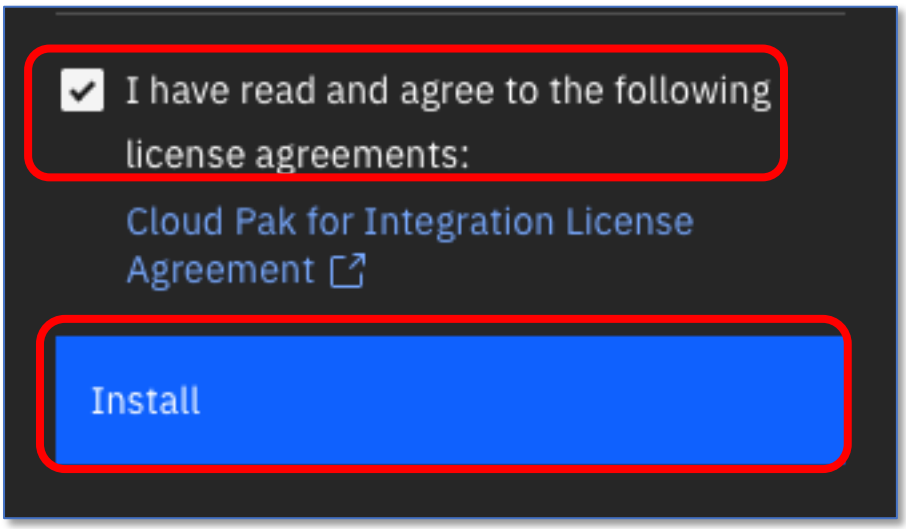
`demoAPICMialServerUsername`: Set this to your mailtrap/mail server user name

`demoAPICMailServerPassword`: Set this to your mailtrap /mail server password.

`useFastStorageClass`: this is a cost/performance decision. We set this to ‘true’.

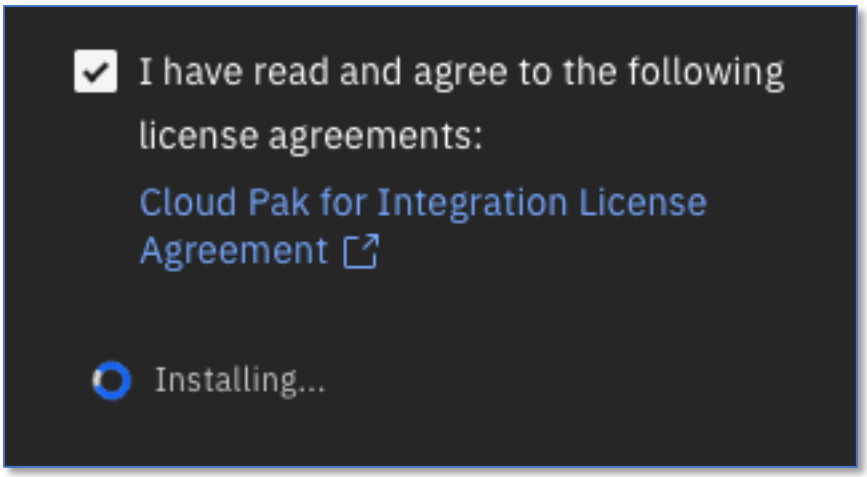
At this point, download the License Agreement using the hyperlink.

When you have read it, if you agree to it, tick the box and click ‘Install’

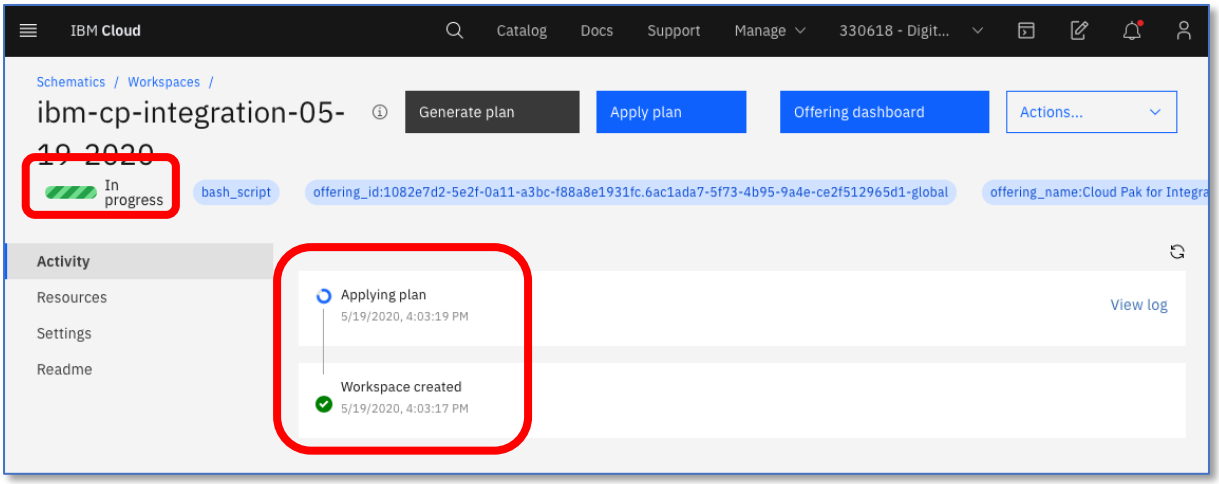


And you’ll see the Cloud Pak installation start, with all of the products you need, configured ready for your demo ready to be installed automatically.

Install changes to ‘Installing’ to indicate that the install is running.



You'll then drop back to the progress screen where you can keep an eye on it if you wish.



This will take about an hour and a half to run – time for a tasty beverage.

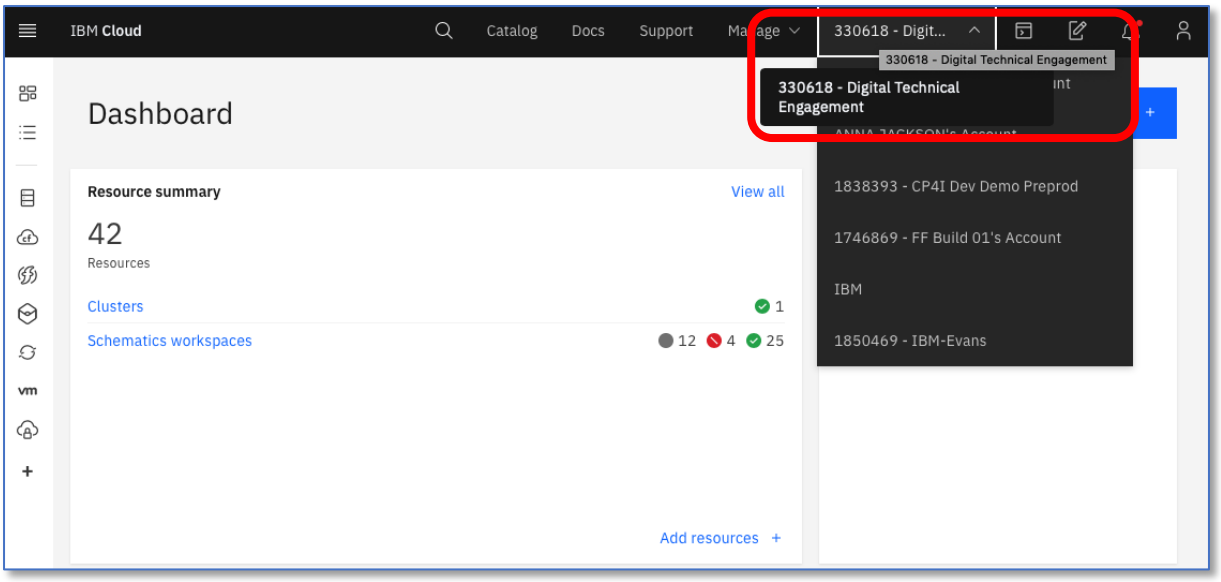
You don't need to stay and watch it, you can go and do other things.

We recommend that you set up your endpoints ready for the rest of the demo so they're good to go (see section 8) when then cloud pak for integration is installed. See further below in this guide.

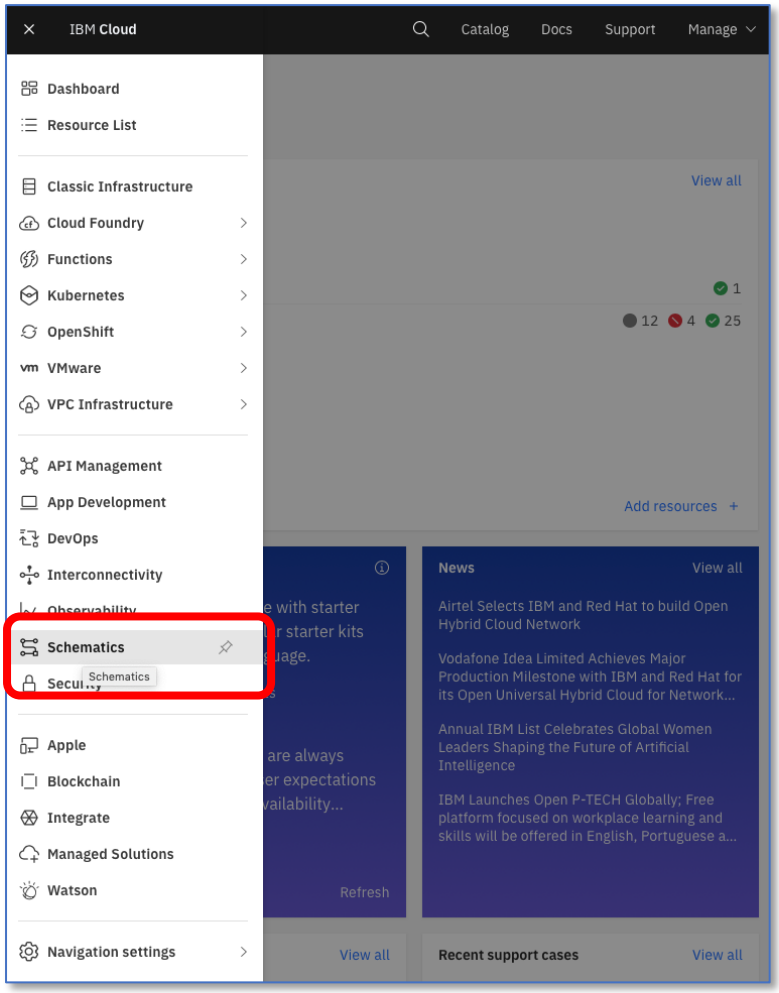
When you want to check on the progress of your installation, do the following:

Go to https://cloud.ibm.com and sign in with your IBM id:

Go to the 'Account' pull down and make sure you're in the 'DTE' account: If you're not pull down and click on the DTE account to change it.

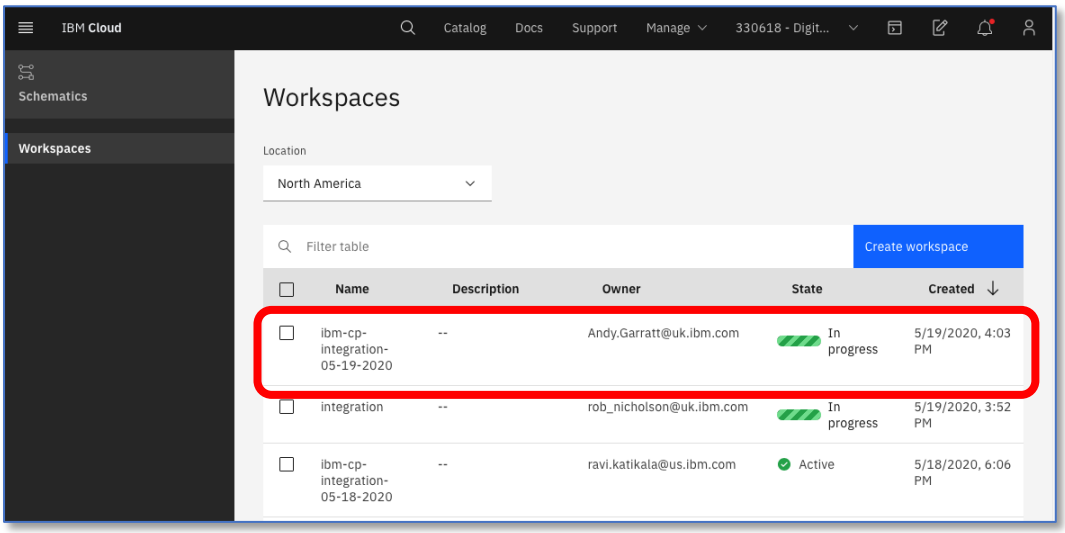


Next, click on the ‘Hamburger’ menu and click ‘Schematics’

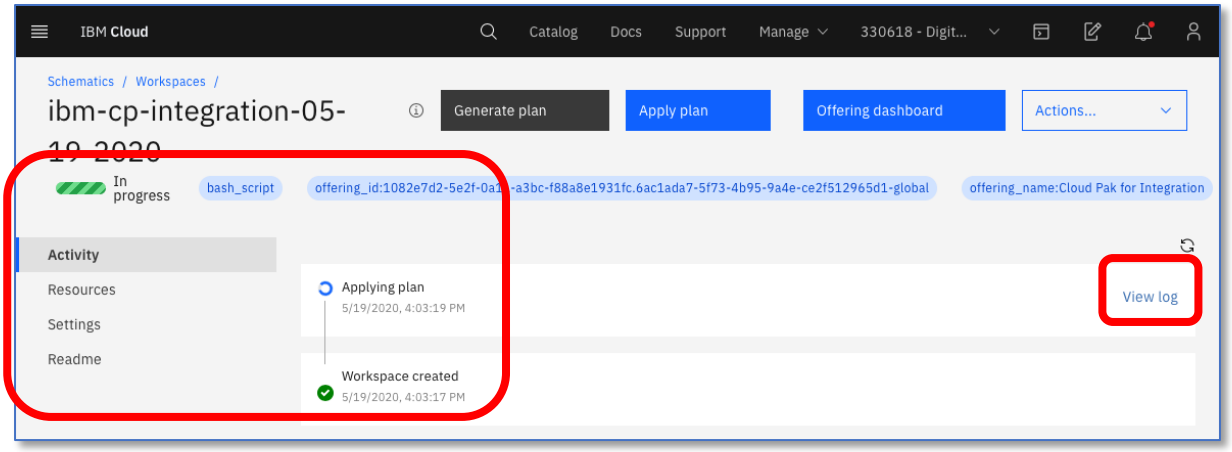


You’ll be able to see if your install is running (This is a shared cloud account, even though you will get your own ROKS cluster – look for your email to see which one is yours)

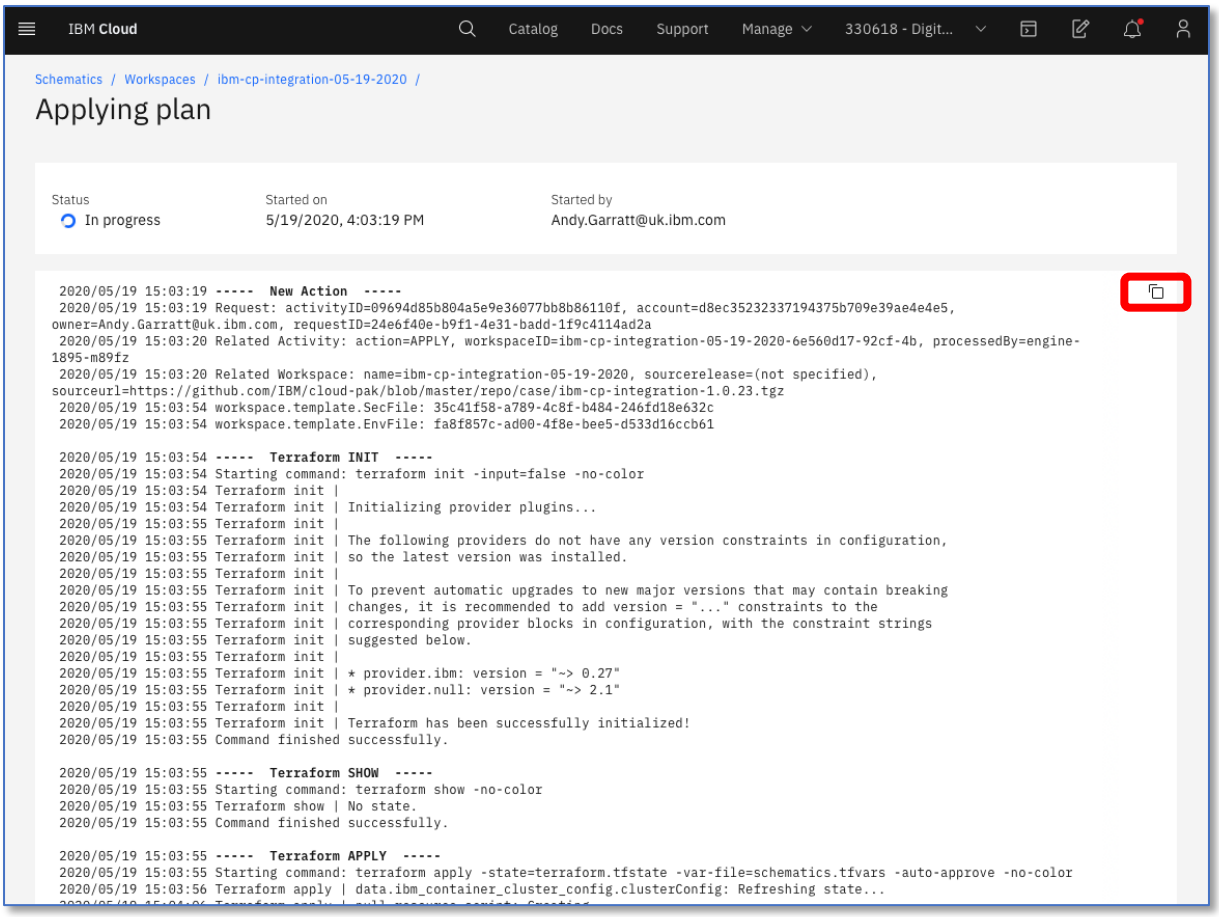
Click on your installation and you can see how it's progressing



And you can look at the logs in 'View log' if you're interested:



You can export the logs using the copy icon if you need to.



At this point, during the time it takes to install, if you do not have a Salesforce developer account, a Watson Image Recognition Service and a Watson Tone Analysis Service, go to section 8 and create these endpoint services ready for when the Cloud pak for Integration is installed.
Also, read the business scenario and plan of work, so you're familiar with what we're going to do.

4 The Demo Business Scenario

We are a Car Repair company: We take in vehicles with problems and repair them – seems simple but..

- We want to gain business advantage by allowing multiple car leasing companies to use us to repair their cars – these companies insist that we expose APIs for them to call to do business with them.

- We want to allow their customers to book their cars in for repair and get an estimate for price and number of days in real time – in seconds. Later we will build more APIs to allow customers to query the status of their repairs, or make updates or add comments to their repair cases.

- We want to allow them to send photos of their cars so we can check for type, damage etc.

- We want to check for errors and issues up-front as quickly as possible to feed back to the customer in real time. Photo not valid? No car in the photo? We'll tell you instantly so you can re-submit.

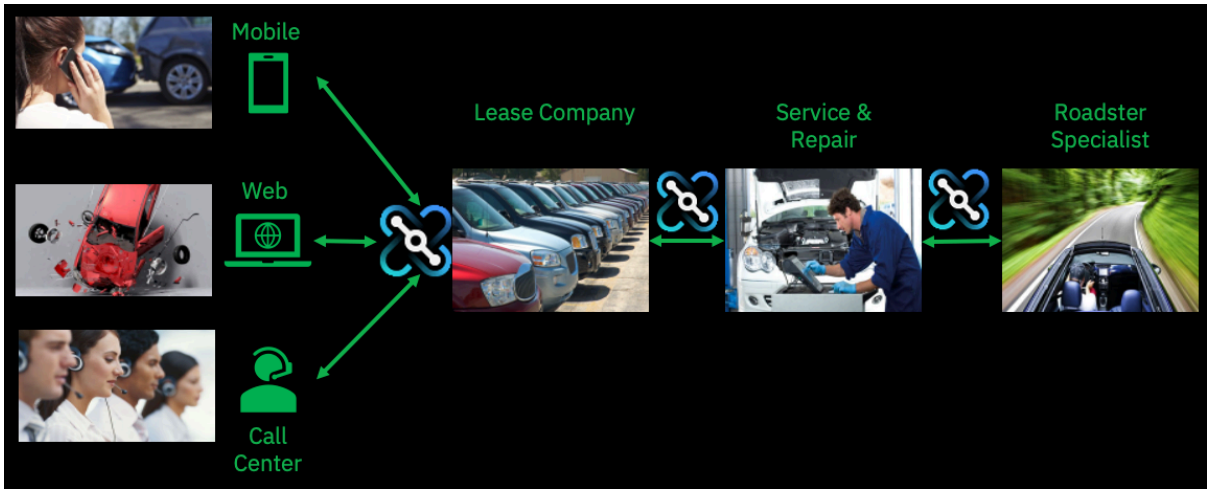
- We want to minimize manual processes and have the repair request automatically create a repair case in our CRM system (Salesforce). If a customer wants to book a repair at 3am on a Sunday, they can – it's their choice.

- (Extension scenario): If the car is a convertible/roadster, we don't repair it, it's a specialist job. We want convertibles to automatically create a repair case in our partner's system (ServiceNow) as well.

- (Extension scenario)Our partner speaks Spanish – we don't! We need the requests translating from English.

- We are wanting to grow our business fast with this new model and expect the use of APIs to really increase the number of requests we get. We need our solution to be scalable and highly available.

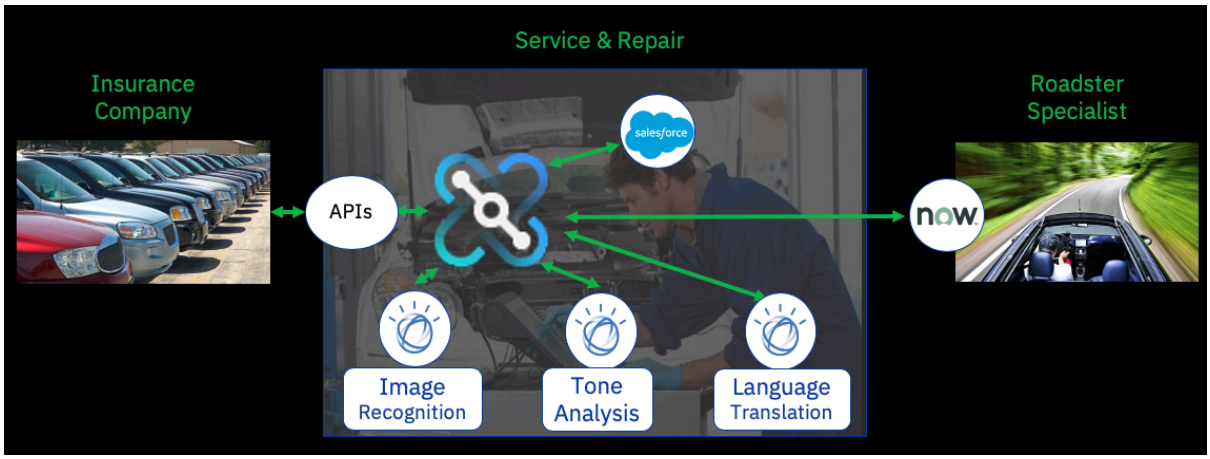
The diagram below shows how all the parties of our business are related:



5 Our API-Led Solution

We want to create an API which enables customers to send us photos of their cars along with descriptions of what needs to be done and their details such as their name, email and car license plate.

The picture below shows what we would like to build:



There is a video showing how this whole scenario is built end-to-end on YouTube here: https://www.youtube.com/watch?v=TRzO26kawu4 .It's about 8 minutes long and might be worth watching to see the whole thing in action and what we are aiming to do.

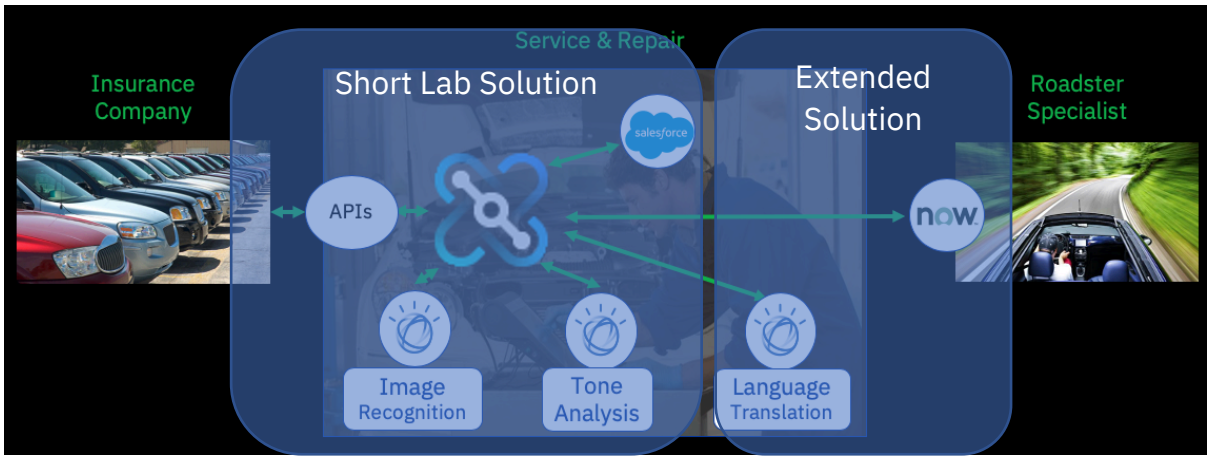
In this lab, we are going to build a shorter version of this which includes the IBM Watson Image Recognition, IBM Watson Tone Analysis and the Salesforce integration.

We will not use the ServiceNow integration or language translation in the 'Short' version of the scenario.

Instructions (and artefacts) to help you extend this to the longer 'extension' version will be given at the end but we will focus on getting something working end-to-end as easily as possible to show you the flow.

5.1 Solution-Flow Breakdown: ‘Short’ version of the solution

- Receive the Customer’s car repair request including their photograph via an API
- Use IBM Watson Image Recognition to analyze the photo. If it is not a valid picture, Watson will return an error immediately to the user calling the API.
- Check that Watson can ‘see’ a car in the picture – if not, we will immediately respond back with an error saying ‘There is no car in this picture’ so the error can be corrected immediately.
- Create a ‘Case’ in Salesforce with the data from the API. This Case is where we store the details and progress of our repair.
- Analyze the description of the problem as described by the customer using IBM Watson Tone Analysis. We store this in Salesforce for future reference – if the customer is angry or upset, we may wish to take further action or treat them more carefully.
- Add an attachment of the photograph to Salesforce so that we have the image stored in our system.
- Send a response back to the customer with their Salesforce case reference for future enquiries and also an estimate of how long it will take to repair and how much it will cost (These are hard coded in this lab – in the extension, it is more expensive to repair a roadster for example)



5.2 Scenario Flow Breakdown ‘Extended’ version

(We will not build these steps in detail in the lab, but guidance and artefacts will be provided at the end if you wish to try it and you have time)

- When Watson analyzed the image, look to see if a ‘Roadster’ (or convertible) car was detected in the image.
- If so, we will send it to our partner for repair. Our partner uses ServiceNow, not Salesforce so we need to create an incident in their ServiceNow system – automatically
- Our partner speaks Spanish and we don’t: No problem, we will use IBM Watson language translator to translate our request into Spanish before we send it to them
- We will also create an attachment of the car photo into their ServiceNow system.

6 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:

6.1 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 Salesforce or ServiceNow will be running at customers already – this is a lab task.

6.2 Create an integration flow for our 'Car Repair Claim API'

This will create our API 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.

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

Once we have developed our flow and tested it, we will deploy it to CP4I running on OpenShift.

This will create a Kubernetes container/pod deployment with three highly available replicas which we can then scale up and down as we wish. (we will not show scaling in the lab but it will work if you want to try it)

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

An API is no use if our consumers/consumers can't discover it, get the information they need to use it and preferably sign up for access in a self-service manner.

Once they have access to our API, we need to make sure that the API is secured using API keys and is rate-limited using API plans.

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

Whilst we won't be building an actual application to test our API, what we will be doing is showing how applications are registered to our API server.

Registering Applications allows us to assign API keys and also to monitor the calls made to the API by that application.

7 List of things we will need:

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

7.1 List of Systems and Services Endpoints

The systems and services we will use are as follows: Instructions for these are further on

7.1.1 Salesforce:

Salesforce is a CRM system provided as a SaaS i.e. it is hosted in the cloud.

In this scenario, we as a car repair company will use Salesforce to create and store our car repair claims.

Salesforce 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. If you already have a developer Salesforce account, you can use that.

7.1.2 IBM Watson – Image 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 our case, we will use Watson to check if there is a car in the picture and, in the extended version, if it is a convertible/roadster car or not. If it’s a roadster, we’ll send it to our partners. If it’s not, we’ll repair it ourselves. If there’s no car in the photo, we’ll send it back and let our customer know.

7.1.3 IBM Watson – Tone Analysis:

IBM Watson can tell if someone is happy or sad or angry or many other emotions!

If your customer is angry, you want to know so you can make them happy – we’ll use this to look at the customer’s description of the damage/problem and put the tone into our Salesforce case so that when we call them, we know what to expect.

For the ‘extended’ scenario, we will also use:

7.1.4 ServiceNow:

ServiceNow is an incident management system which is provided as a SaaS on the cloud. Our repair partner which repairs convertible/droptop cars uses this system to manage their repairs.

ServiceNow allows you to create free developer instances as well – you will need one to extend this lab.

7.1.5 IBM Watson – Language Translation:

Our convertible car repair partners speak Spanish – and our Spanish isn’t great (yours might be!).

Fortunately, Watson speaks Spanish and many other languages better than we do, so we’ll use Watson to translate our ‘please repair this car’ request before we put the request into our partner’s ServiceNow system.

7.2 IBM Cloud Pak for Integration (CP4I) Capability list:

The Cloud Pak for Integration contains components and capabilities to implement multiple integration patterns – we won’t be using all of them in this lab.

We will be using the secure gateway (DataPower) to secure our APIs as part of API Management but we will not be using messaging (MQ), event streaming/Kafka or the high-speed Data Transfer (Aspera) in this lab.

These capabilities are all included in CP4I and there are other labs and demos available to help you explore these.

We will use the following CP4I capabilities in this lab:

7.2.1 Application Integration (IBM App Connect):

IBM App connect provides a low-code/no code integration capability with a large number of pre-built connectors to connect to a variety of different endpoints.

We will use this to create the API contract with a simple model and then the integration flow API Implementation that is started with our API call and which contains all the integration logic and data transformation.

7.2.2 Smart Connectors

These connectors contain everything needed to connect to the systems and endpoints – we just need to give them the endpoint location and credentials.

We will use one connector each for

- SalesForce
- Watson Image Recognition
- Watson Tone Analysis.

For the extended version:

- Watson Language Translation
- ServiceNow

The connectors will take care of things like authentication, session management, retries etc – you just need to give them credentials to connect and they will handle the rest.

7.2.3 API Management (IBM API Connect)

Once we have our integration API built, we need to expose it securely to the outside world via an API gateway.

We also need to be able to create a self-service portal to allow consumers to discover out APIs and sign up to use them.

We also need to create rate plans to limit how many times the API can be called.

We will push our API from the Application integration capability directly into the API Management capability where the API Product and API artefacts we need will be created for us automatically.

We will then add security and rate limiting plans and publish our API to our secure gateway and portal.

8 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
- Salesforce
- IBM Watson Tone Analysis

(If you already have Watson Services and a Salesforce Developer Account e.g. from doing a previous demo, you can skip this section)

Later, we will connect to the following endpoints.

We will set up the language translation in the 'main' lab as it is similar to the other Watson services and done in the same place, but we will leave ServiceNow until later.

- IBM Watson Language Translation
- ServiceNow

8.1 Setting up IBM Watson Services

We will set up Watson Image Recognition, Tone Analysis and Language Translation.

We can set up all three IBM Watson services at the same time.

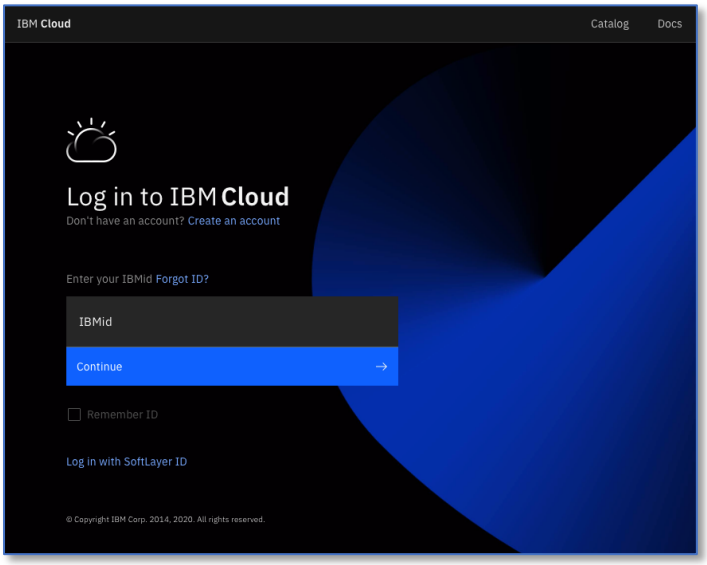
You will need an IBM Cloud account to do this. You can use your existing one if you wish or you can set up a new one.

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.

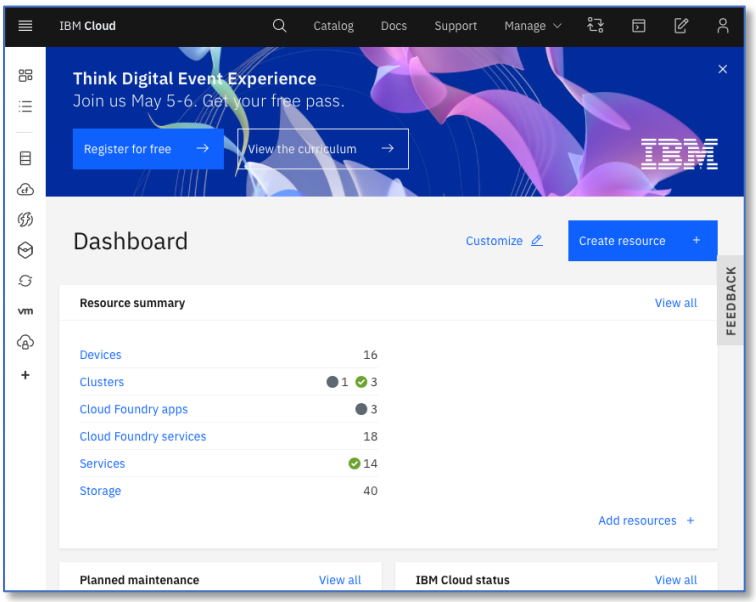
8.1.1 Logging in to IBM Cloud

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



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:



(You may not see this many services, clusters etc – the lab authors have many things in their IBM Cloud accounts.)

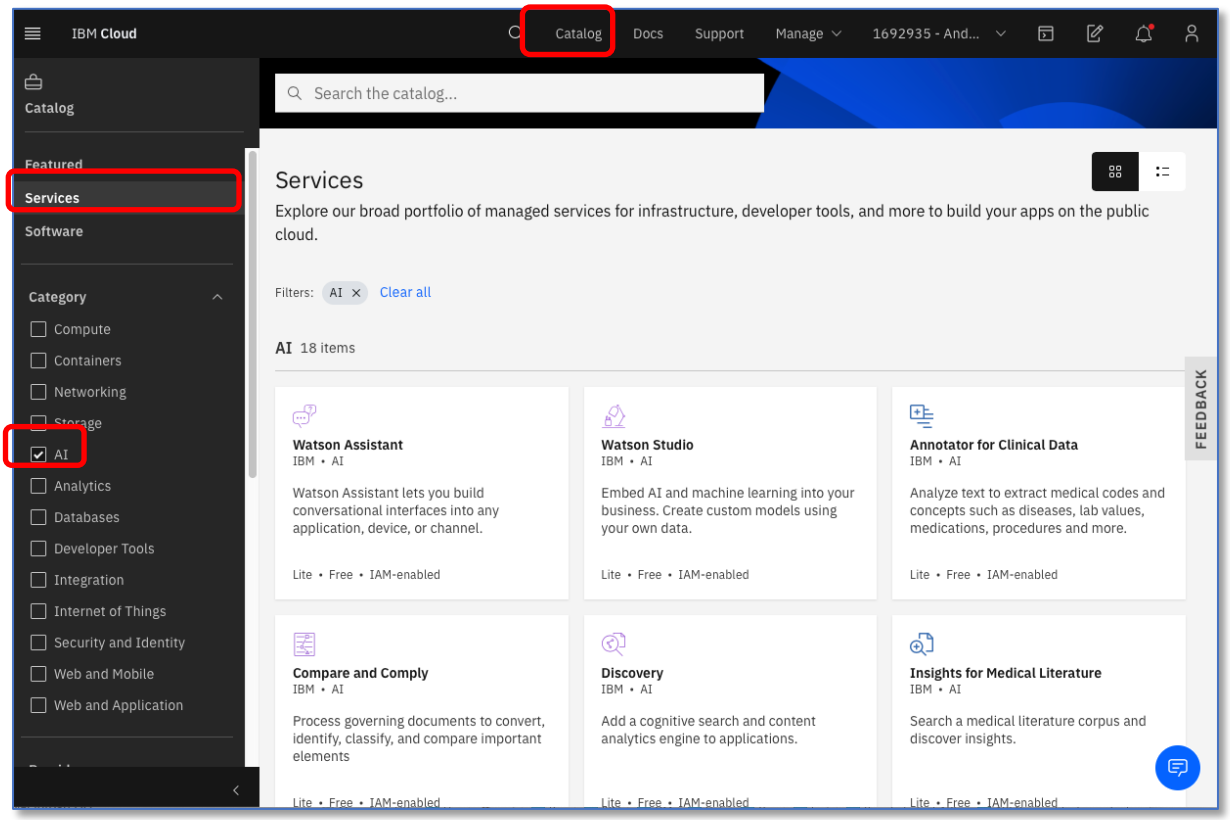
If you already have the IBM Watson services in your account, or you know how to create them then skip to 'Obtaining your Watson Credentials'

8.1.2 Creating your free Lite-Plan IBM Watson Services:

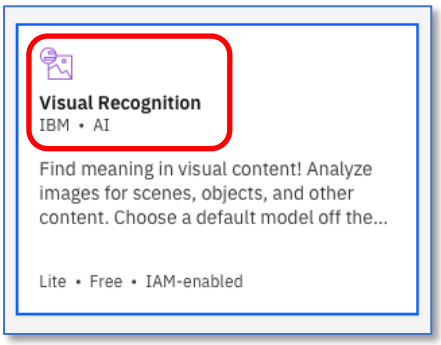
On the IBM Cloud Dashboard, click ‘Catalog’.

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

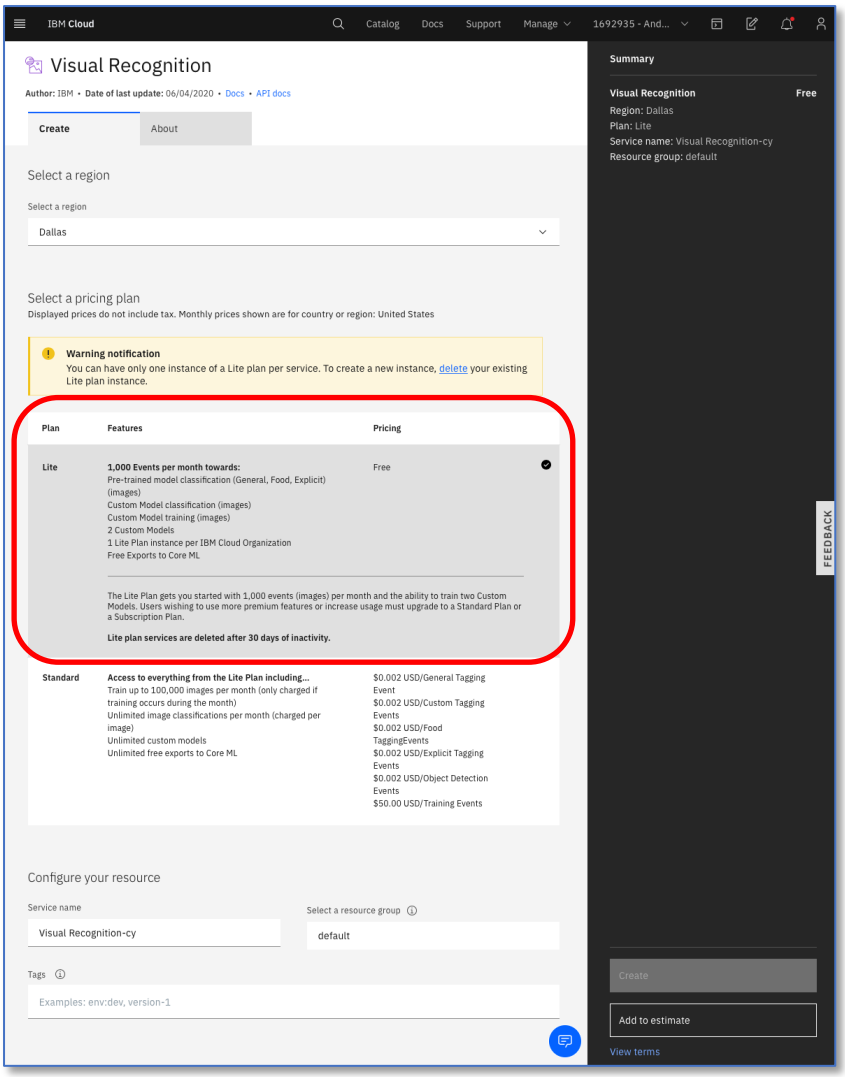
Check the ‘AI’ filter checkbox on the left to filter for Watson services. (You can also search for them by name)



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



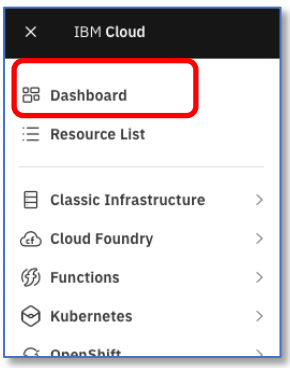
Inside, you’ll be able create a lite plan (free) instance as shown in the screenshot below (ignore the warning on our screenshot about only being able to have one lite plan per account– that’s because we already had one lite instance set up in the lab authors’ IBM Cloud Account)



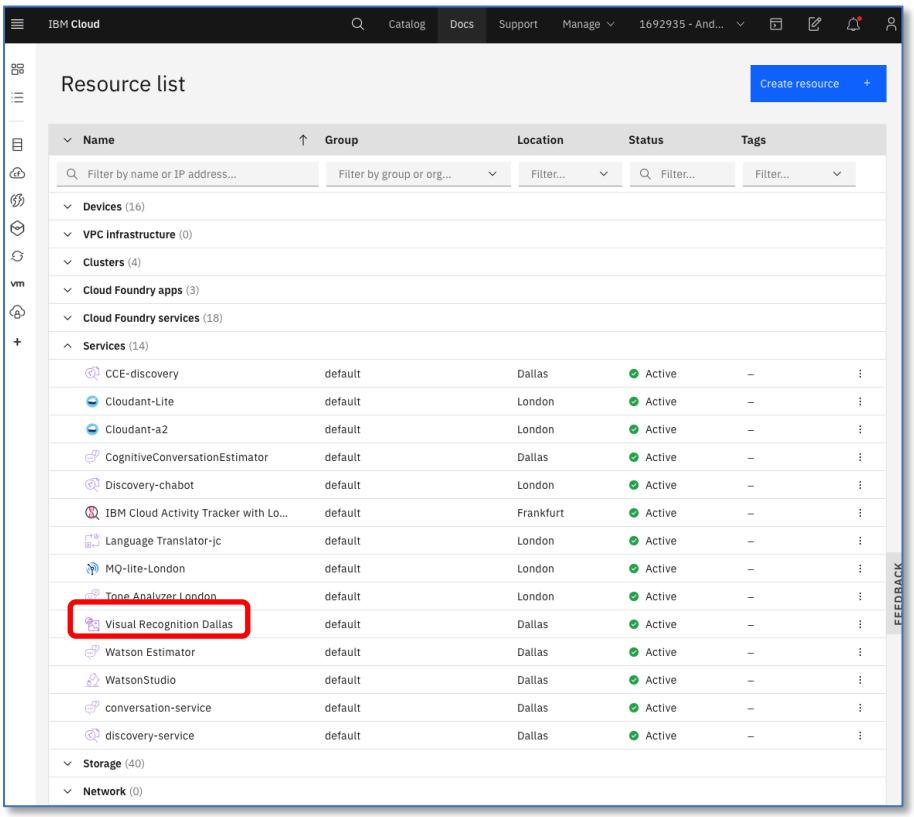
Select the free ‘lite’ plan and provision the service.

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

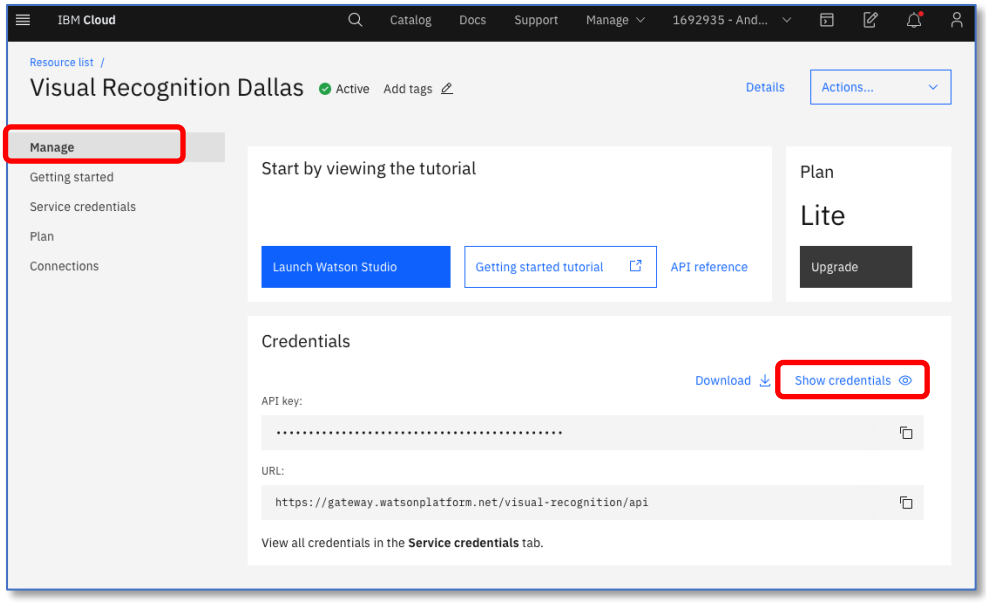
Once you create the service, you’ll be able to see it in your cloud dashboard (to get to the cloud Dashboard, click the ‘hamburger’ menu at the top left of the screen and select ‘Dashboard’



Look under ‘Services’ and you’ll find your newly created service (you can see a number of services in our screenshot below – we’ve renamed ours to add ‘Dallas’ on the end but yours will have a similar name)



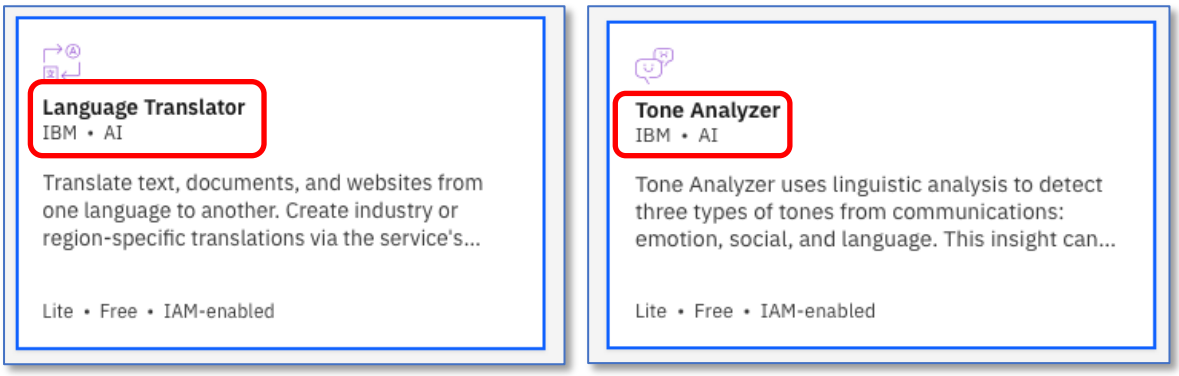
Click on your new service and you’ll see the ‘Manage’ tab:



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.
You'll next need to do similar for "Language Translator" and for "Tone Analyzer" – both have free 'Lite' plans and are set up in the same place on the IBM Cloud.

If you're offered a choice of region, take your pick. If it influences your choice, the lab environment is cloud-hosted in the USA but it shouldn't make that much difference.

Make sure you obtain the URL and API keys (in 'credentials') for all of these Watson services – we'll be needing them later. You obtain the URL and API keys for all 3 services in the same way.

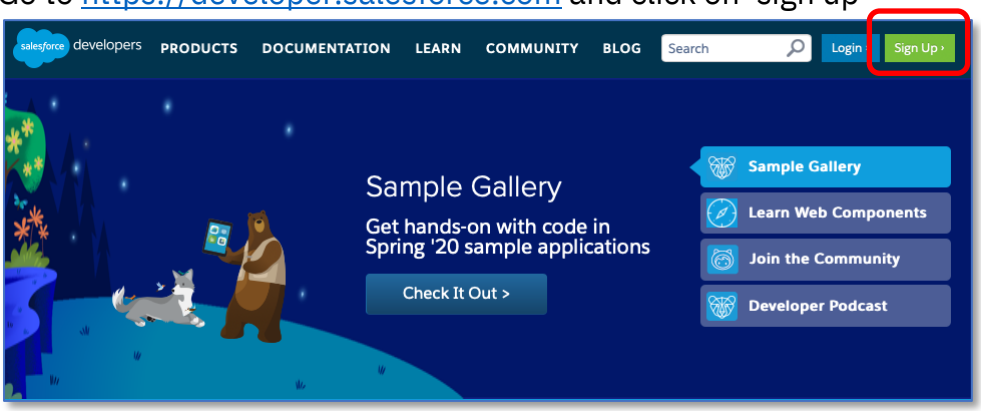


8.2 Setting up SalesForce

Salesforce is a CRM system hosted as a SaaS in the cloud.

We will need a **developer** account to use for testing – if you already have a Salesforce developer account, you can use that – if not, you can sign up for a free developer account now.

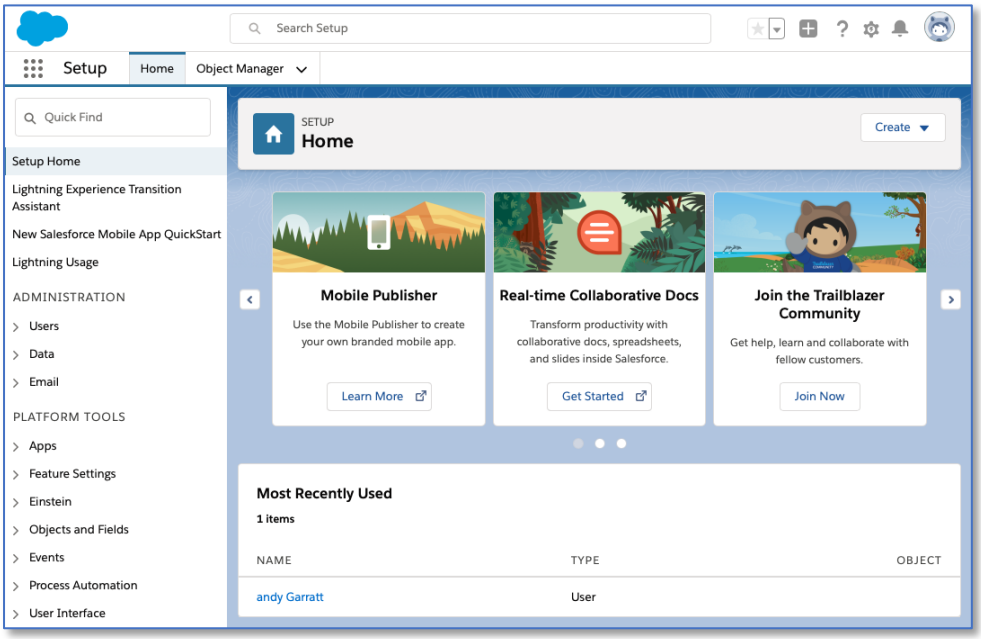
Go to https://developer.salesforce.com and click on 'sign up'



Not that this is NOT the same as ‘salesforce.com -> try for free’. **You will need a developer account to use this lab**. You can use a webmail email address to sign up if you wish, rather than your company one.

(we emphasize this a lot but one of the most common reasons for ‘My integration to Salesforce doesn’t work’ is that the account being used is not a developer one). Once you have a salesforce developer account, log in to check it – you’ll get to something like this:

(Remember to log in at your salesforce developer/instance URL, not just at salesforce.com)



You will require admin level access to your Salesforce account.

When you create a free Salesforce account to test, make sure that you create a Developer account rather than a Trial account. If you connect to App Connect with a ‘Free Trial’ account, the Salesforce integrations will not work.

OK, we have our endpoints – we’re ready to integrate!

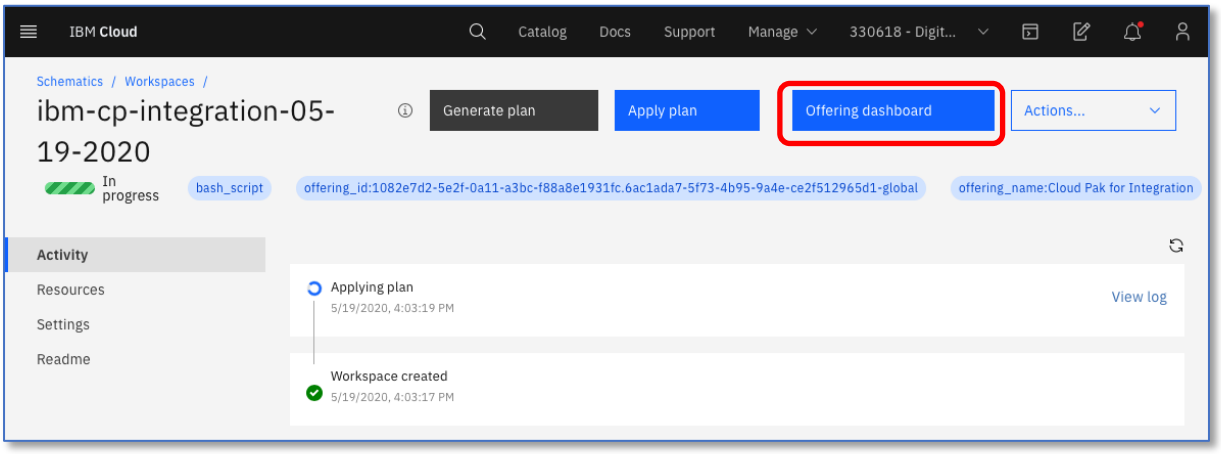
9 Getting into the Cloud Pak and Building Your Integration

For this lab, an instance of the IBM Cloud Pak for Integration (CP4I) has been created for you. It's set up running on Redhat OpenShift 4.3 in the IBM Cloud.

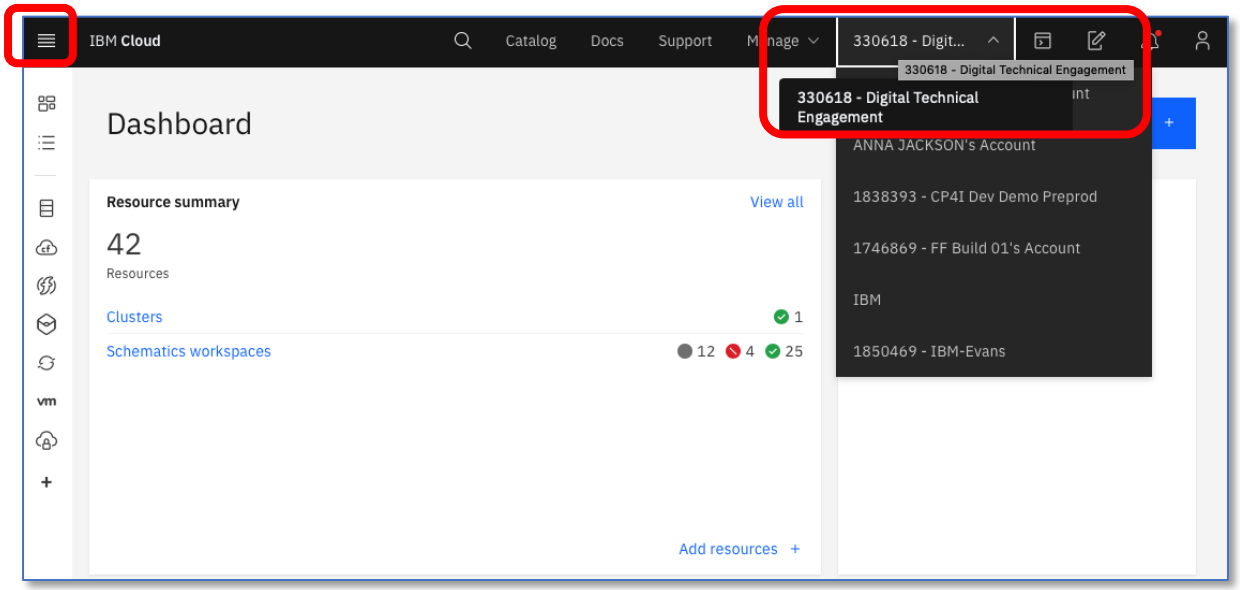
9.1 Accessing CP4I

Use a browser to navigate to the CP4I Home Page.

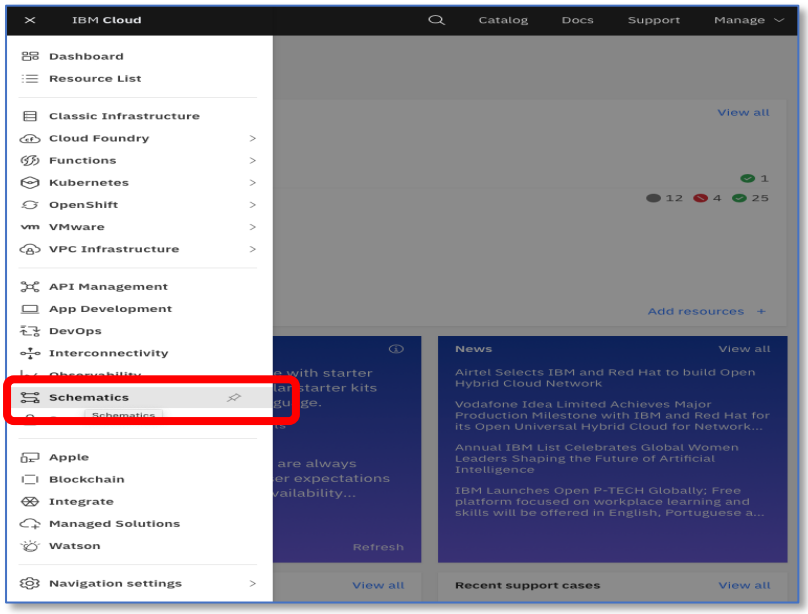
You can find the home page by clicking in 'Offering dashboard' from your workspace in 'Schematics'



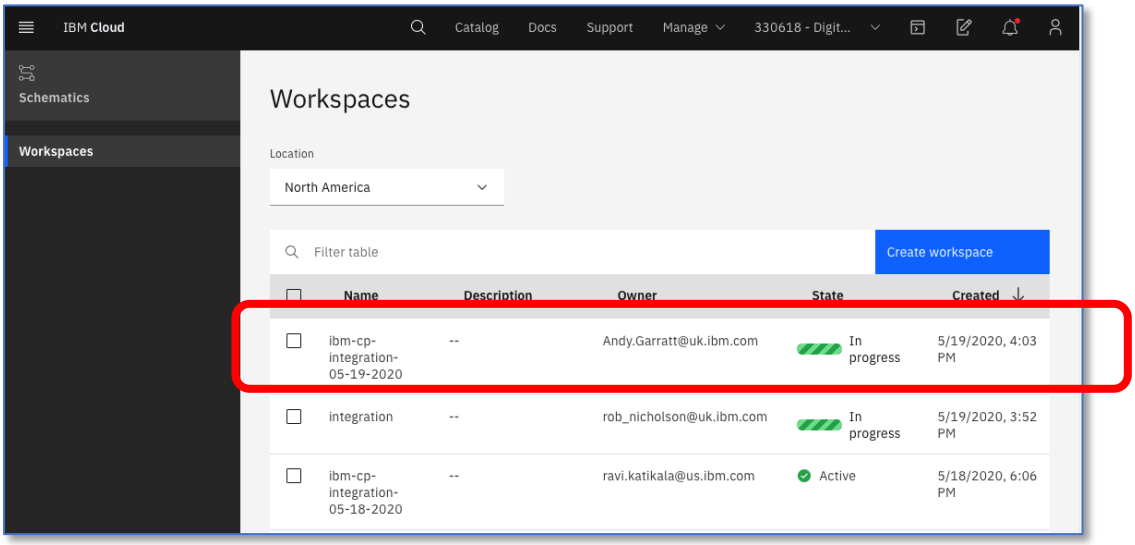
(To Navigate to the Offering Dashboard from the IBM cloud , do the following:
Go to the 'Account' pull down and make sure you're in the 'DTE' account: If you're not pull down and click on the DTE account to change it.



Next, click on the ‘Hamburger’ menu and click ‘Schematics’



Then Click your install (look for your email) and you’ll see the offering dashboard.



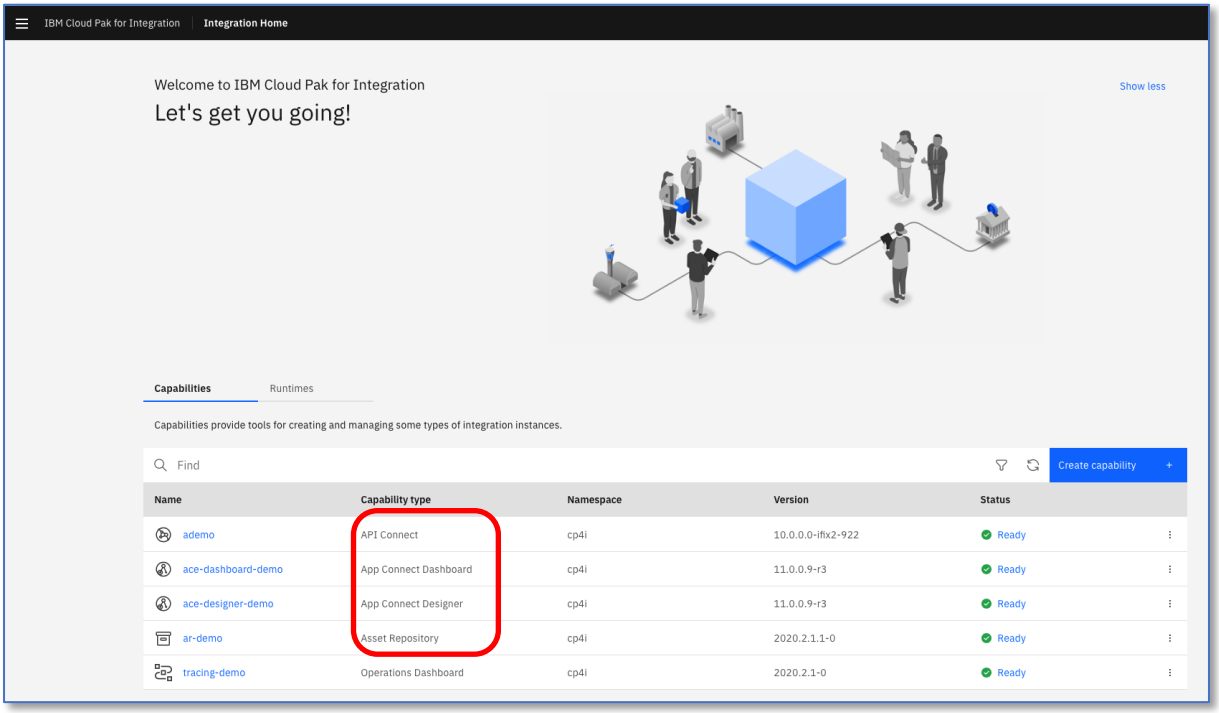
Sign in to the Cloud Pak for Integration

The username will be “`admin`”
The password will be the (very long) one you entered during the 1-click install earlier.

Enter your credentials and click ‘Log in’

Welcome to CP4I! You're now at the home screen showing all the capabilities of the pak, brought together in one place.

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

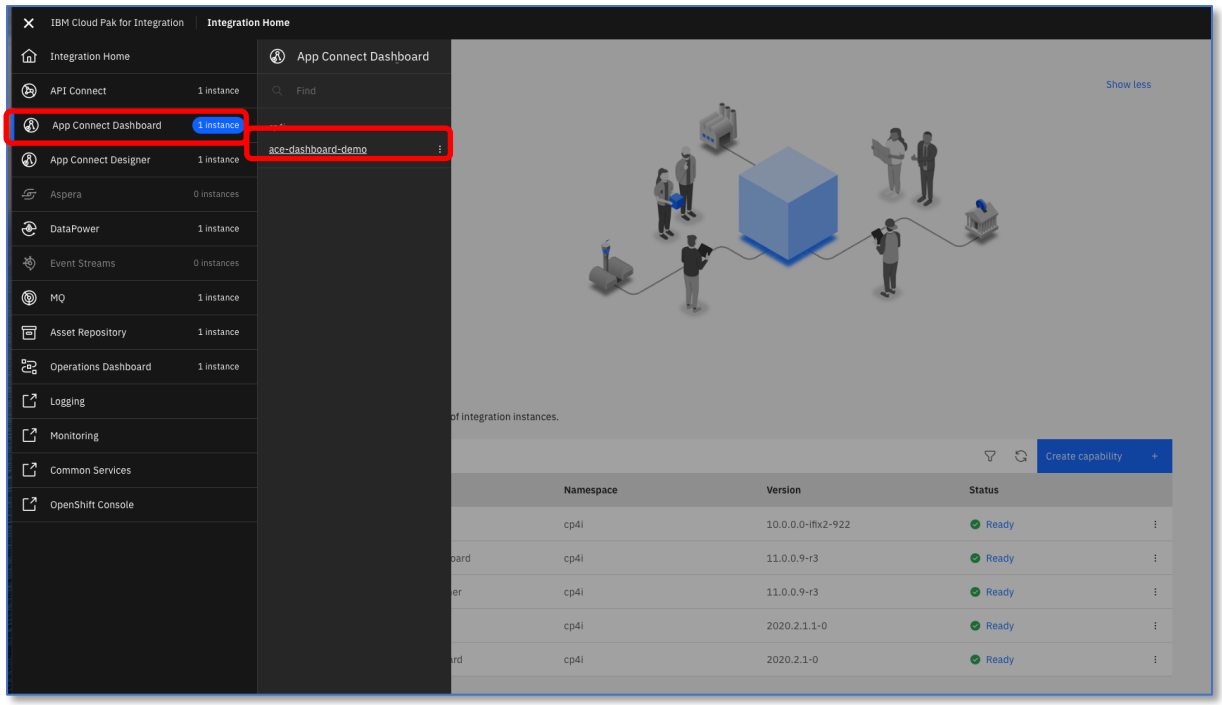


We've already created the instances of the capabilities you need, so you don't need to create new instances:

IMPORTANT: Don't worry if your installation has more capabilities than listed here e.g. MQ, Event Streams, Tracing etc. We are adding more capabilities on an ongoing basis so that we can give you more demos. Ignore the extras for the moment!

You can see that we have App Connect Designer (Tooling for building integrations), the App Connect Dashboard (this is what manages the integration runtimes) and API Connect (for managing APIs). You can also see the Asset Repository where we will store and share re-usable artefacts.

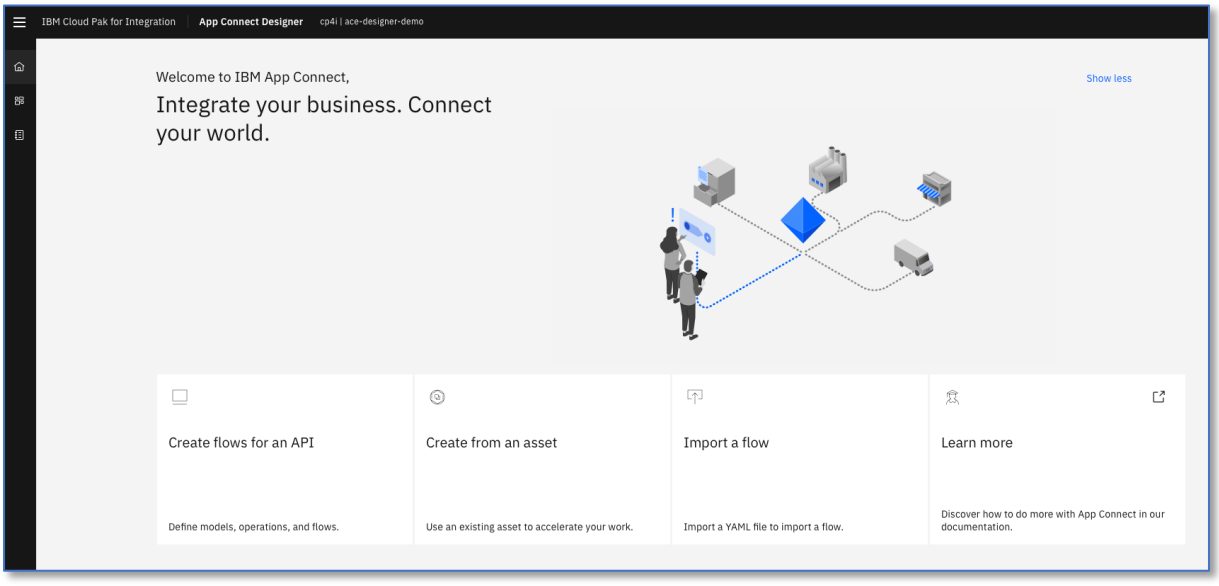
At any time, we can use the menu to navigate between these capabilities, as well as using the platform home screen. Use the 'hamburger' menu at the top left to access the menu and choose the capability you want like below:



9.2 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.

At the moment, there’s nothing here yet, so let’s build some integration logic.

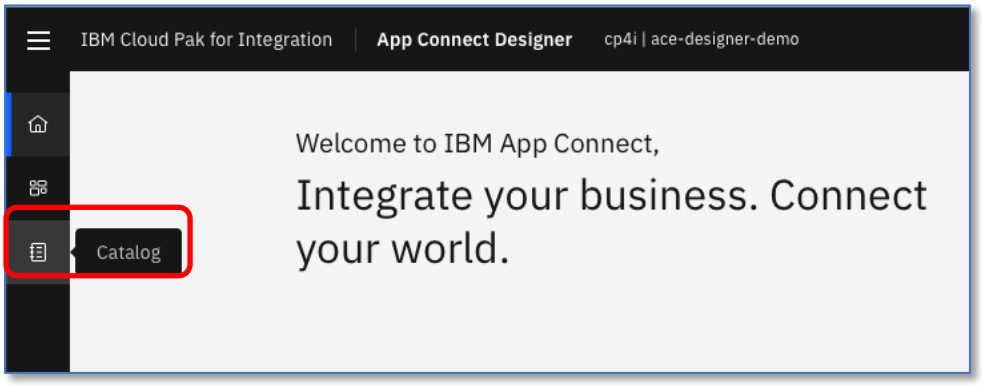
First, we’re going to connect the designer tooling to our endpoints that we set up earlier.

The Smart Connectors are meta-data driven, so they need to be able to connect whilst we’re using the tooling to ensure that they show us the correct data and functions available from our endpoints.

To connect to our endpoints, we’re going to need the credentials we obtained earlier when we created our endpoints.

9.3 Connecting the tooling to our endpoints

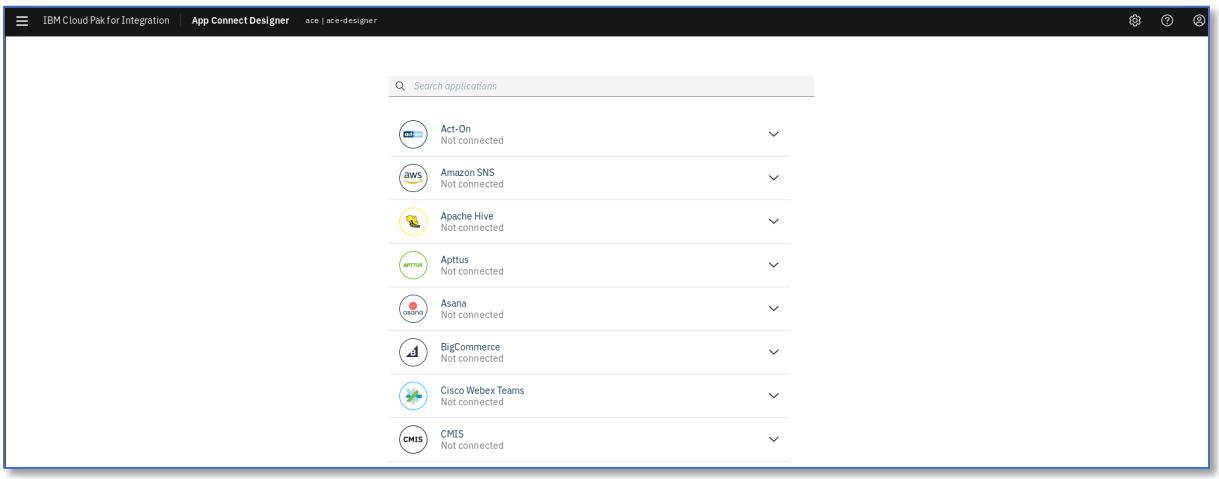
Let's go to the connector catalog: click on the catalog icon on the leftand click 'Catalog'



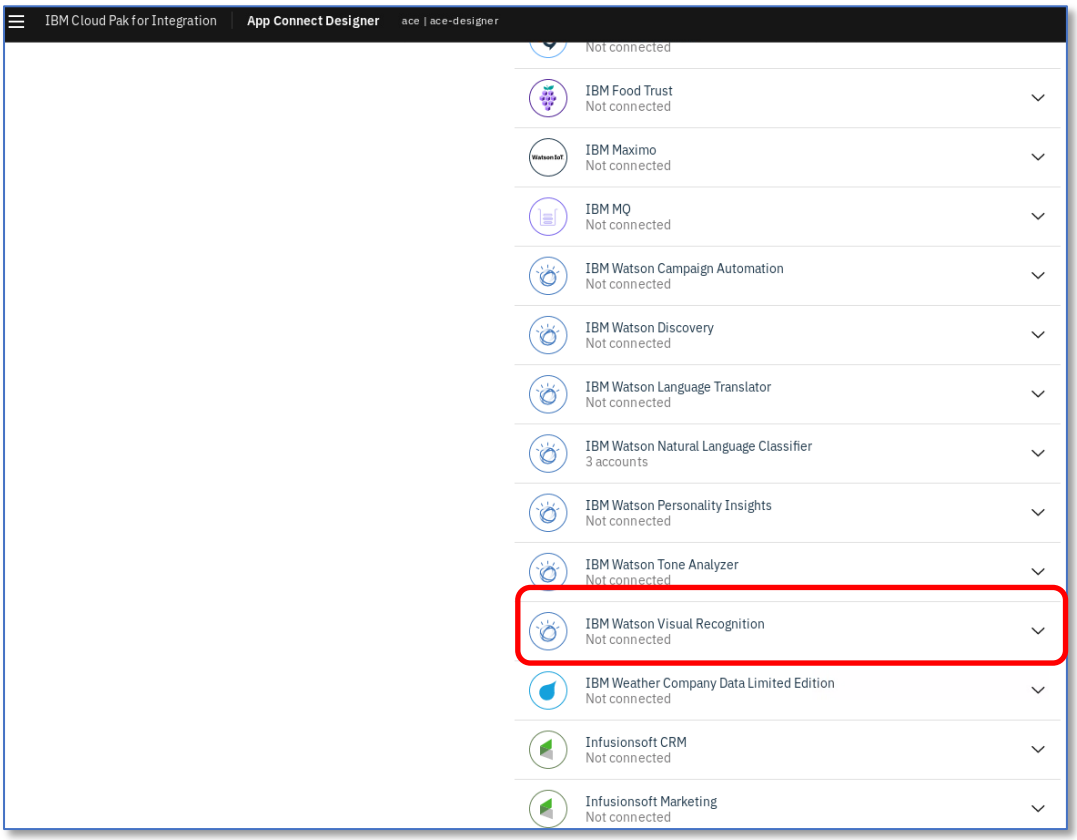
The connector catalog appears with a list of the cloud pak connectors which are installed locally to this lab. There are many connectors available although not all of them run 'locally'. Some of the connectors are currently available in the pak locally, all of them are available on the IBM cloud – you can use the ones that run on the IBM cloud directly from CP4I designer as well – you just need to link CP4I 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:

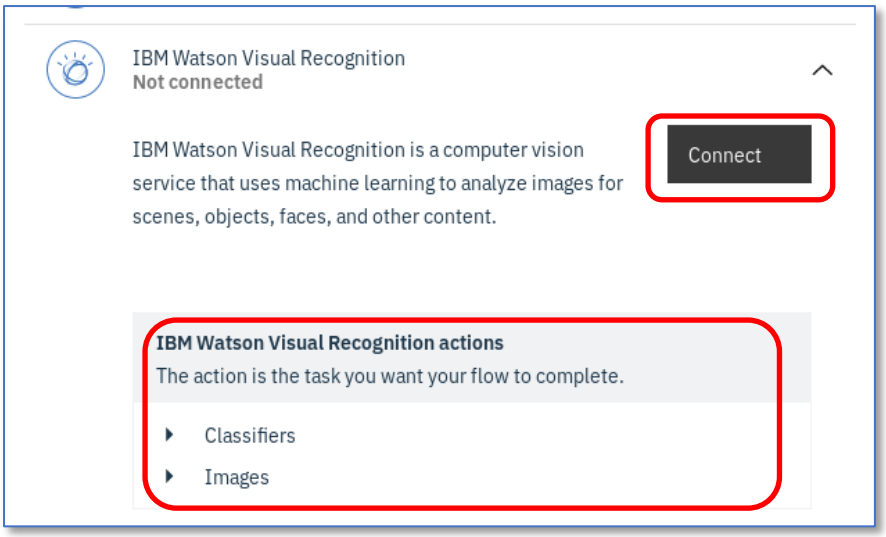


Let’s set up our Watson AI endpoints – scroll down until you see the IBM Watson connectors:



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.

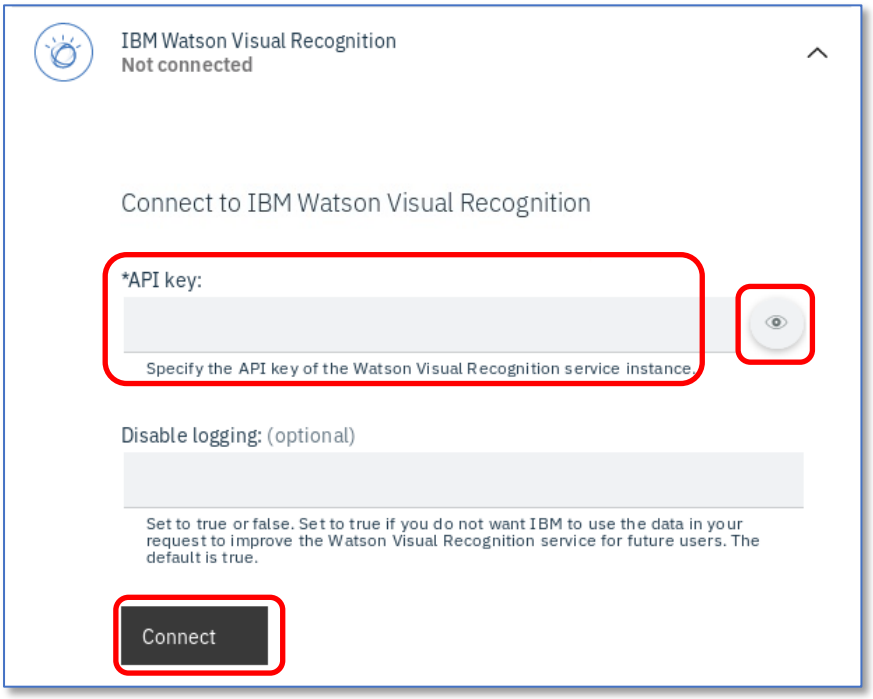


Click on ‘Connect’

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’.

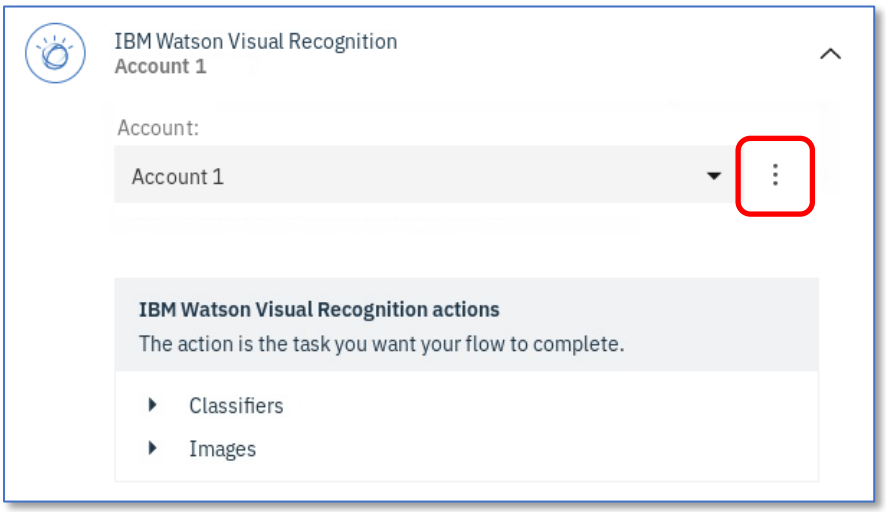
Make sure you have the right one – the one for e.g. Tone Analyzer will not work for Visual Recognition.



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

If you’ve ‘forgotten’ your API key, go back to the service you created in the IBM Cloud – you can view it from there.

If all goes well, (i.e. you've entered your key correctly), a connector account will be created for you – that's it! You've added image recognition capability to your integration

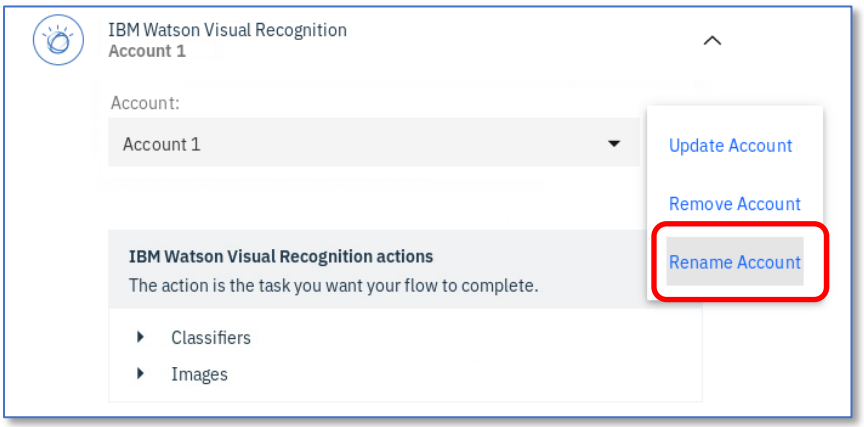


IMPORTANT: DON'T MOVE ON YET! You'll see 'Account 1' as the name of the account.

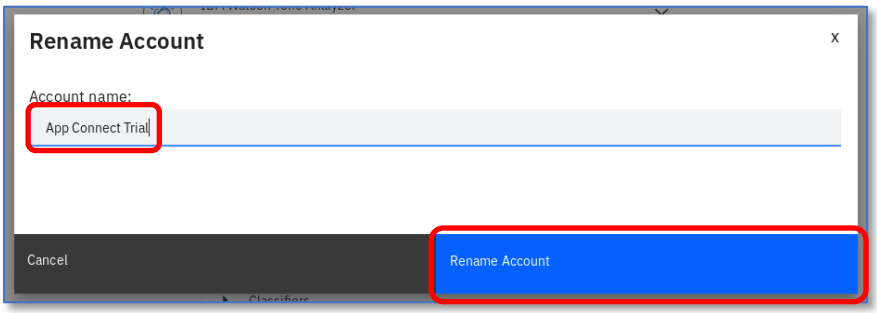
WE NEED TO RENAME THE ACCOUNT FOR THE LAB TO WORK SEAMLESSLY (we'll tell you how to fix it if you don't later....but it's easier if you do!)

CP4I lets you have multiple accounts for connecting to each type of system. For example you could have a DEV account, a TEST account and a PROD account. Or you may have a USA instance and an EU instance. The name is what the integrations use to reference the correct account. You can connect your connectors to as many places as you wish – there's no extra charge – all connectors are included.

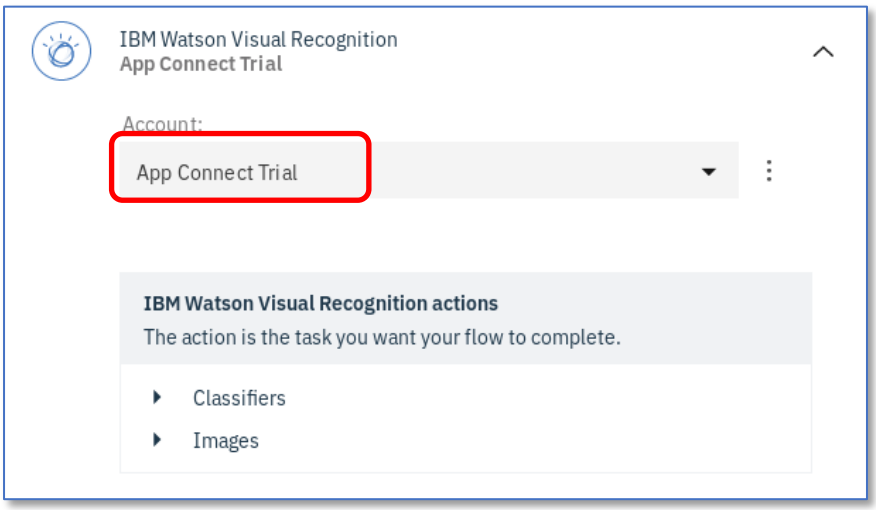
To rename your account, Click the three dots menu and click 'rename account'



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,

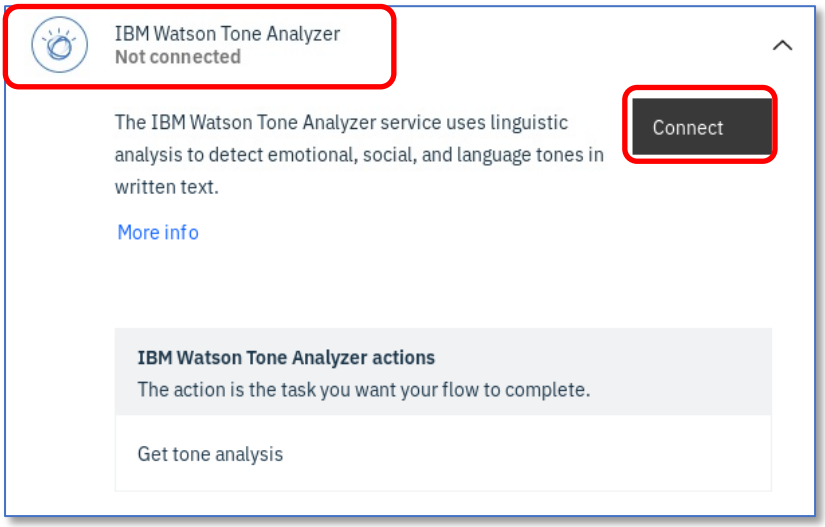


Your connector should now look like this:



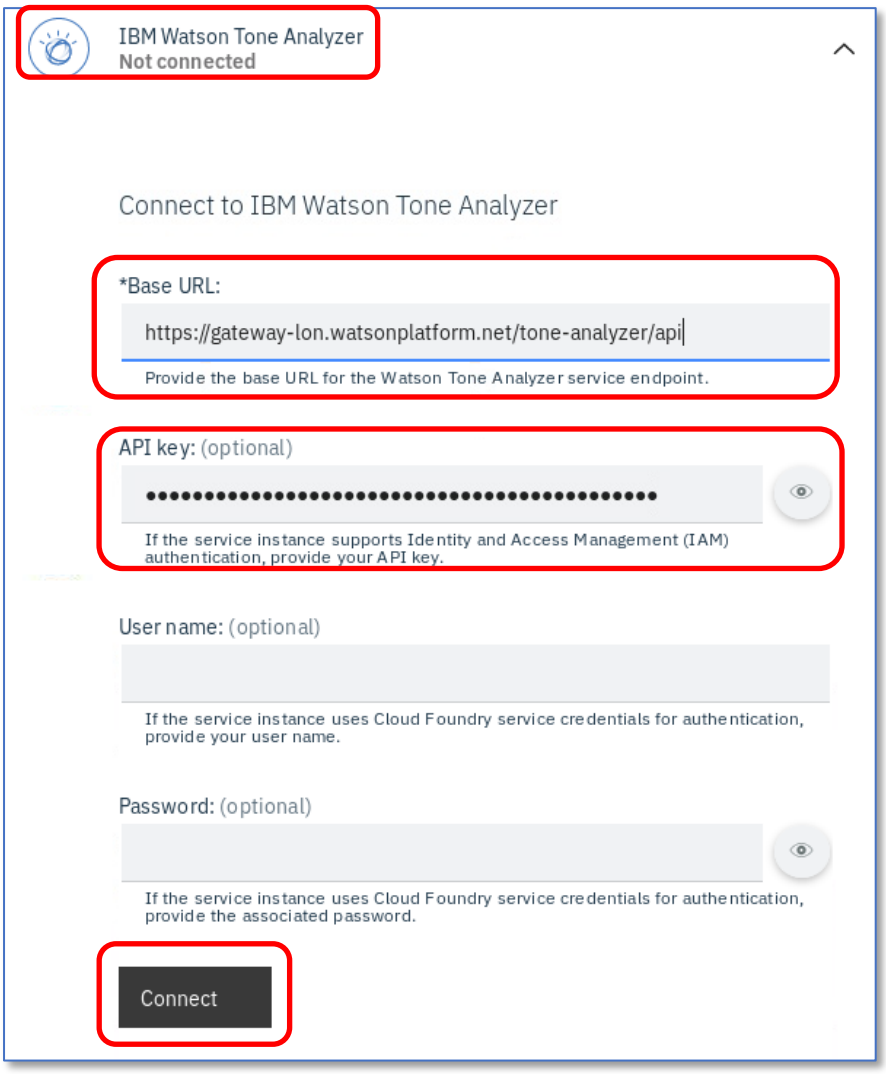
OK, we have our Visual Recognition sorted – let's do the next two Watson connectors:

Click on 'IBM Watson Tone Analyzer' and click 'Connect'



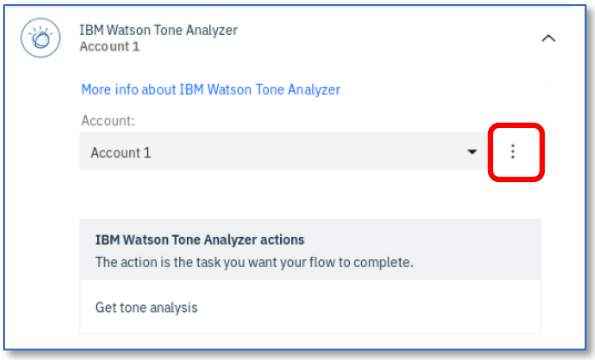
For this connector, we'll need the URL and the API key that we got earlier: Enter them in the dialog below – (you won't need the User name and Password).

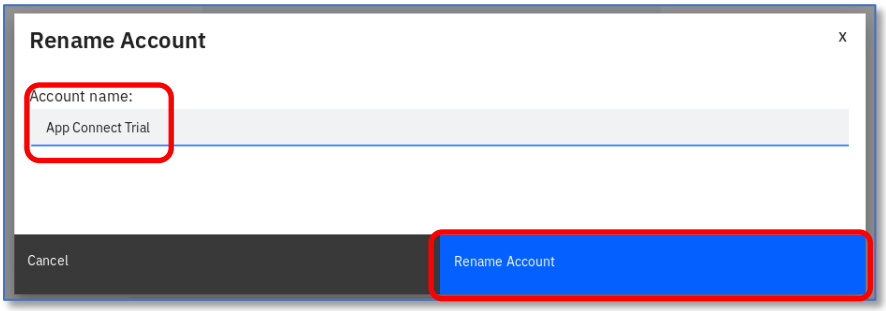
Note: Your URL may be different to our screenshot – it depends in which cloud region your service is running. Click 'Connect'



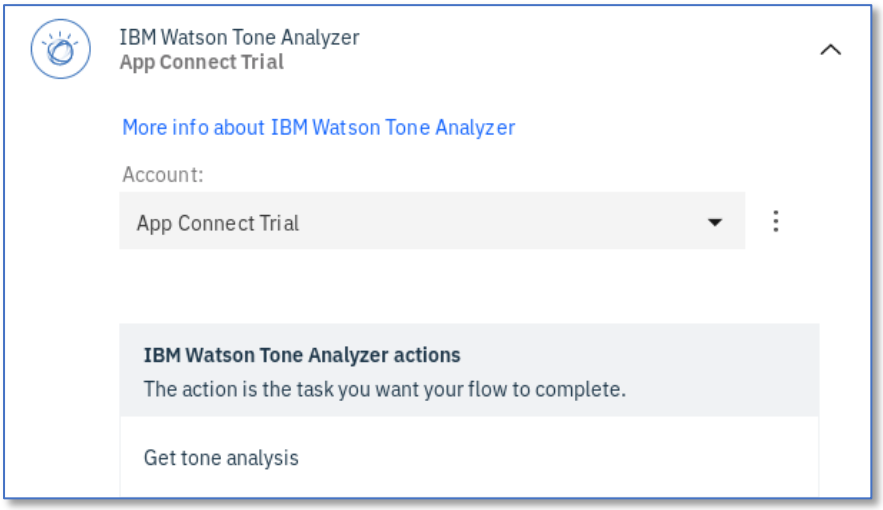
And we're connected!

IMPORTANT – FOR THE LAB, RENAME THE ACCOUNT to '`App Connect Trial`'. (use the three dots menu and click 'Rename Account')

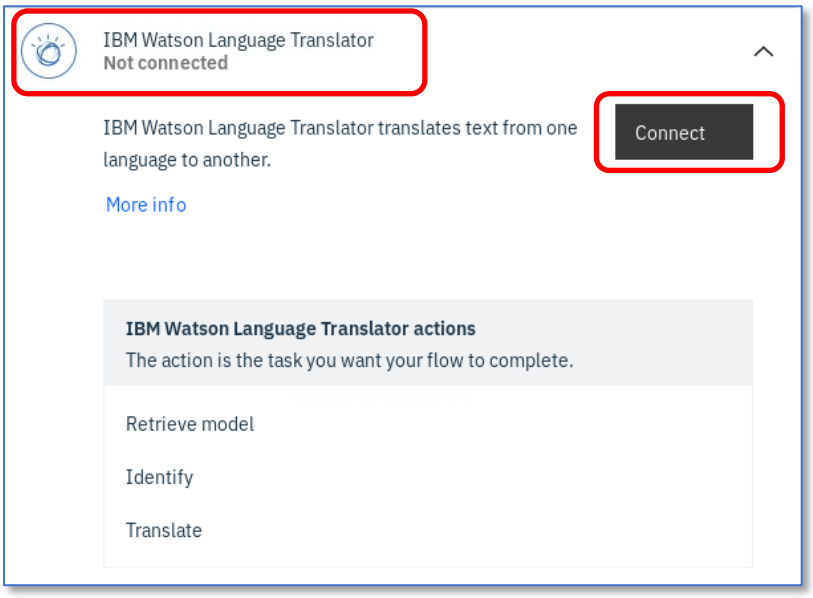




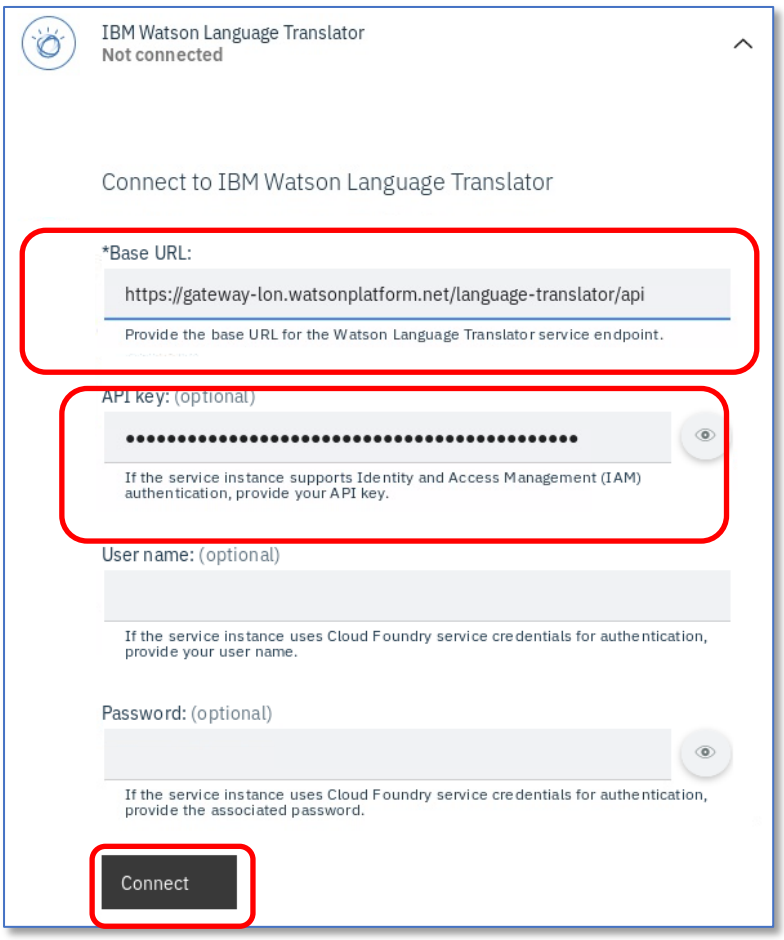
Your connector should look like this:



Finally, let's connect to the Watson Language Translator – it's very similar:



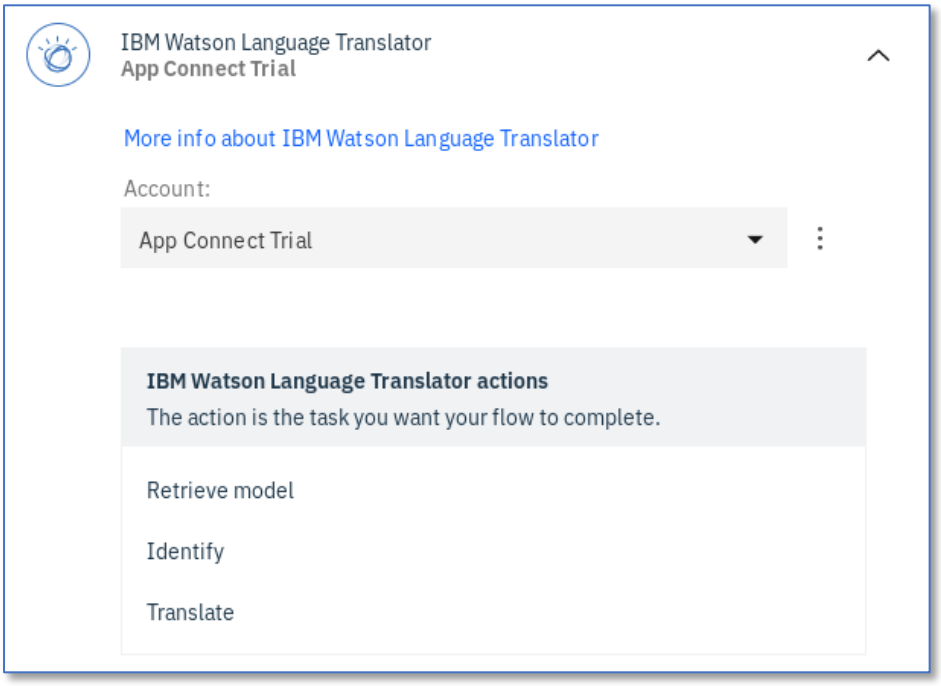
Click 'Connect' then enter your URL and API key (note your URL may be different from our screenshot depending on the region of your Watson services)



Click 'Connect'

IMPORTANT – FOR THE LAB, RENAME THE ACCOUNT to '`App Connect Trial`'. (use the three dots menu and click 'Rename Account')

It should look like this:



Why is it important to rename the accounts? We're going to import an integration flow to save you some typing and clicking. This flow is configured to look for connector accounts named '`App Connect Trial`'

If you don't rename your accounts, you'll need to edit the flow to point to the ones you've created and match the names. It's not hard to do, but it does add extra work.

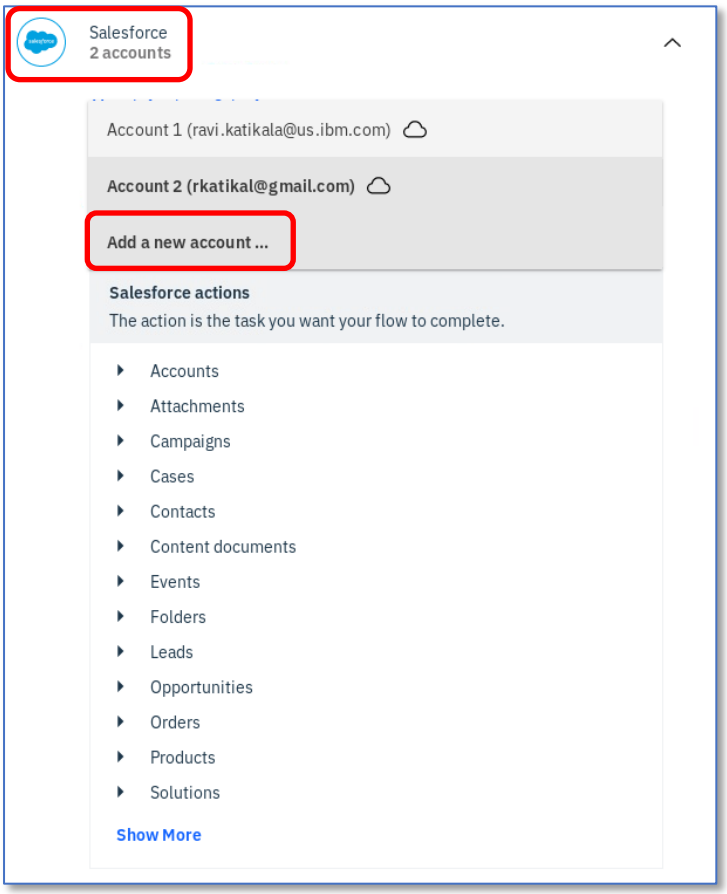
9.4 Setting up the Salesforce Connection

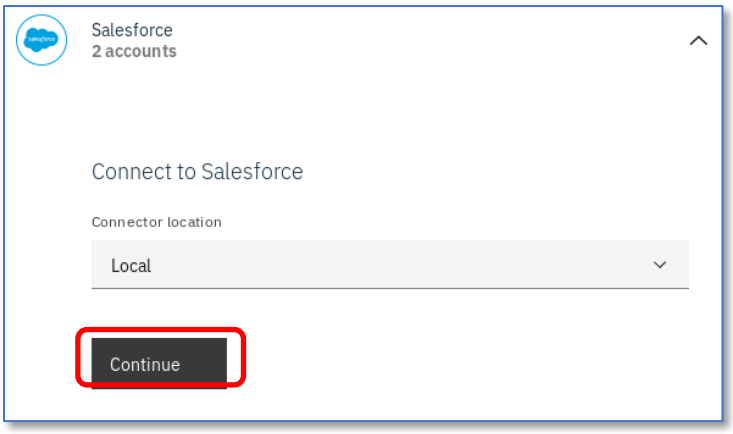
Just one more endpoint to go, then we can look at API flows.

Scroll down to the Salesforce connector. There may be multiple types of salesforce connector shown , pick the first one just called ‘Salesforce’.

(You may see there are already accounts created – we’ll be creating a new one to connect to your Salesforce account anyway – don’t use the existing accounts – you won’t be able to see where your integrations go..)

Click ‘Add a new account’ if there are existing accounts, or just click ‘Connect’ if this is the first one.

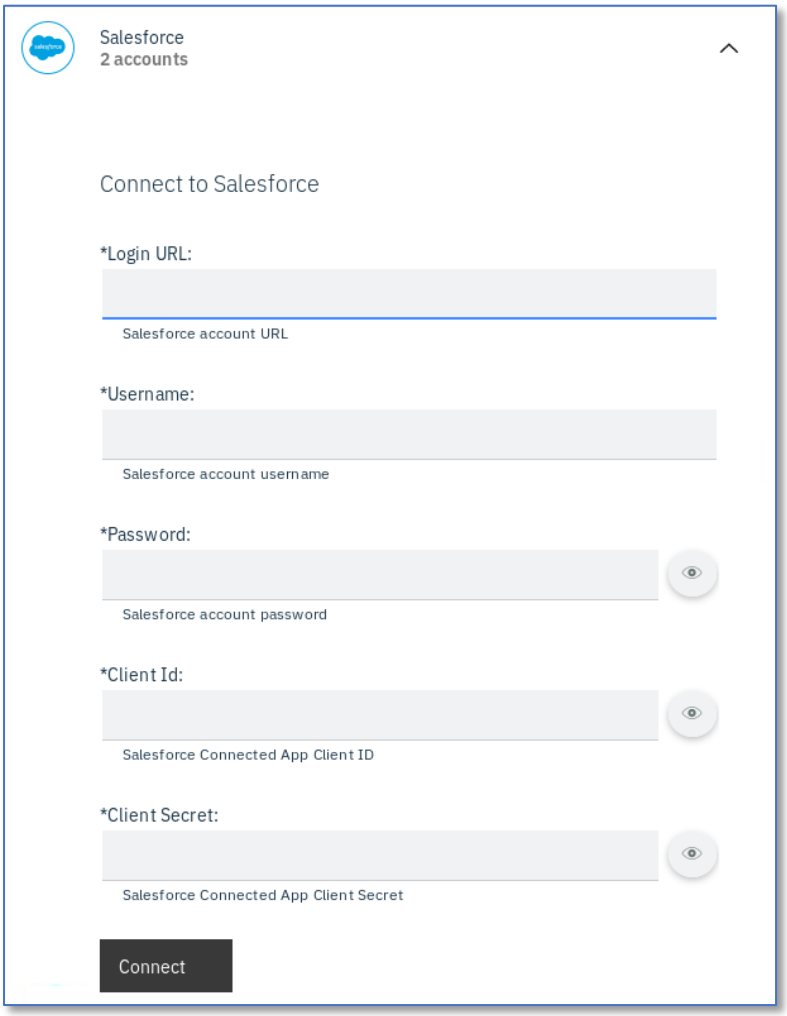




You'll now be asked for the Salesforce credentials – how do you get these? Follow the steps below.

Salesforce needs more than just your userid and password – it needs a client Id and Client Secret as well. Also, what you type in the 'Password' field in the connector isn't just your password that you log in with.

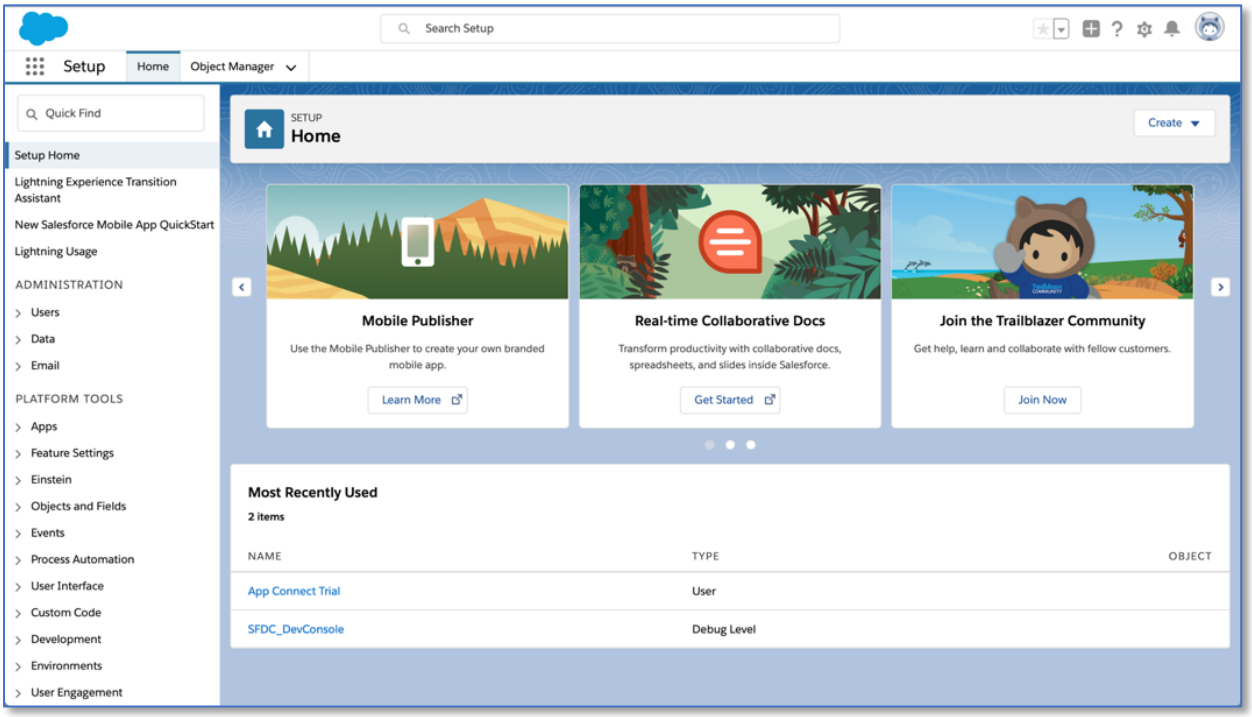
The fields we need are shown below



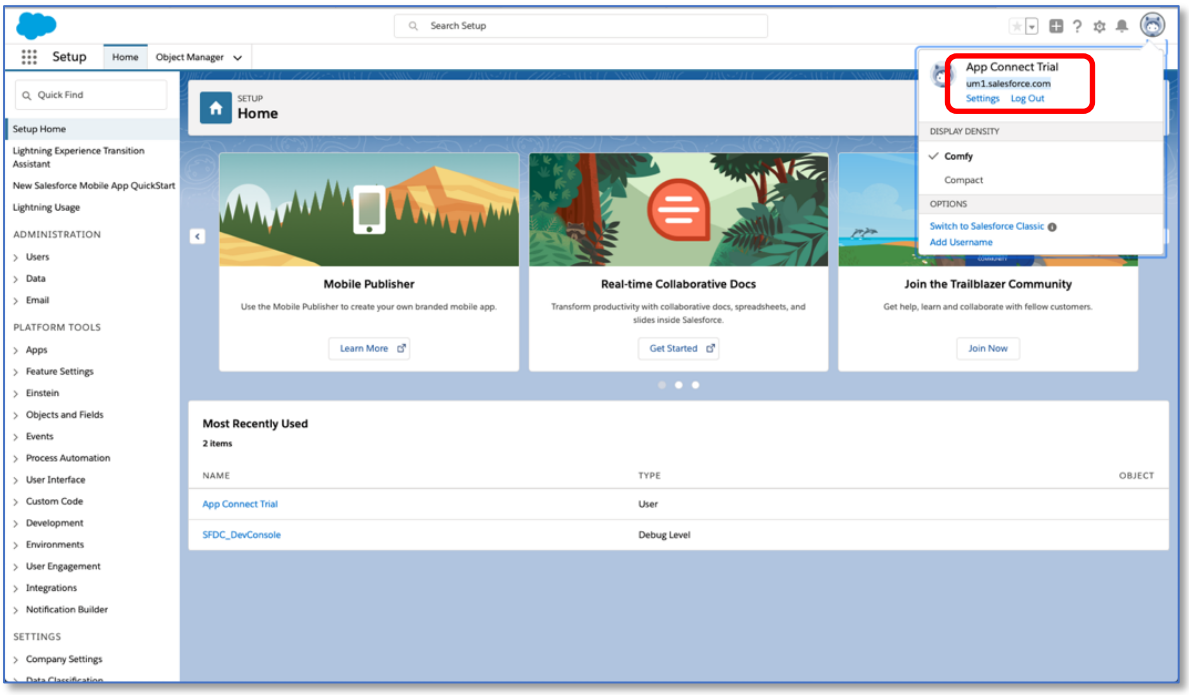
You will require admin level access to your Salesforce account.

When you created a free Salesforce account to test, make sure that you created a Developer account rather than a Trial account. If you connect to App Connect with a “Free Trial” account, the Salesforce integrations may not work.

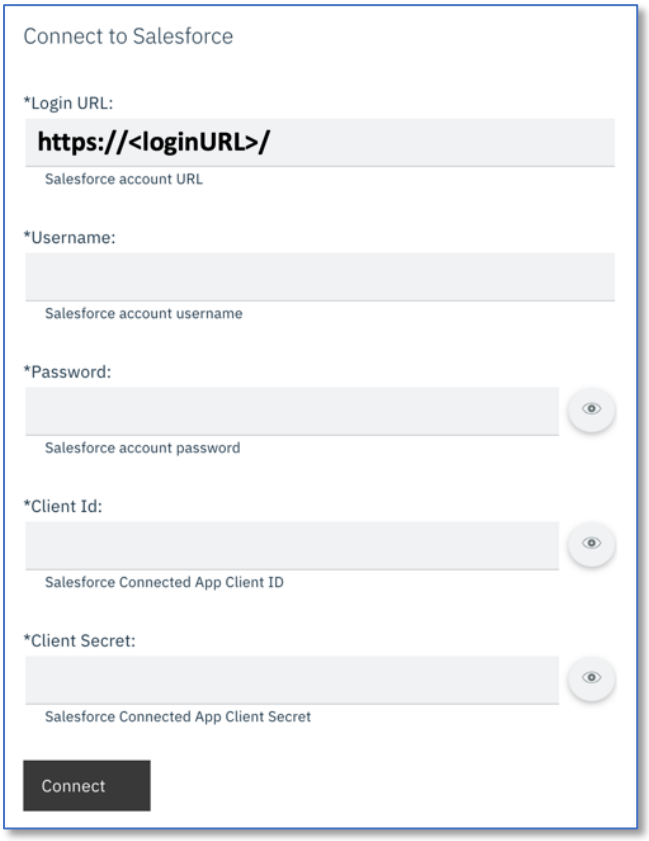
Login to your Salesforce Developer account – you should see the screen like below:



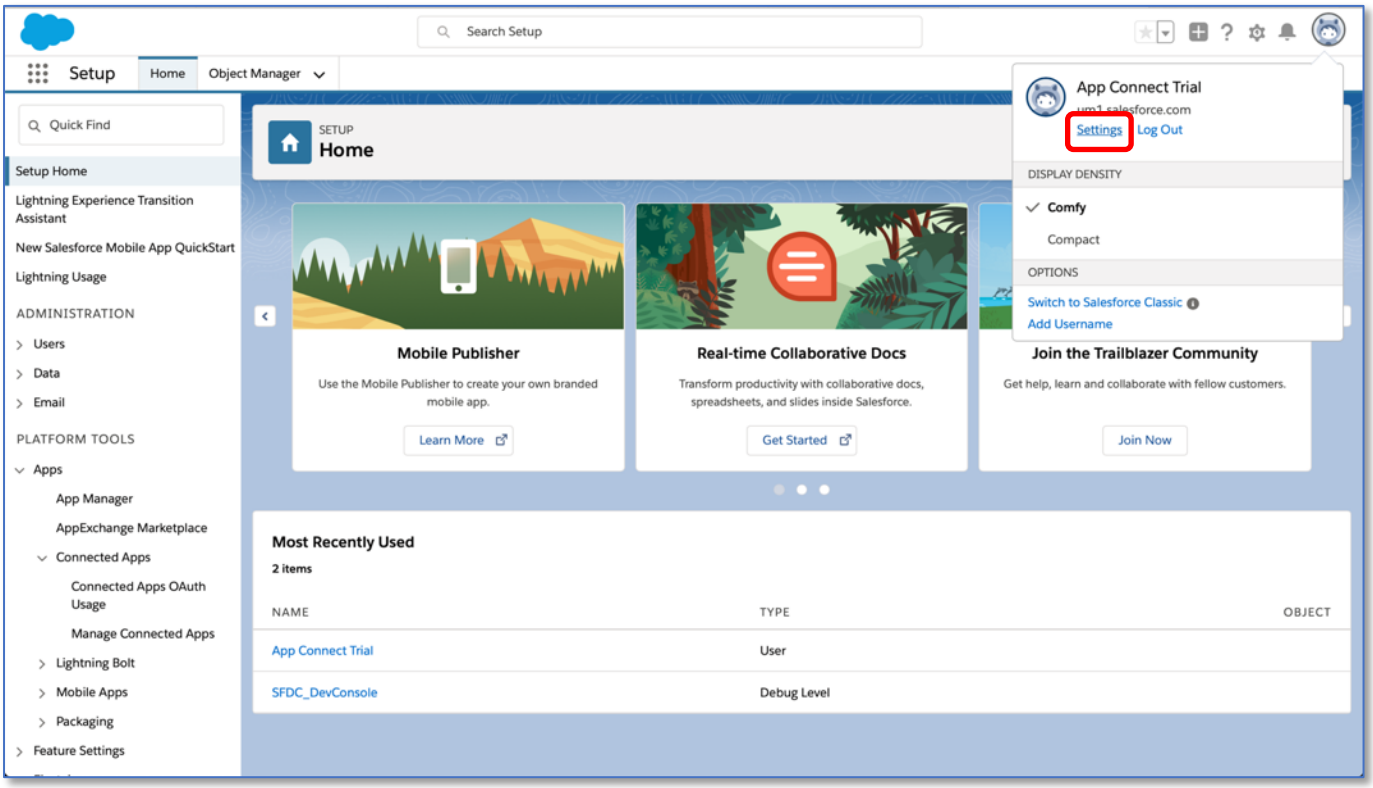
To get your **loginURL**, click on your user profile. The URL text below your Account Name is your login URL – BUT WITHOUT THE LEADING HTTPS:// .



Insert the **login URL** into the connector account form as shown below:
IMPORTANT: You MUST enter the '`https://`' part as well – it won't work if you just copy/paste from the salesforce screen e.g. "`um1.salesforce.com`" will **not** work. "`https://um1.salesforce.com`" will!



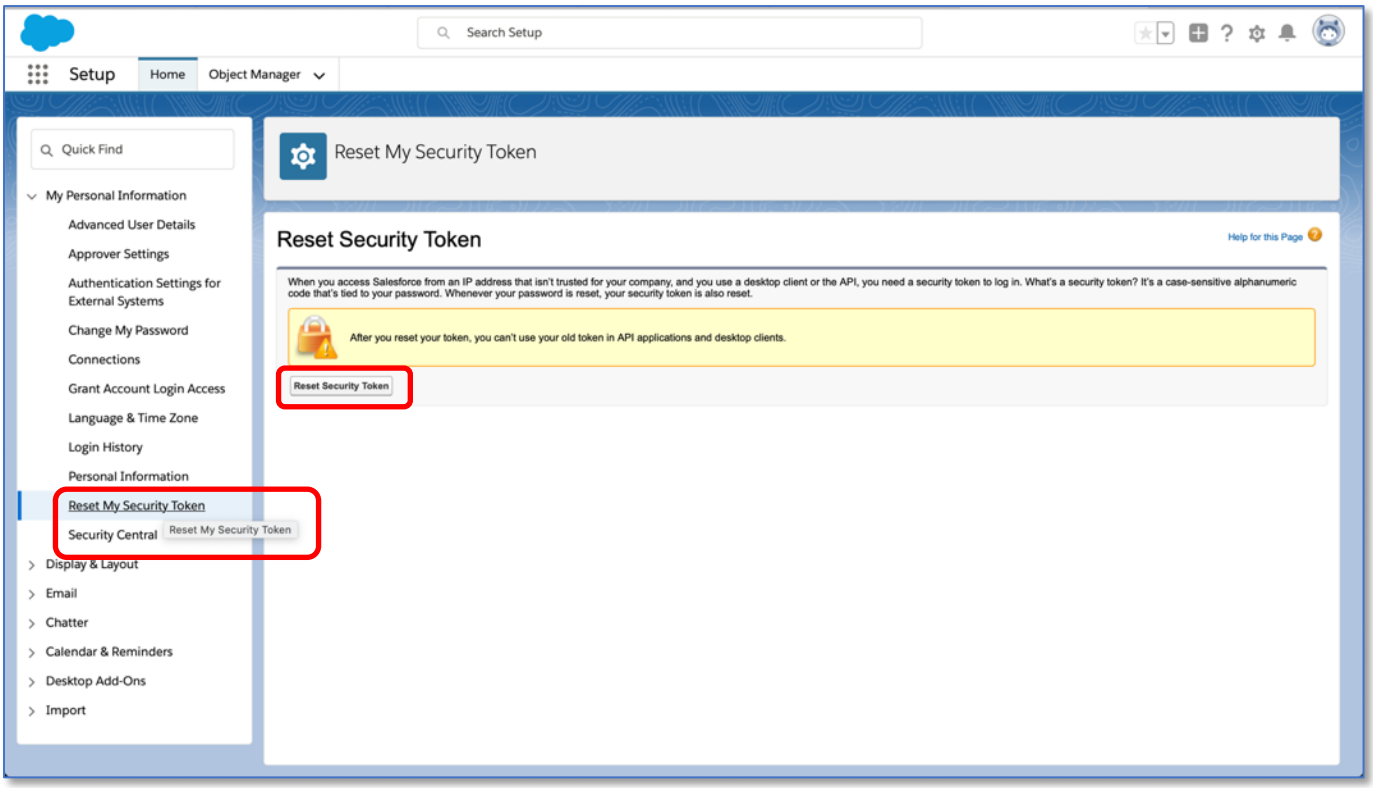
Next we will need to **retrieve Security Token**. For this click on your user profile and select the Settings option in the profile panel.



Under Settings, find and click the “Reset Security Token” option

(you may need to go to ‘Switch to lightning experience’ to see this)

(On the top right if you see it)

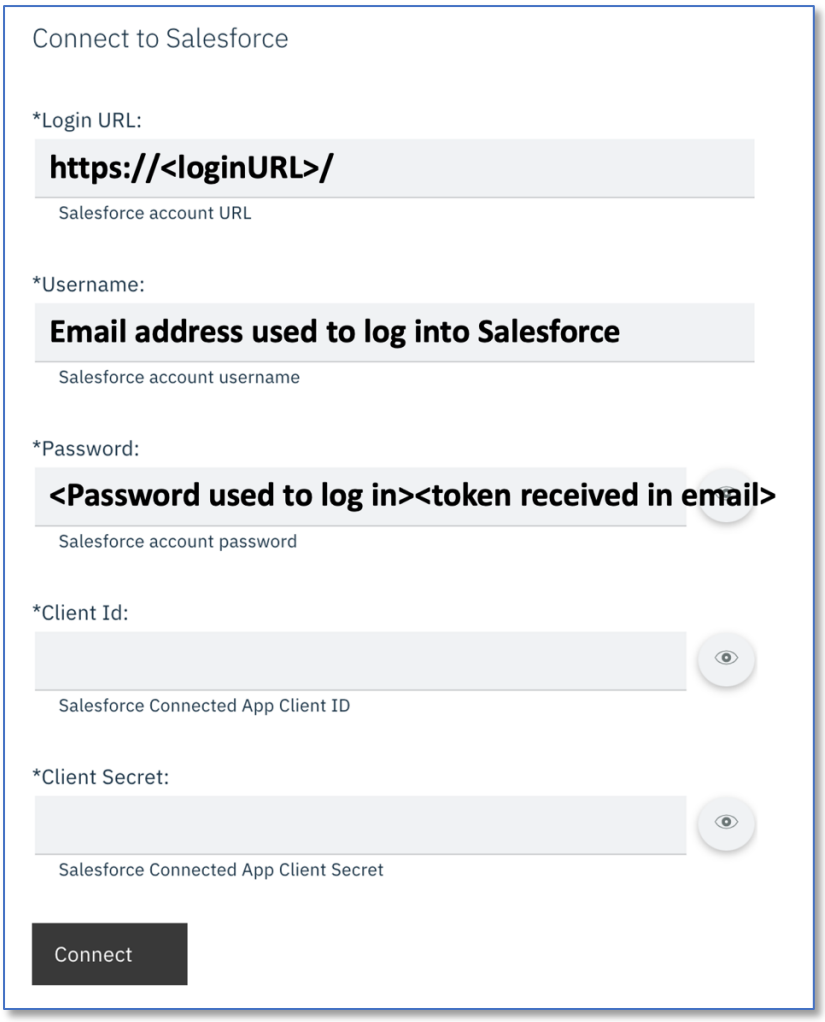


Click on Reset Security Token Button and it will send the **newly generated security token to your admin email address**. Use the token for your credentials.

To populate the Password field on the connector account screen you will need to *concatenate the Password used to log into the Salesforce account with the Security Token received via above step* as shown below:

For example if you Salesforce password is '`myGreatPassword`' and your Salesforce security token is '`2325jsdhew4312hs534dh`' then you should enter

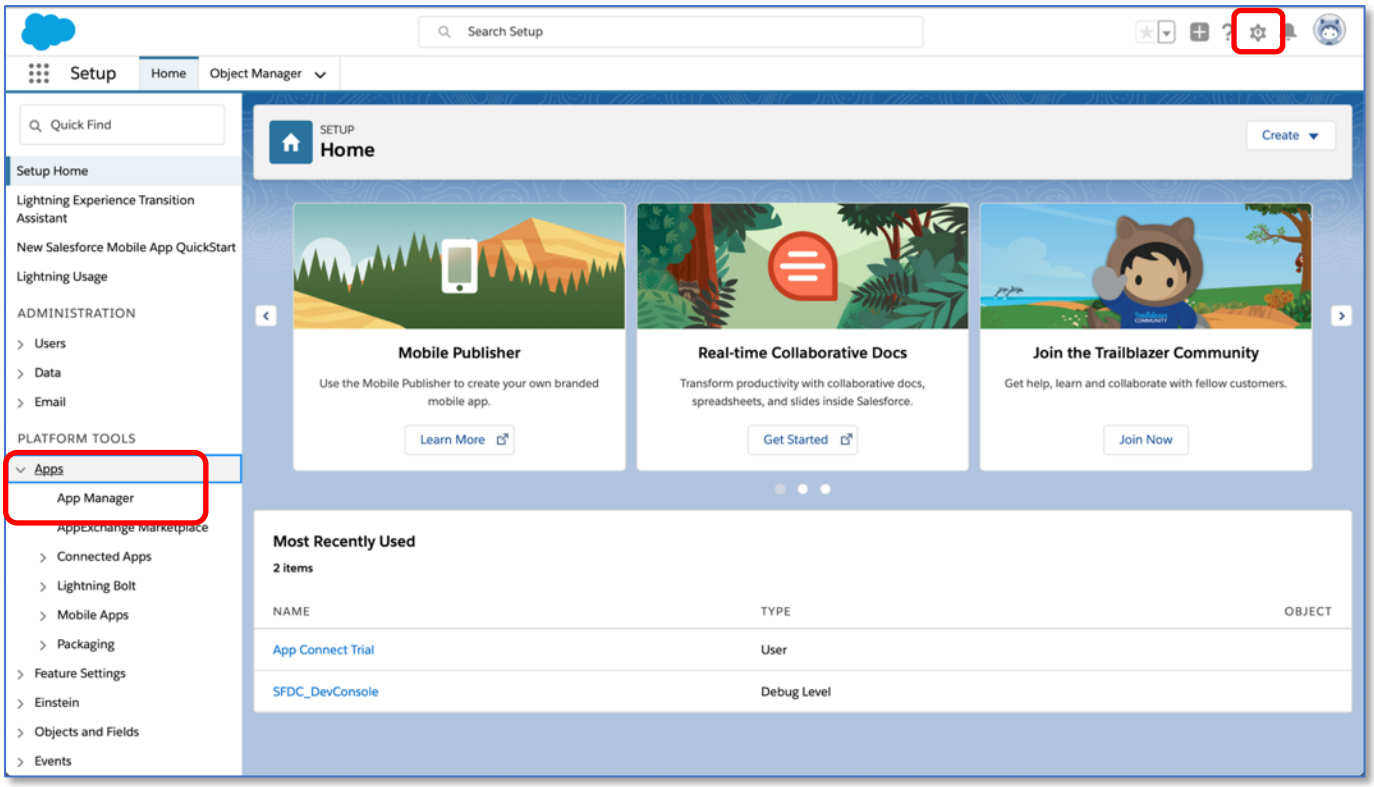
'`myGreatPassword2325jsdhew4312hs534dh`' in the 'password' field.



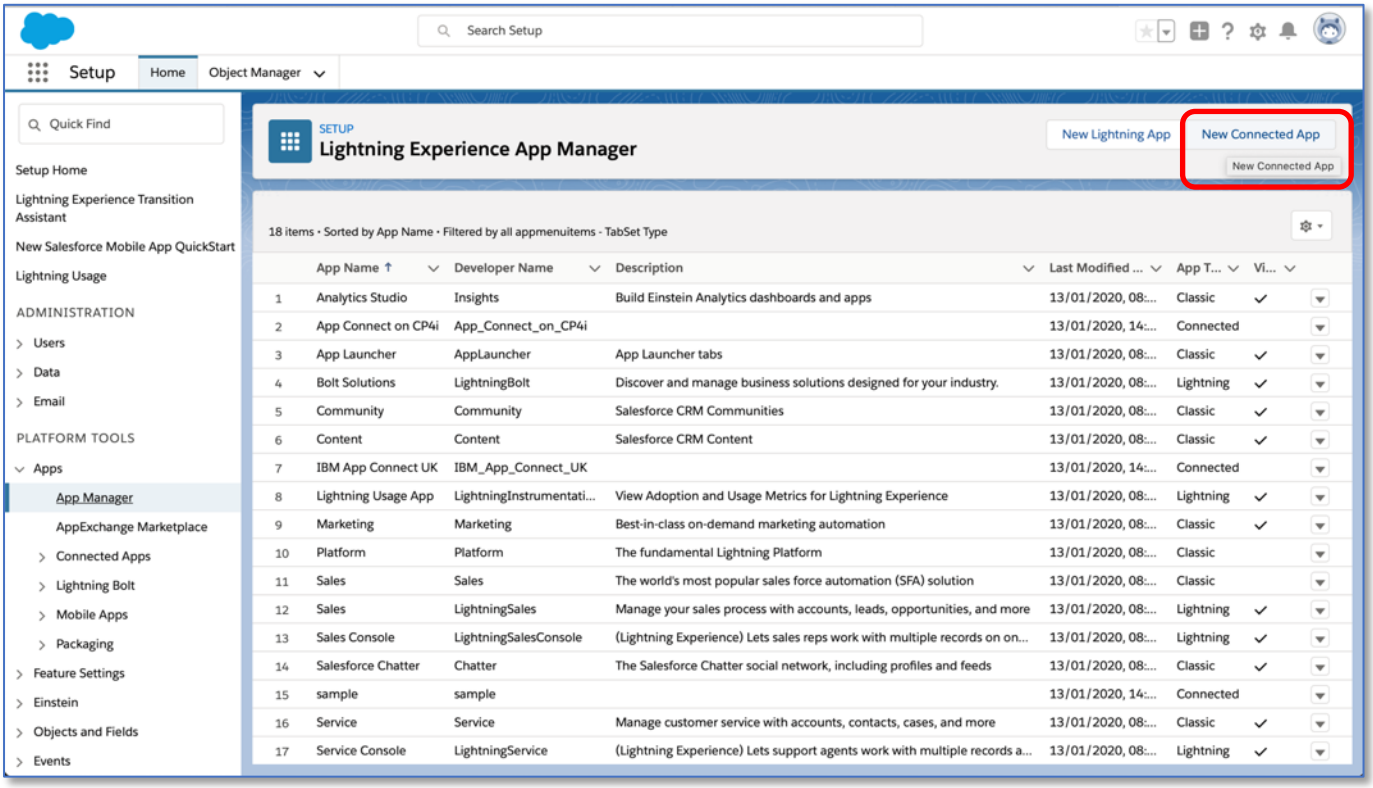
Next we will retrieve the **Client ID and Secret**

Click the 'setup' cogwheel at the top right.

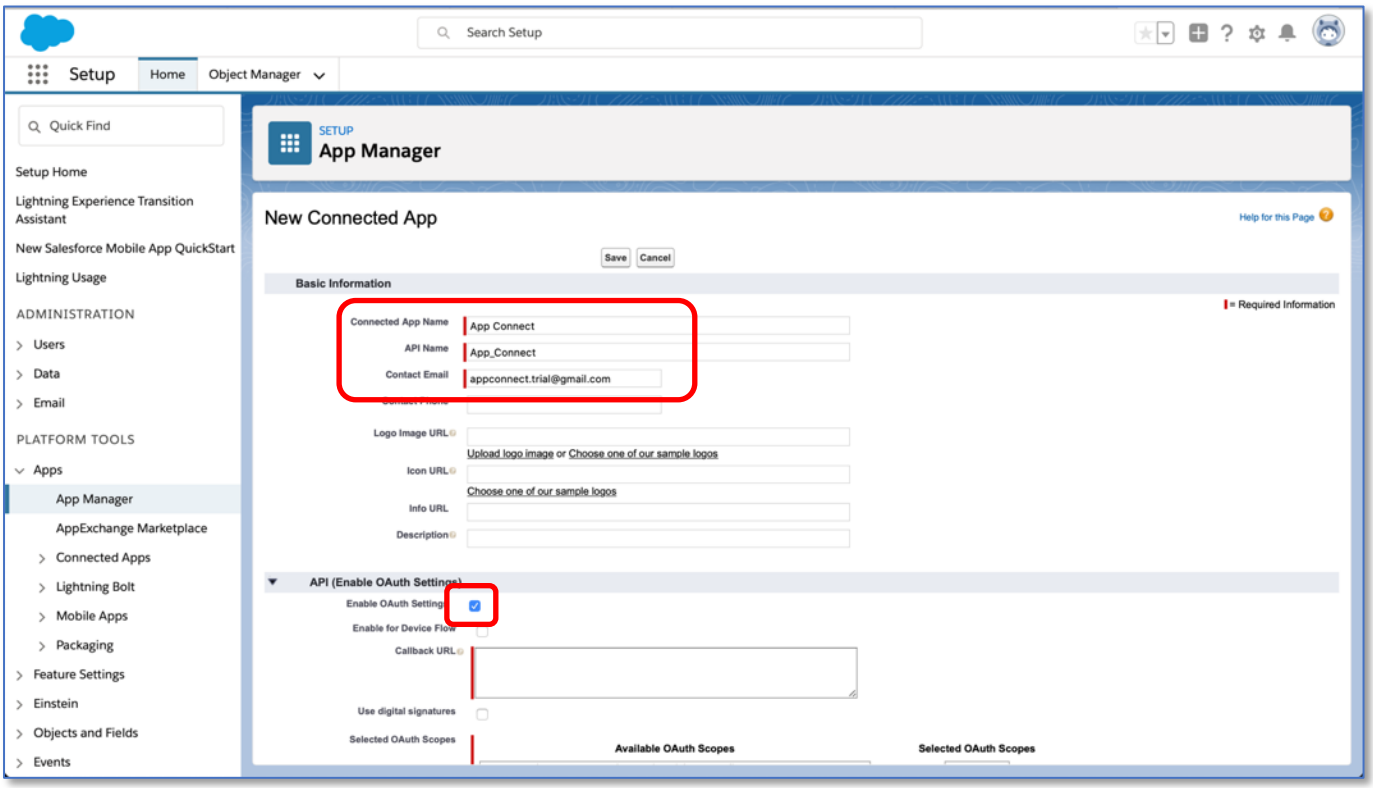
On the left-hand Finder panel go to:
PLATFORM TOOLS > Apps > App Manager



You then want to **create a New Connected App** or use an existing one. Steps for creating a new app are as follows:



Provide a Connect App Name and an API Name is automatically generated for you. Provide a Contact Email (usually admin email address). Please make sure you Enable OAuth Settings and follow steps below to configure the OAuth setting.



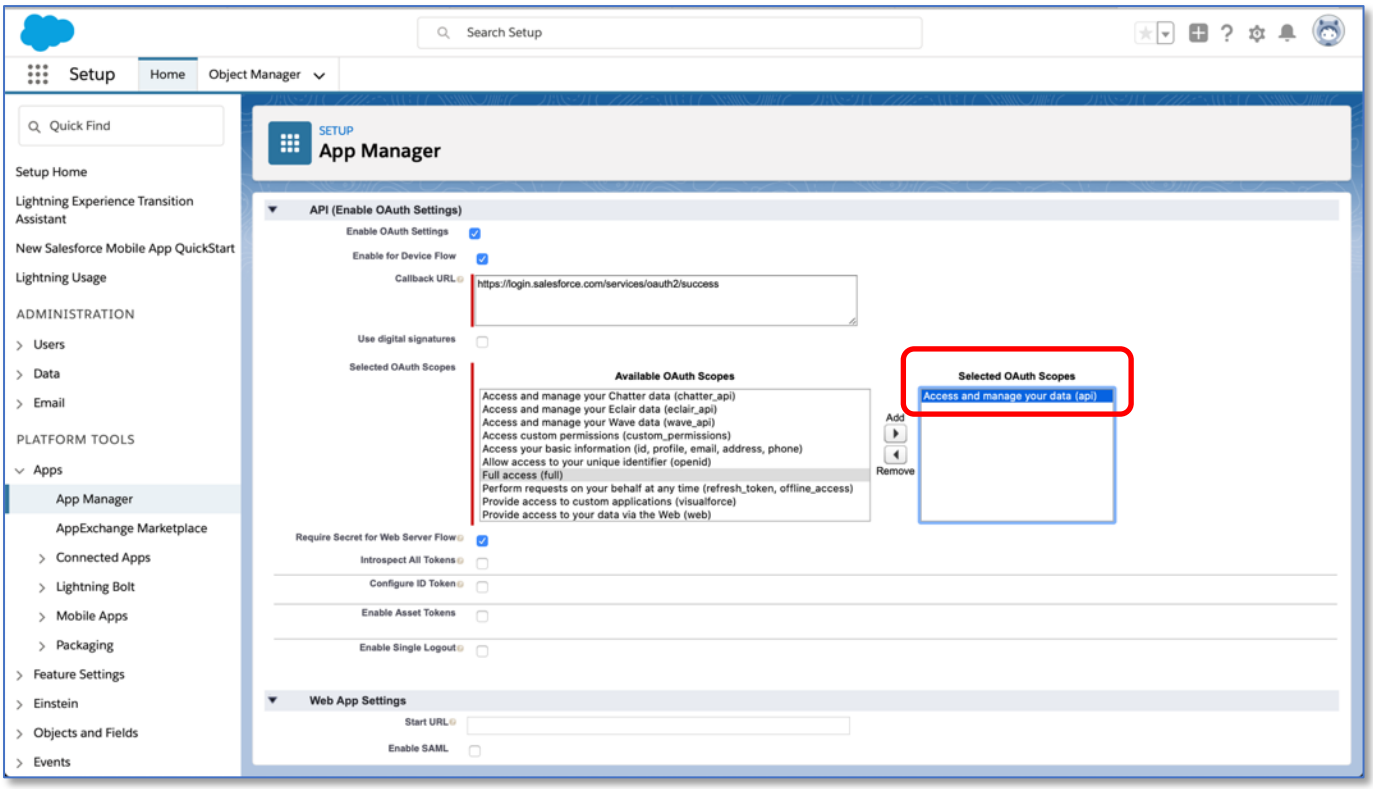
Click on Enable OAuth Settings to get the configuration panel.

Either click on Enable for Device Flow and that will auto-generate a Callback URL or alternately you can provide your own fully qualified Callback URL

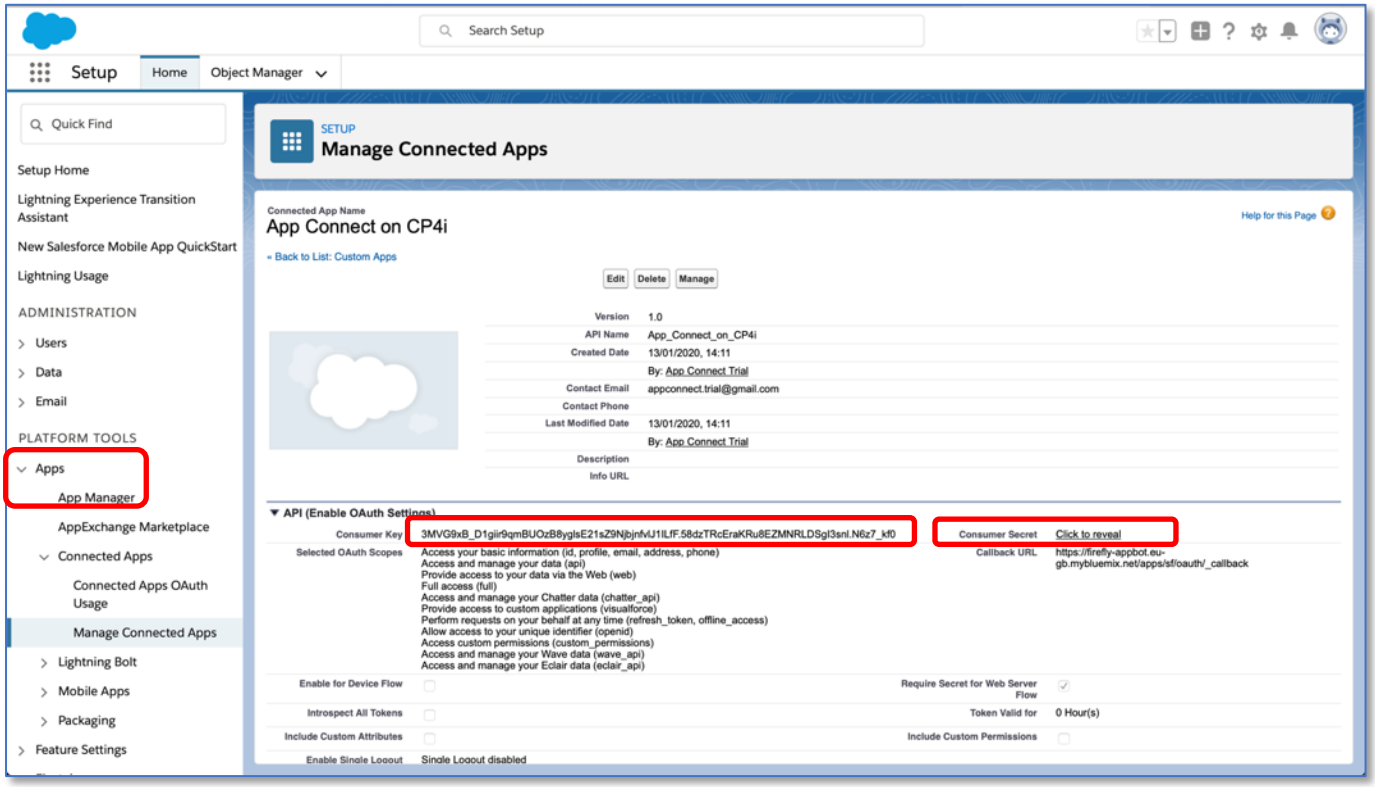
Next step is to configure the scope of access for our connectors which will be the Connected App in this case.

Connectors technically only require “data api” - you can optionally choose to enable all the scopes for this connected app.

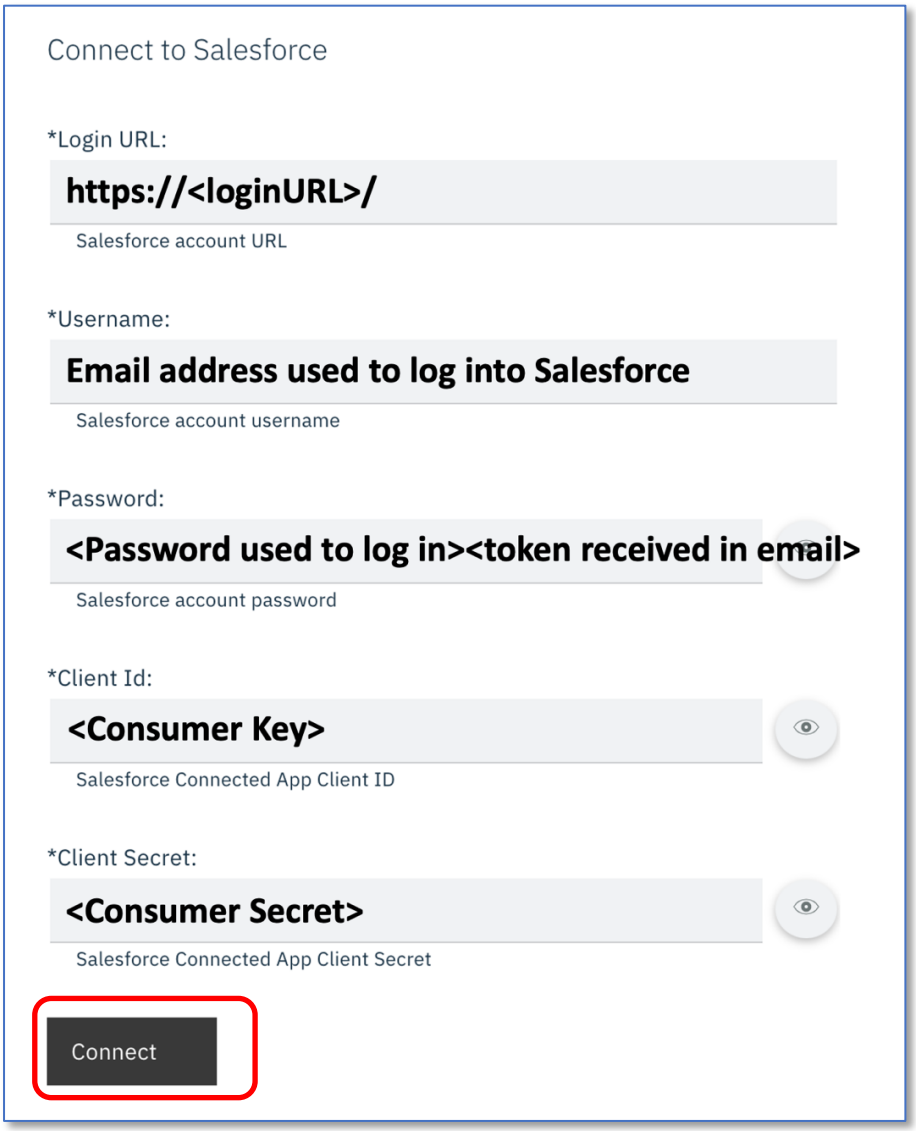
And then click on Save.



It may take several minutes for newly created Connected App to be registered.
Once registered go back to App Manager, select and view the created App

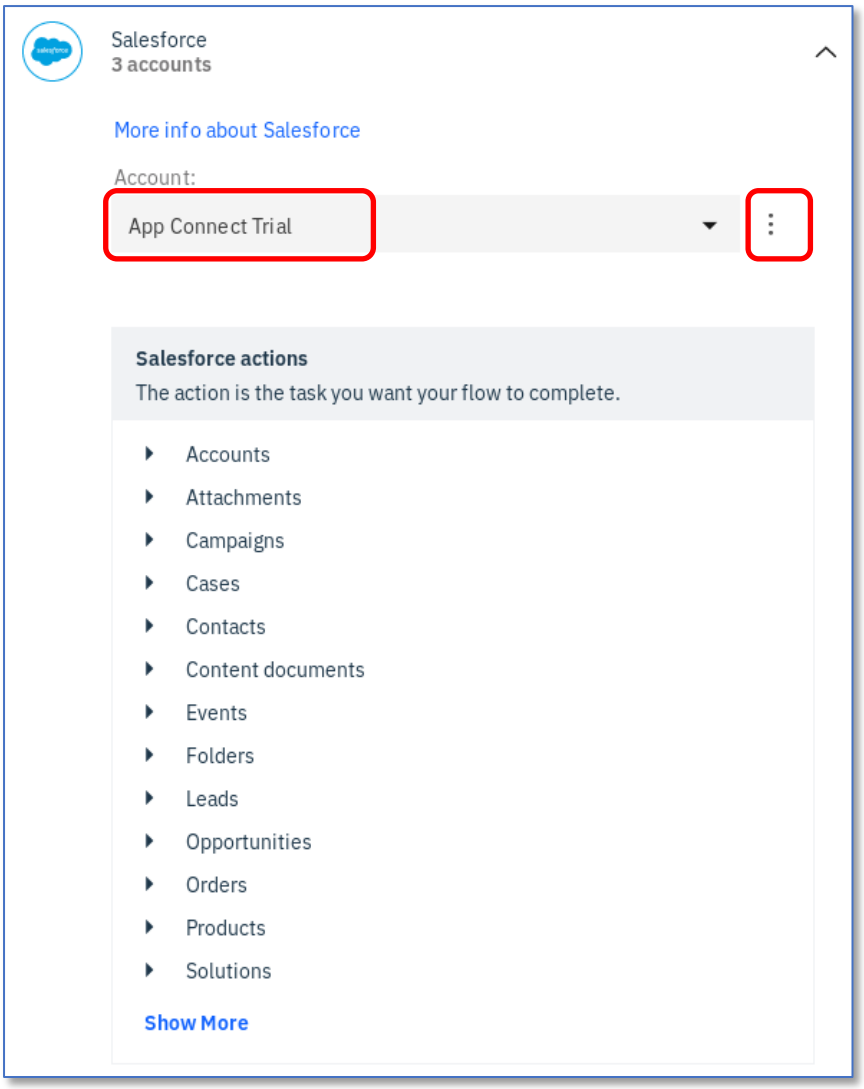


Use **Consumer Key and Secret as Client ID and Client Secret** respectively as needed in the connector account UI as follows:



Click Connect – you should see your account created!

IMPORTANT – After all that, we need to rename our account! Don't forget to use the three dots an rename our account to 'App Connect Trial' as shown below.

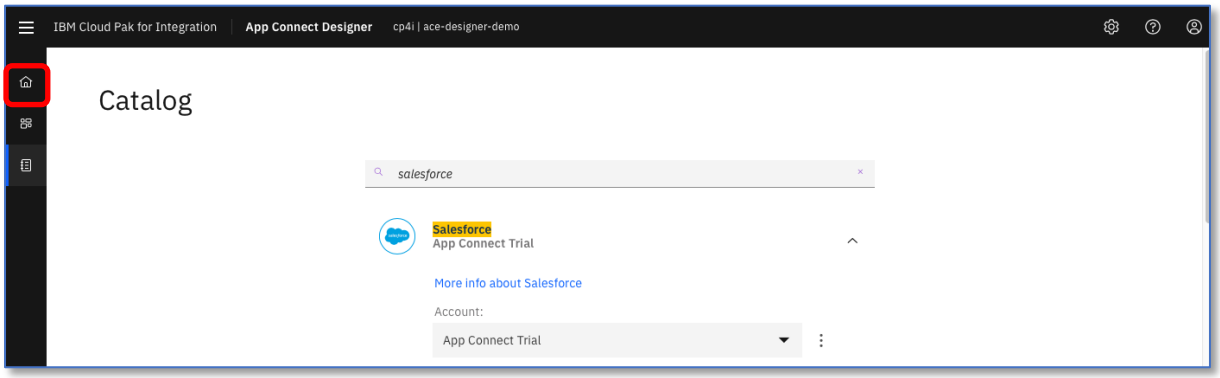


Just as an aside, look at the sheer amount of data and functions available through the connector – you can expand them to see what the actions are. As the connectors are metadata driven, if you customize Salesforce with extra or customized fields, the connectors will pick them up automatically.

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

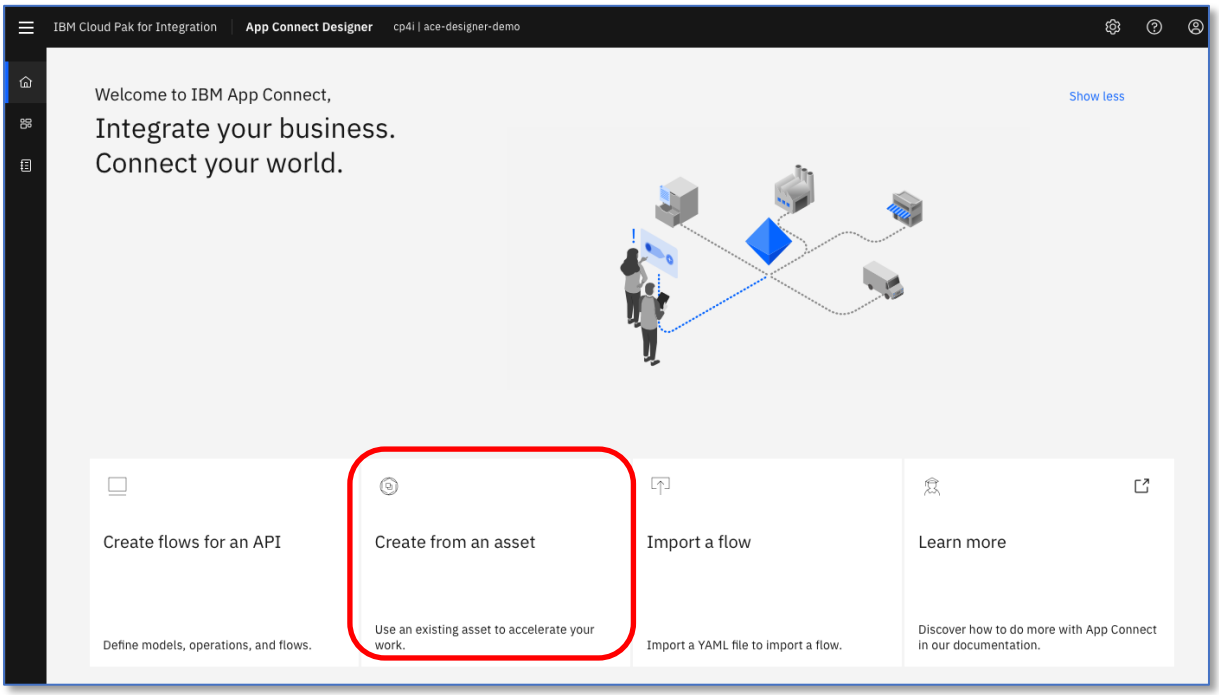
9.5 Importing the Integration flow into designer

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…

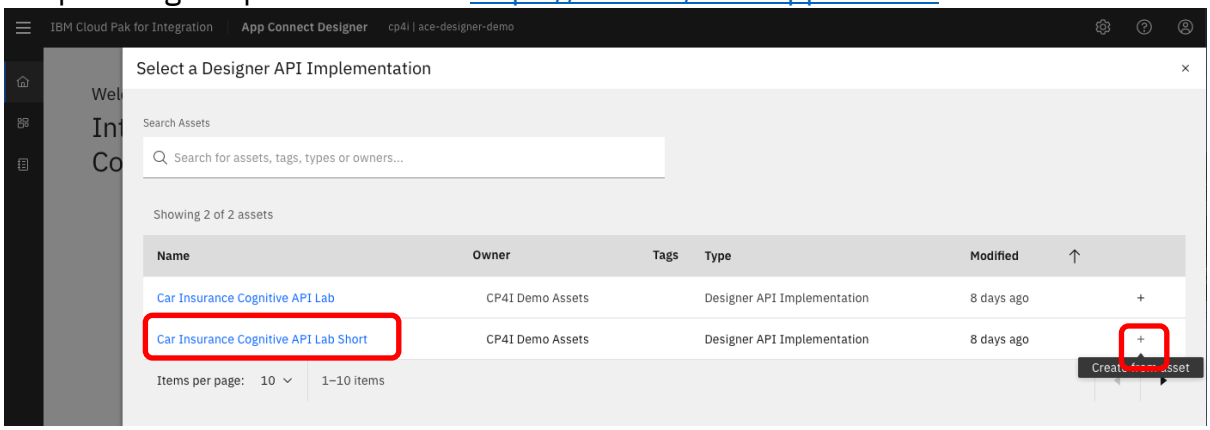
Click on 'Create from an asset' (you might have to wait a few seconds for this tile to appear)



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

It also avoids a LOT of screenshots and 'click here, click there, type this instructions' – you could even probably work out how the flow works just from watching the video here: https://www.youtube.com/watch?v=TRzO26kawu4 but we'll step you through it in this lab.

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



Click on the '+' sign to the right on the 'Car Insurance Cognitive API Lab Short' asset and Create from the Asset (*ensure you select the 'Short' version – the other one is for the extension scenario*)

Integration flows are stored in the Asset Repository as .yaml source files – they look a lot prettier in the tooling though!

9.6 Reviewing our API Integration Flow:

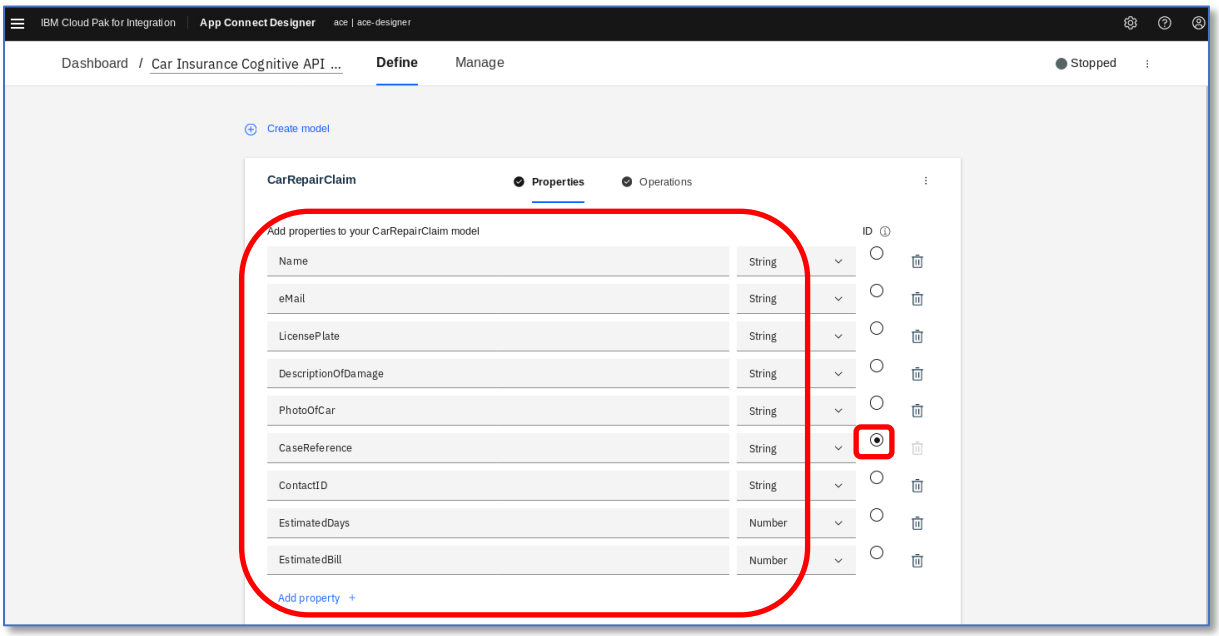
*Lab tip: This section can take some time. If you're more interested as to how the flow is built, go through this section. If you may be short on time, as the flow is pre-built and we won't change it in the lab, you can skip straight to '**Starting the flow**' section and come back here later. There are lots of screen shots, so you can read this lab guide afterwards at your leisure.*

You should now see our flow API open in the designer.

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. If you want, you can use objects for complex structures but we won't here

These are the fields we are going to use for our API – we've created them as part of the asset to save you time. You can rename them if you wish but if you do, our test scripts for the APIs won't match – or work, so leave them as they are for now.



Note that we tell our API which field is the key – in our case, CaseReference. When creating RESTful APIs, they should be resource based and each resource should have a unique key.

CP4I Designer bakes in good REST API creation right into the tooling so you don't need to worry too much about it.

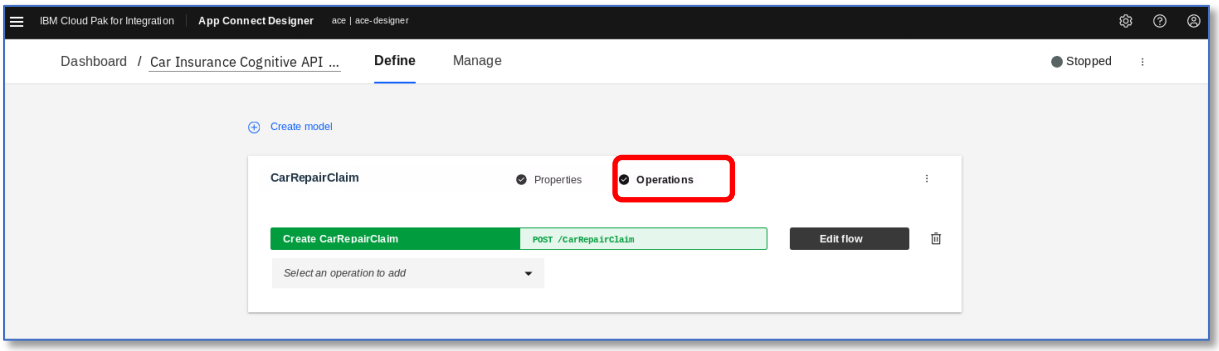
Note that the '`PhotoOfCar`' property is a string – our consumers will pass the photo data in as a string of base64 encoded text. This is one way of passing a binary image to an API.

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

For this lab, we've defined our 'CarRepairClaim' data model. We have data fields – what do we want to do with them?

Now we want to do a ‘Create Car Repair Claim’ operation.

Click ‘Operations' – operations are the actions that the API exposes with the data.



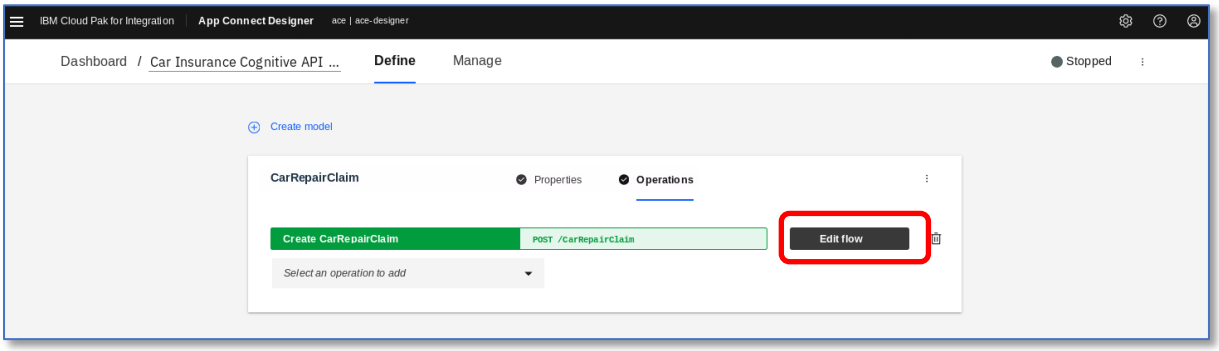
We can have multiple operations in one API – such as ‘Create’ ‘Retrieve’ ‘Update’ etc.

The tooling auto-generates good REST for you, translating into HTTP verbs like GET and POST automatically. You don’t need to know REST to build APIs with CP4I – the knowledge you need is built in.

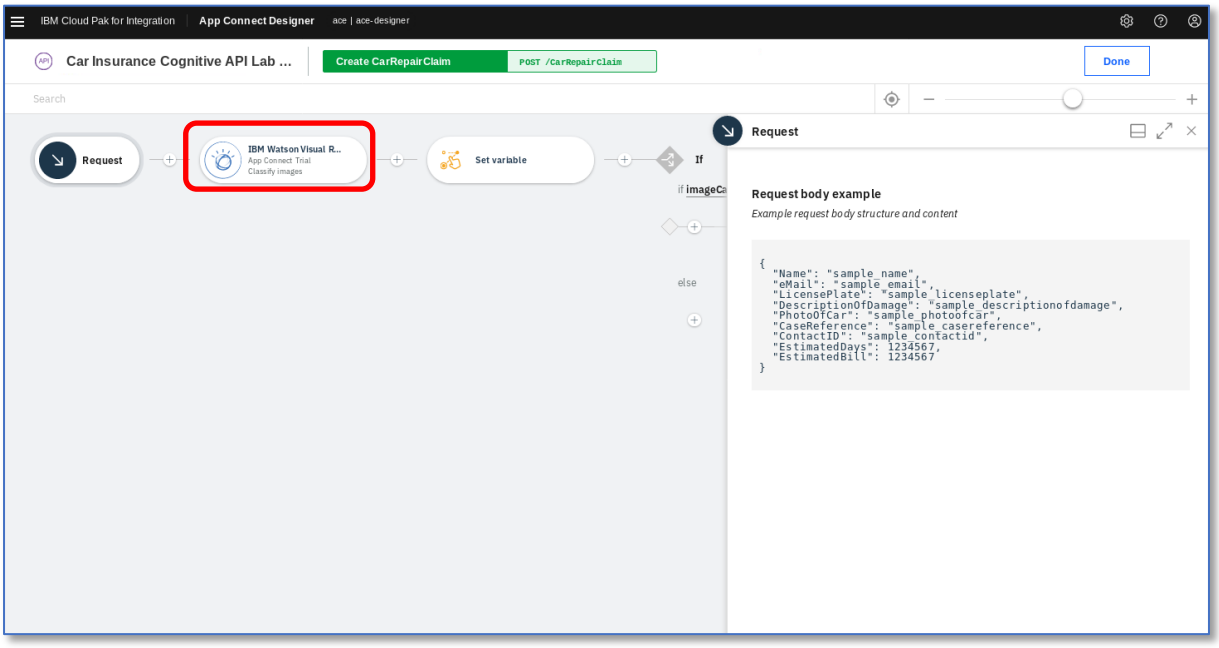
For example, look how a pull-down menu auto generates the HTTP ‘POST’ and the path of /carrepairclaim.

In this lab, we’re going to build just one operation – you can add more if you wish.

We’re going to go into the flow logic – click ‘Edit Flow’



You'll now see the flow in the designer flow editor here:



See the 'App Connect Trial' account name on the IBM Watson Visual Recognition? That's the reason we had to get the account name correct earlier so it matched.

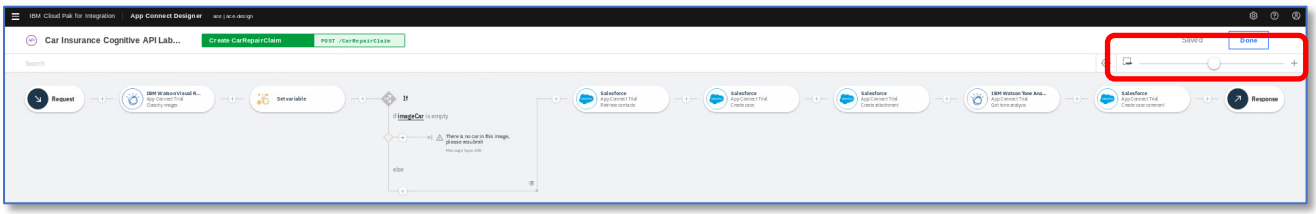
Scroll through all of the connectors in the flow (use the scroll bar at the bottom) and make sure there are no red dots anywhere.

App Connect Designer connects to the endpoint service every time you open the flow to see if there is updated metadata (this is why you can see spinners when you open the flow). This means the services need to be connected correctly. If there is an issue, there will be a red dot on the connector node.

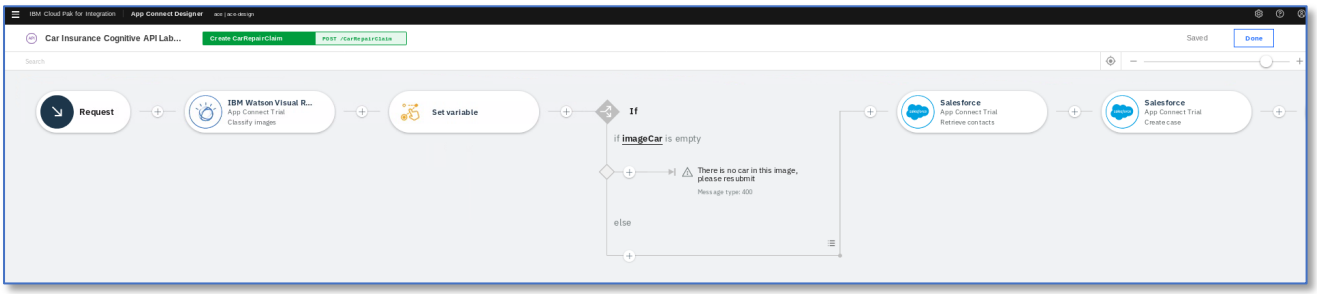
The most likely reason for a red dot is that your connector account name does not match the name of the account used in the flow. To fix this, go back and rename it in the connector (click the cogwheel at the top right and click 'Catalog' to get back to it).

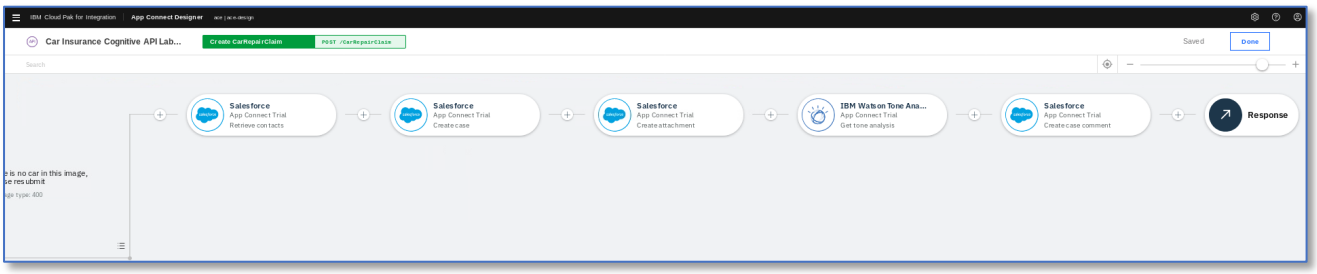
You could rename the accounts in the flow if you wish, but that might make the lab harder to follow: App Connect doesn't really mind what the accounts are called as long as the references all match.

Using the zoom in/out bar, we've shown you the entire flow below:



Or two readable chunks!



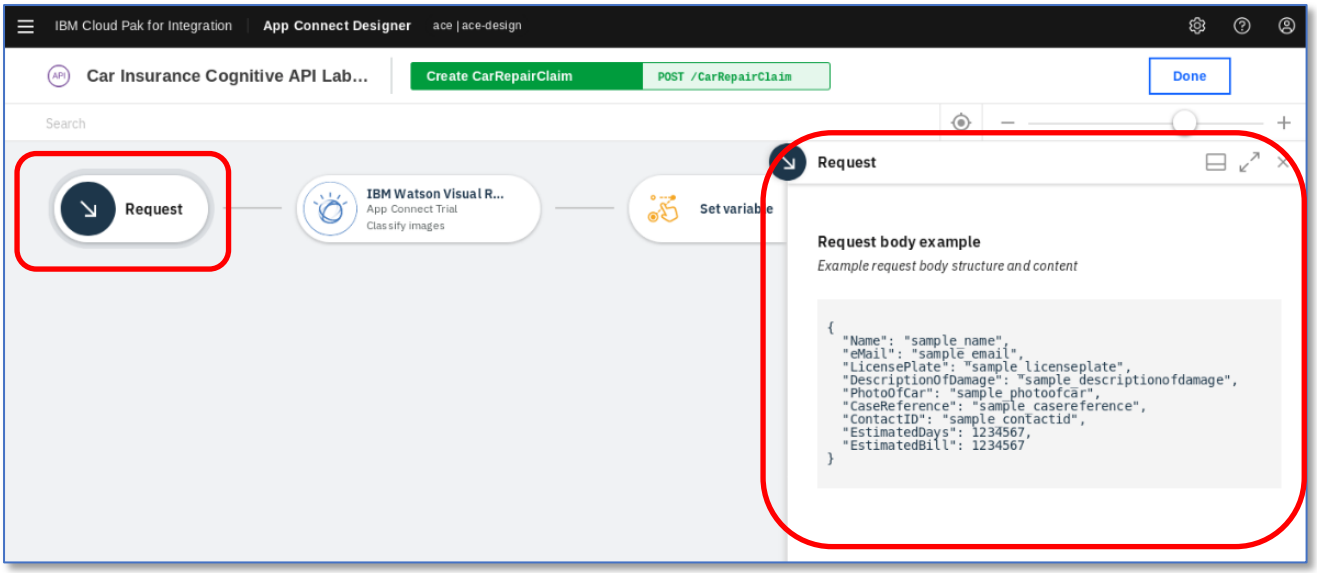


You can see that the flow visually matches the logic we defined at the start. Let's step through.

Lab tip: Again, you may want to skip through to 'Starting the flow' and come back here due to time constraints.

9.6.1 Receive the Customer's car repair request with photograph via an API

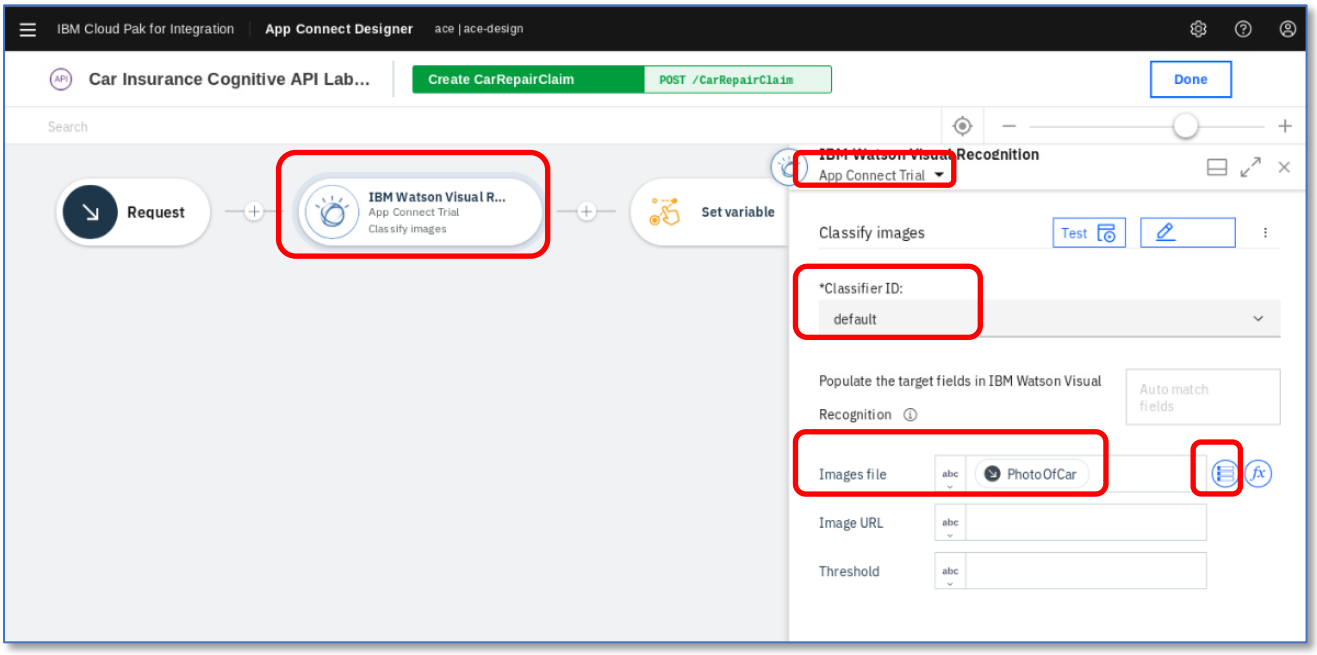
Designer automatically creates an API "Request" and "Response" node for your API flow.
Click on the 'Request' node.



Note how the request body is created from the model – and sample data is automatically generated. When building there is literally nothing to do here – it's done for you.

9.6.2 Use IBM Watson Image Recognition to analyse the photo.

If it is not a valid picture, Watson will return an error immediately to the user calling the API.



We use the built-in Watson Visual Recognition connector that we configured earlier. Note that we selected ‘App Connect Trial’ account here. If you had it named incorrectly (e.g. ‘Account 1’) then you would have an error. To fix it, change to ‘App Connect Trial’ in the pull down.

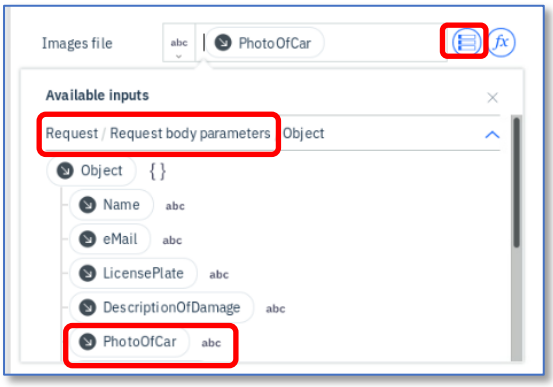
The ‘Classifier ID’ is to tell Watson which image training data set, or classifier it should use. You can train Watson with your own image data e.g. for products your company sells or assets it uses. If you create a custom classifier, the connector will go and find it and offer it to you in a drop-down. We will use the default ‘out of the box’ classifier.

We only need one ‘target field’ to populate – our images file. This is our base64 string with the photo in it.

Designer doesn’t have ‘Mapping Nodes’ – it’s inspired by spreadsheets where you concentrate on what data you want to put in a cell, rather than where source data needs to be mapped to. All of the fields that have been populated in the flow from variables, requests or connectors are automatically stored and are available for you to use at any time.

You can see that we’ve mapped out ‘PhotoOfCar’ field from our request. You can tell it’s from the request because it has the same icon as the request node next to it!

Click the hamburger (three lines) pull down button next to the field:



All of the fields that are in the flow so far are available – just click on the one you want. This is how ‘mapping’ is done in designer – it’s like filling in cells in a spreadsheet.

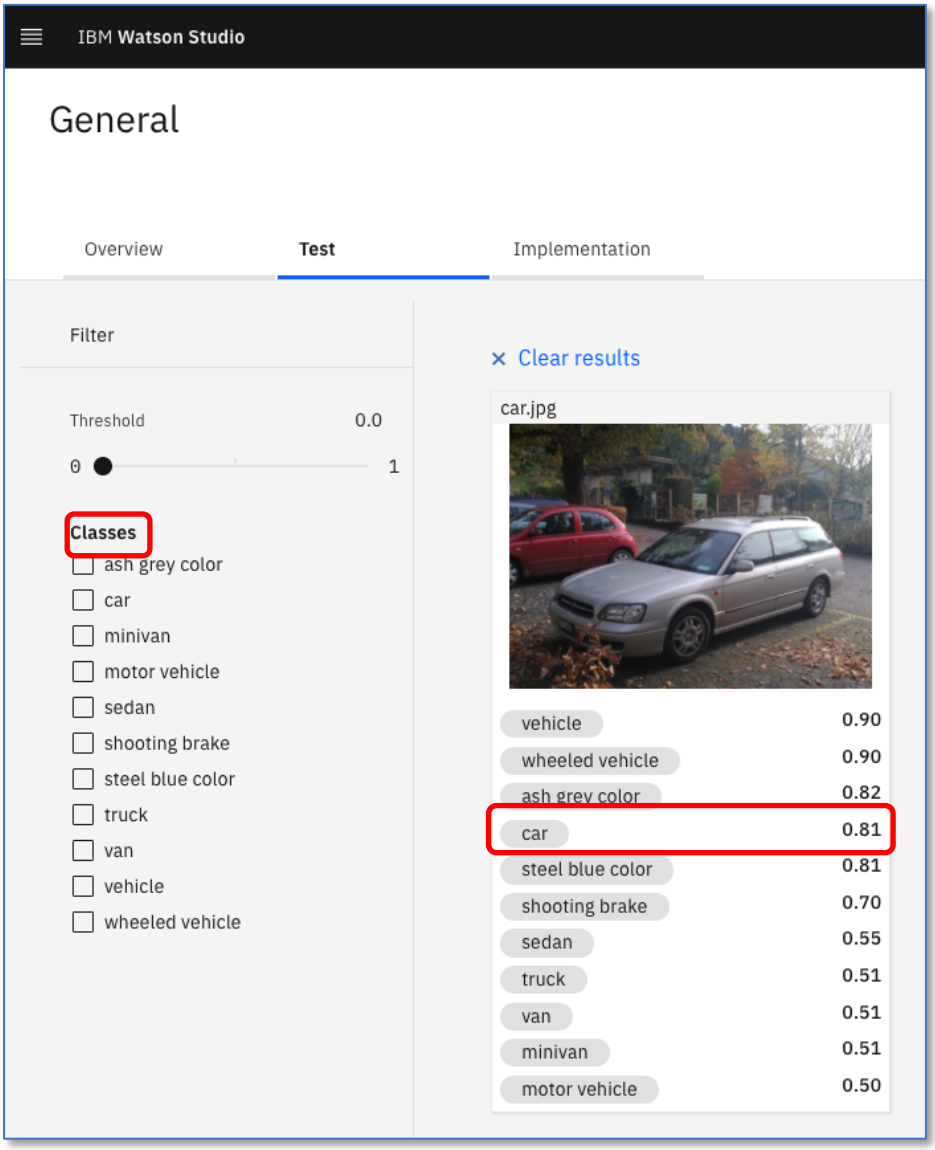
Make sure ‘PhotoOfCar’ is selected (or don’t change it) before you move on.

9.6.3 Check that Watson can 'see' a car in the picture

Watson will return a list of what it thinks it can see in a picture, each with a confidence rating.

For example, for one of our test pictures, we will use a picture of a Subaru SUV. If we ask Watson to look at this, we see the following – it's .81 (81%) confident it can see a car in the picture.

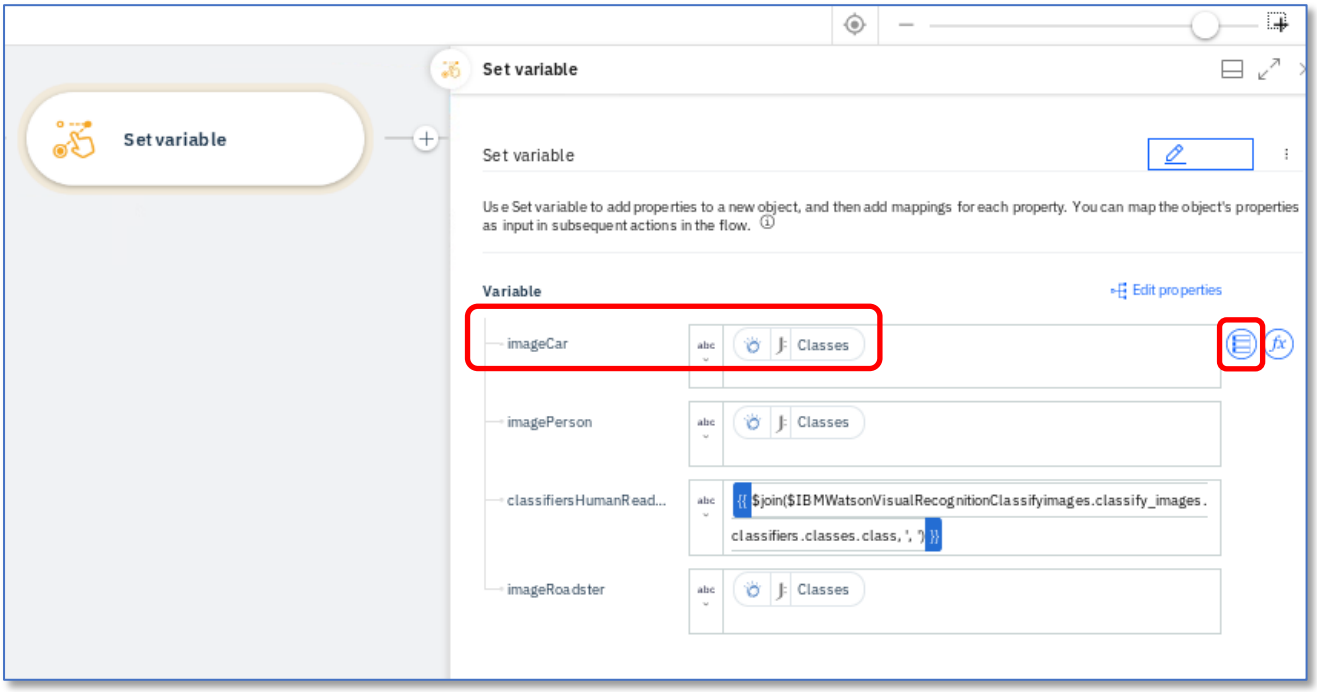
Note the 'classes' that Watson returns – these are the same classes that we will be using in our flow.



(This screenshot is from Watson Studio – Available for free in the IBM Cloud – search for it and you can try it yourself with your cloud account and your Visual Recognition Service instance)

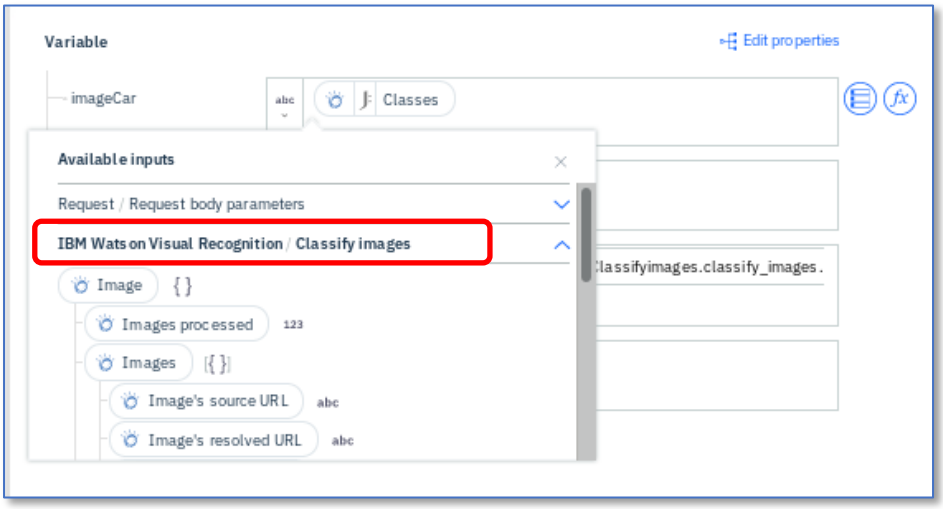
We're going to set variables to check for three things:

- Is there a car in the image? '`ImageCar`'
- Is there a person in the image? '`imagePerson`'
- Is there a roadster (convertible) car in the image – this is for the extension lab, but we have the logic here anyway. '`imageRoadster`'

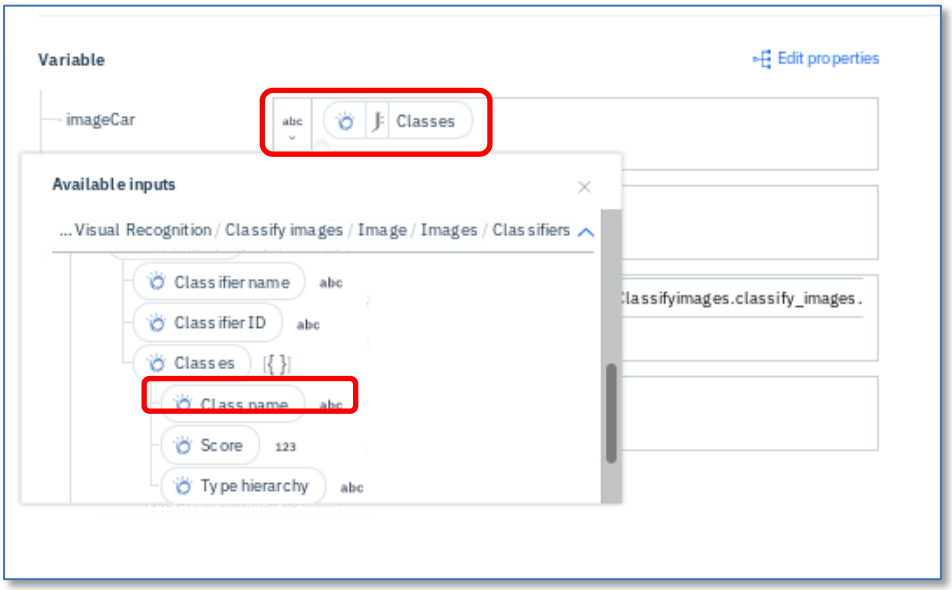


Let's look at 'is there a car?' – '`imageCar`'
Click on the menu on the right, then expand 'IBM Watson Visual Recognition'

The 'Available Inputs' menu appears. You can see we now have fields from both the request and IBM Watson Visual Recognition.



If you scroll down, you'll see we get down to Image->Images[]->Classifiers[]->Classes[]->Class name (together with the Score)

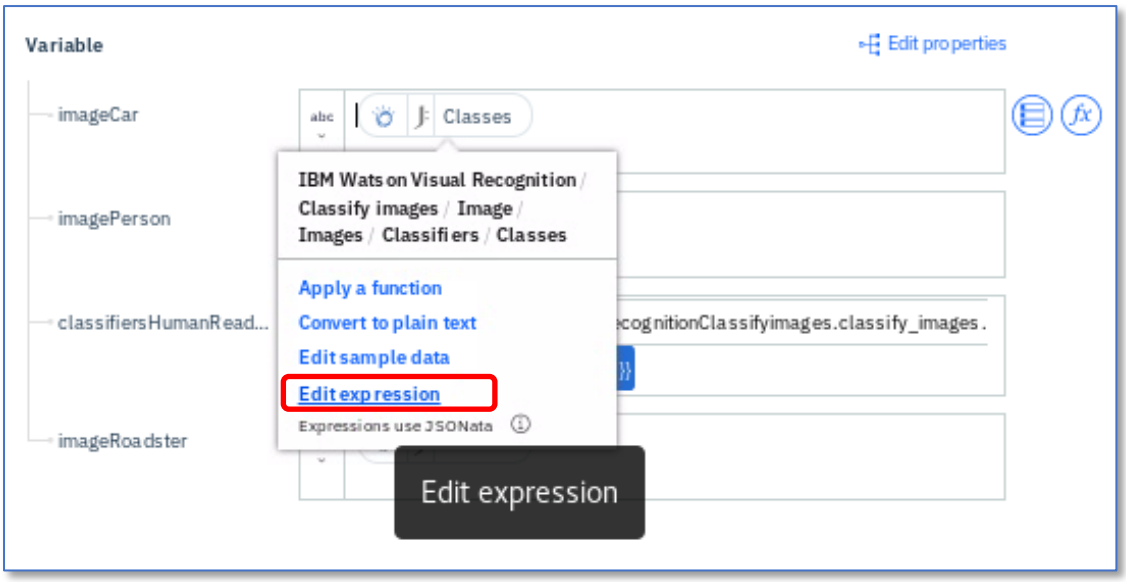


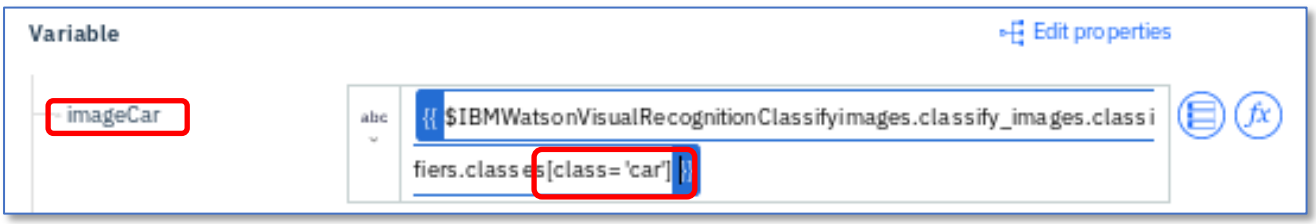
What does this mean?

It means Watson has returned an Image object.
--In the Image object is a list (array) of images (we denote this using [])
----In each image (they may be more than one) there is a list(array) of Classifiers (Watson training)
------In each Classifier, there is a list(array) of Classes. This is what Watson sees. e.g. 'Car'
--------In the Class, there is a class name and a score – amongst other things.

In all of that, we need to say 'Hey Watson – thanks for the data: Can you see a car?' Normally we'd end up coding loops around arrays and if/thens to find it...

App connect does it by using a formula – just like a spreadsheet.
Close the pull-down and click on the 'Classes' bubble in the 'imageCar' field. Click 'Edit expression'





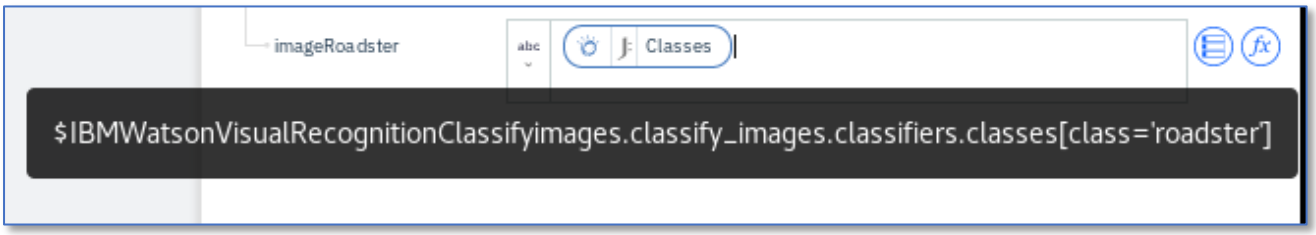
What we did is click on the hierarchy pull down to build a query that looks like this:
`$IBMWatsonVisualRecognitionClassifyimages.classify_images.classifiers.classes[class='car']`
(we had to manually add the [class='car'] part at the end)

All this does is go down the hierarchy, each level separated by dots and then has a select query at the end [class='car'] to give us all the entries where the class is a car.

App Connect automatically scans through the entire hierarchy, sorting out things like arrays/lists and objects and lets us get straight to the data we want.

We use the same approach to populate imagePerson using [class='car'] and imageRoadster [class='roadster'].

If you want to more easily see the query expression, then hover over the 'classes' bubble e.g. here:



All mapping is done the same way - for example, we want a string that joins (concatenates) all of the classes (things that Watson can see) together, separated by commas so it's Human Readable. For this we use 'apply a function' and select 'Join' from String functions, just like building a spreadsheet formula.

To get the 'dot hierarchy', just use the pull down variable explorer. Pick the field you want and the choose 'Apply a function'

```
$join($IBMWatsonVisualRecognitionClassifyimages.classify_images.classifiers.classes.class, ',
')
```

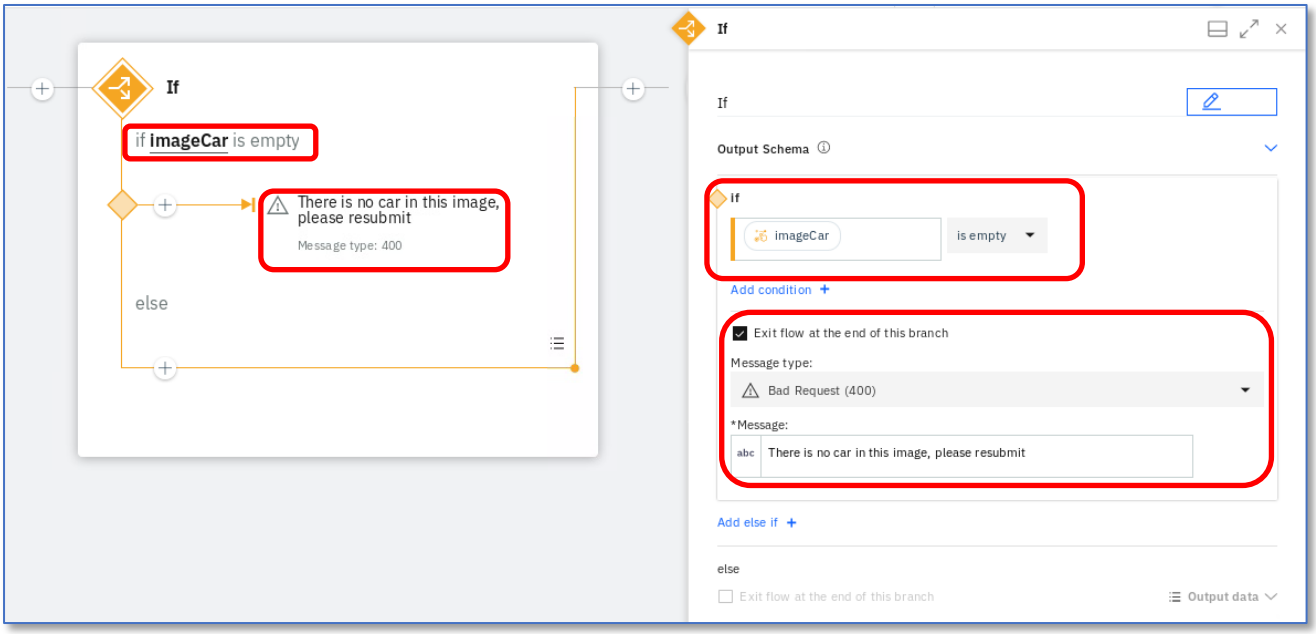
Note that App Connect joins all the classes together in one go – no for loops, building up strings etc. It's a different way of looking at mapping.

All of this mapping uses a language called JSONata – more details here https://jsonata.org

9.6.4 If not, we will immediately respond back with an error saying 'There is no car in this picture'

We now know if there is a car or not in our image...if there is no car image (i.e. if 'imageCar' is empty) we want to send back an error.

We use an App Connect 'If' node to visually show us our logic:



We visually create an 'If imagecar is empty' check. If it is empty, we send a 'bad request' response – note that we don't need to remember that in REST APIs, 'Bad Request' is 'HTTP 400' – App Connect knows this – just pull the response from the drop down.

We also add a 'There is no car in this image, please resubmit' error.

9.6.5 Create a ‘Case’ in Salesforce with the data from the API.

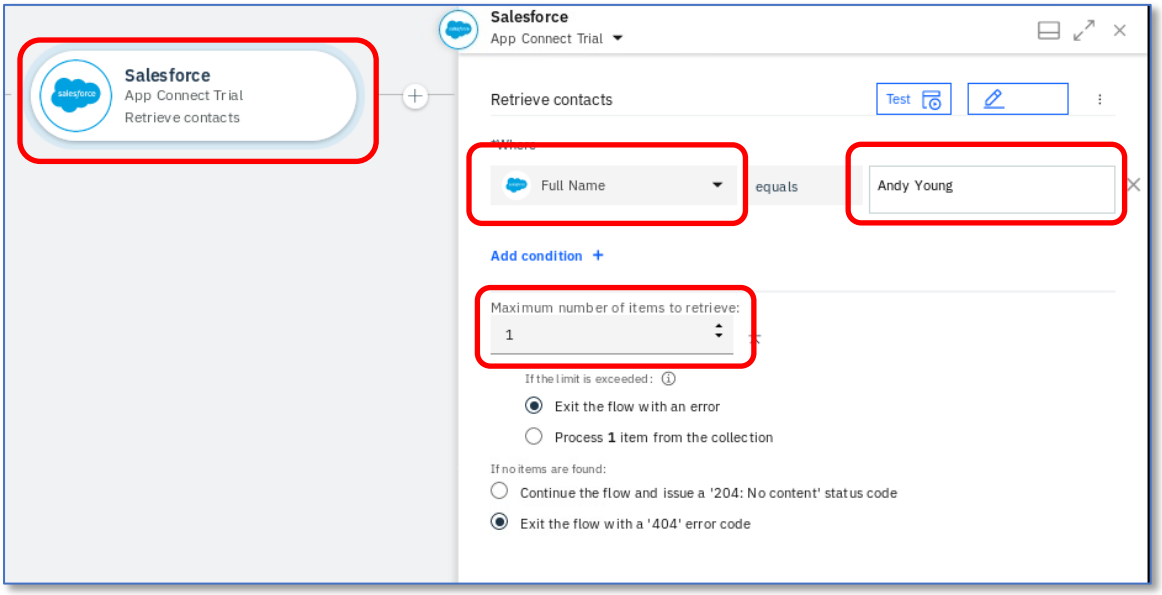
This Case is where we store the details and progress of our repair.

We’ve already connected to Salesforce, so we can use the Salesforce connector.

We want to add a contact for this case, but we need the contact ID from Salesforce, not the name when we create the case.

This is a very common integration issue – systems need IDs and not names. No problem, sorting this is simple!

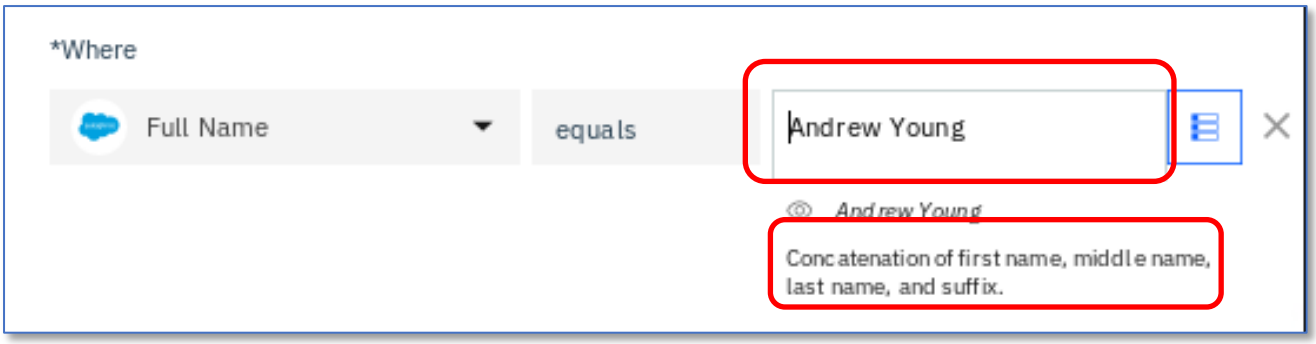
Click on ‘Retrieve Contacts’



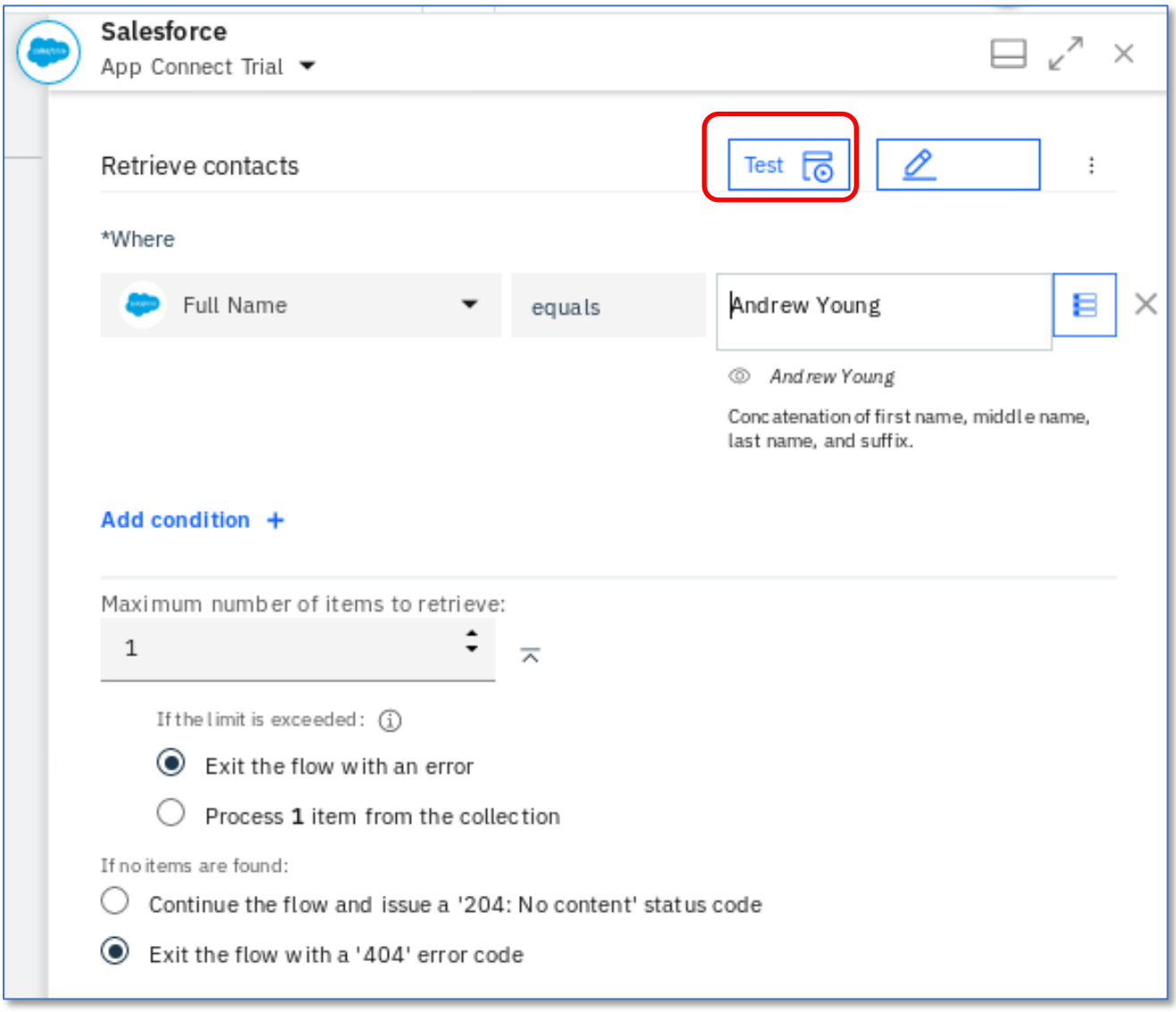
We’re going to use ‘Andy Young’ as our contact – he’s the contact for the insurance company that sends customers. Salesforce Developer Accounts have a pre-populated set of data that you can use to test. ‘Andy Young’ is one of those pre-populated contacts. We will hard-code his name for speed in this lab.

But how do we know exactly what ‘Full Name’ means? Does it have ‘Mr?’ in it? Is it ‘Andy’ or ‘Andrew’? Do we need his middle name?

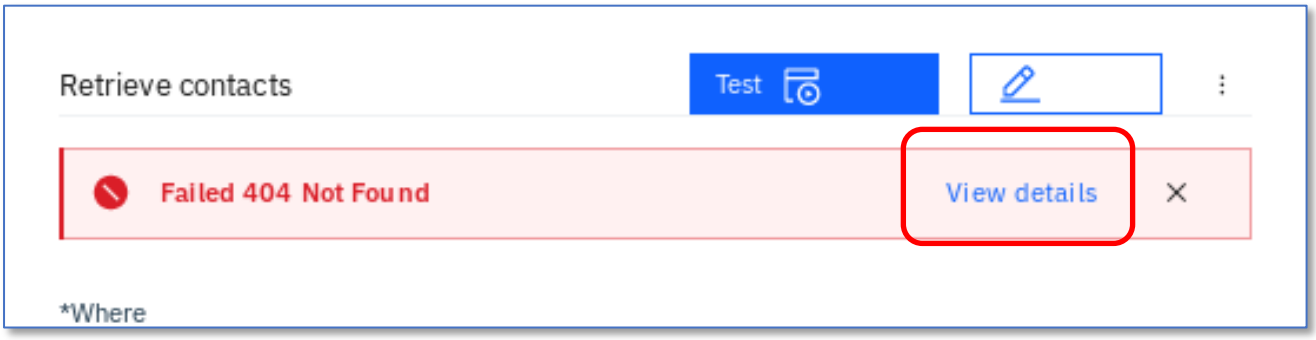
Change the name to ‘Andrew Young’ and we’ll find out. Note that App Connect gives you the field description to help you out. If you add any expressions such as $uppercase(Full Name) then the preview (next to the eye) shows you what your field will look like after your expression.

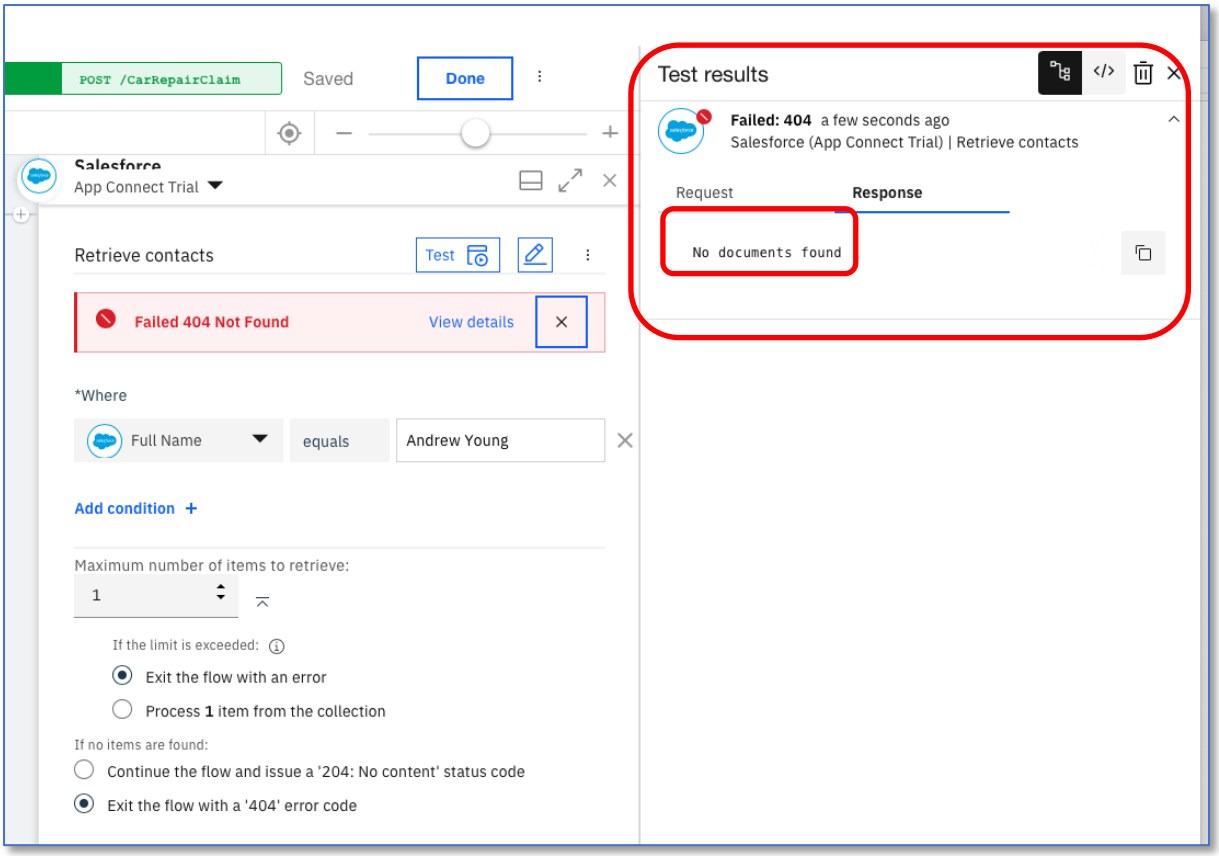


Now click the ‘Test’ button

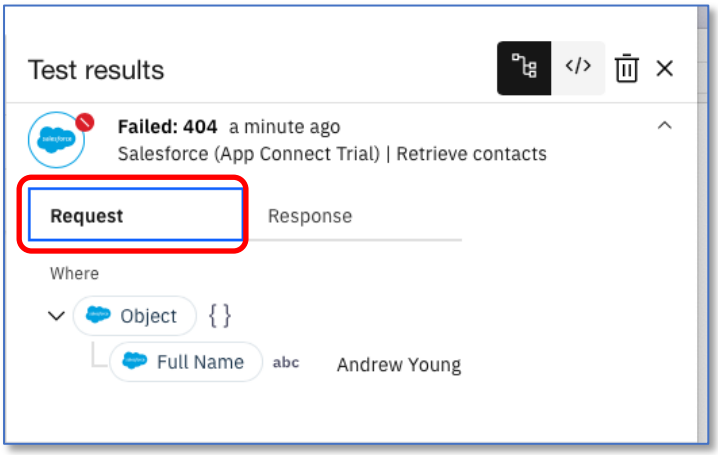


We can go straight off to SalesForce to check – we get this:

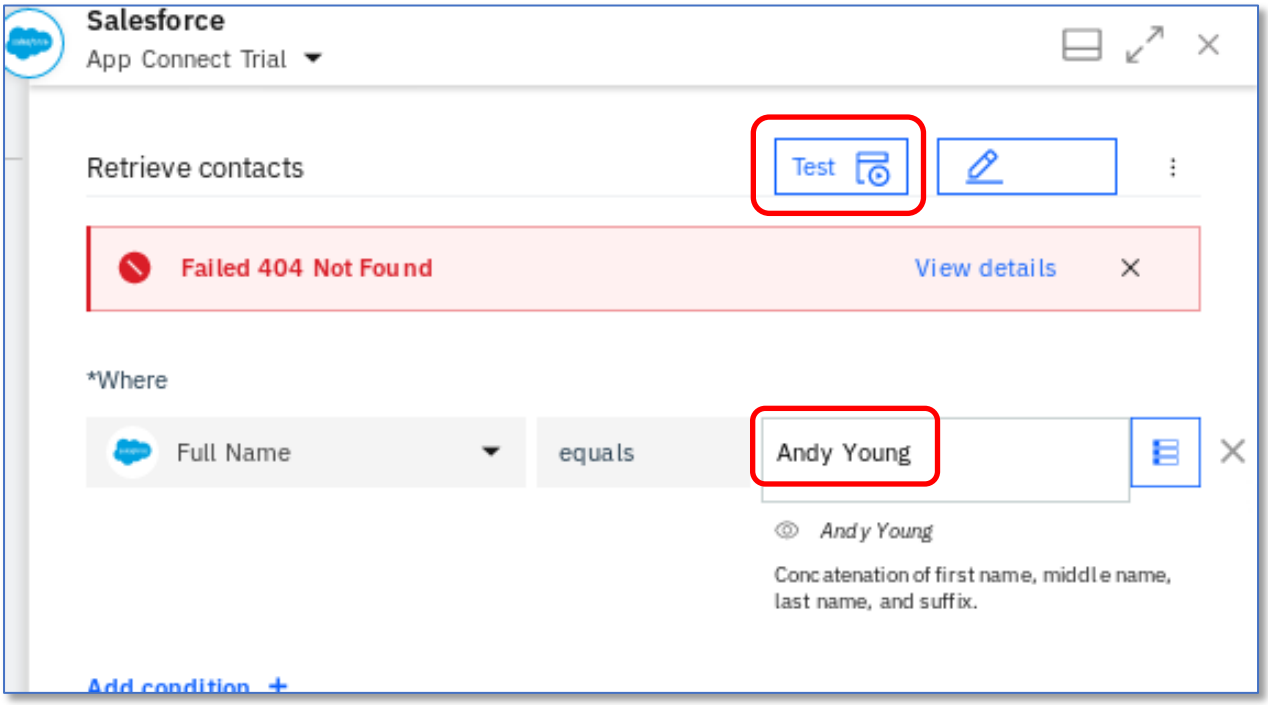




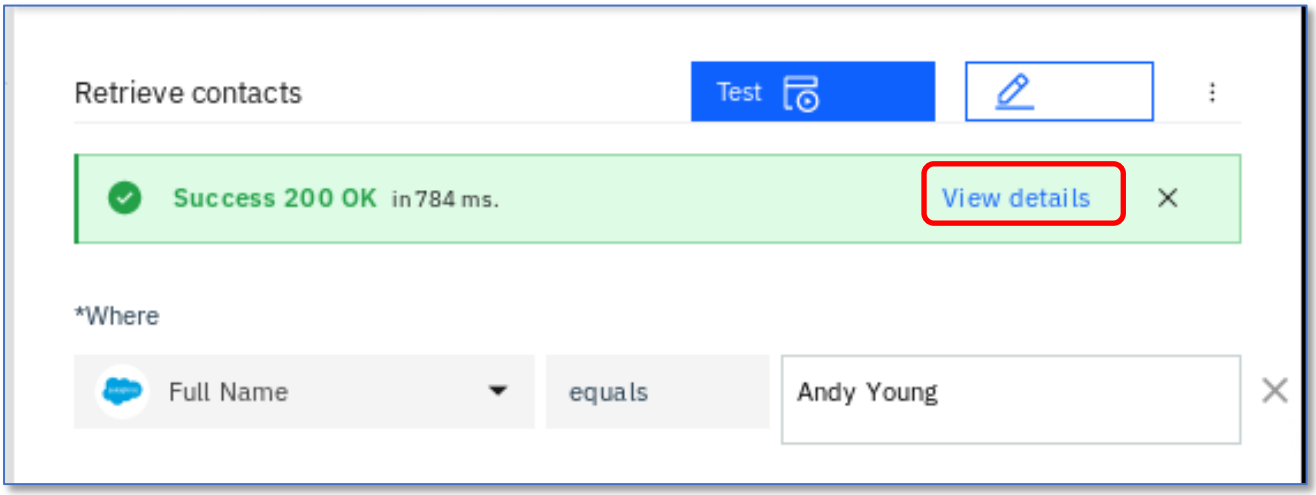
You can see the request as well as the response -click ‘request’

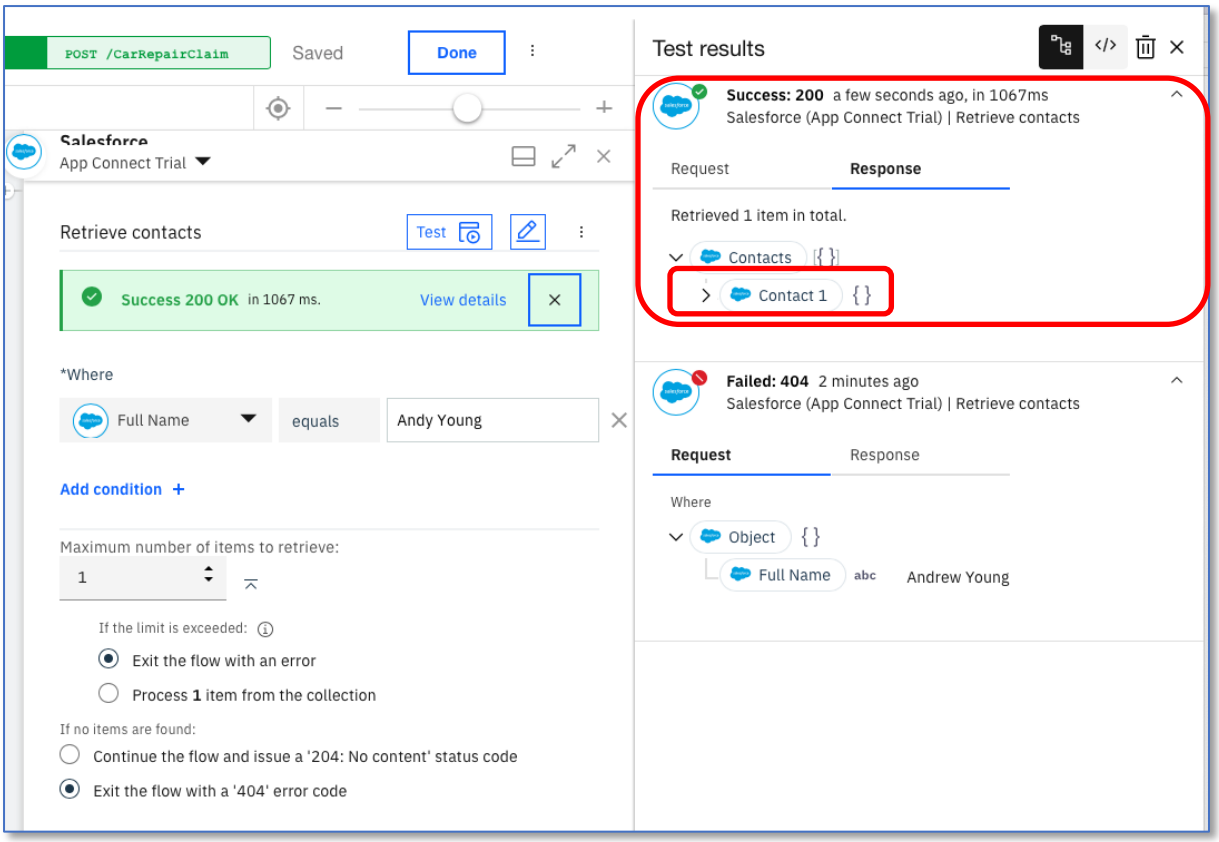


OK, let's put the field back to 'Andy Young' and try clicking 'Test' again.



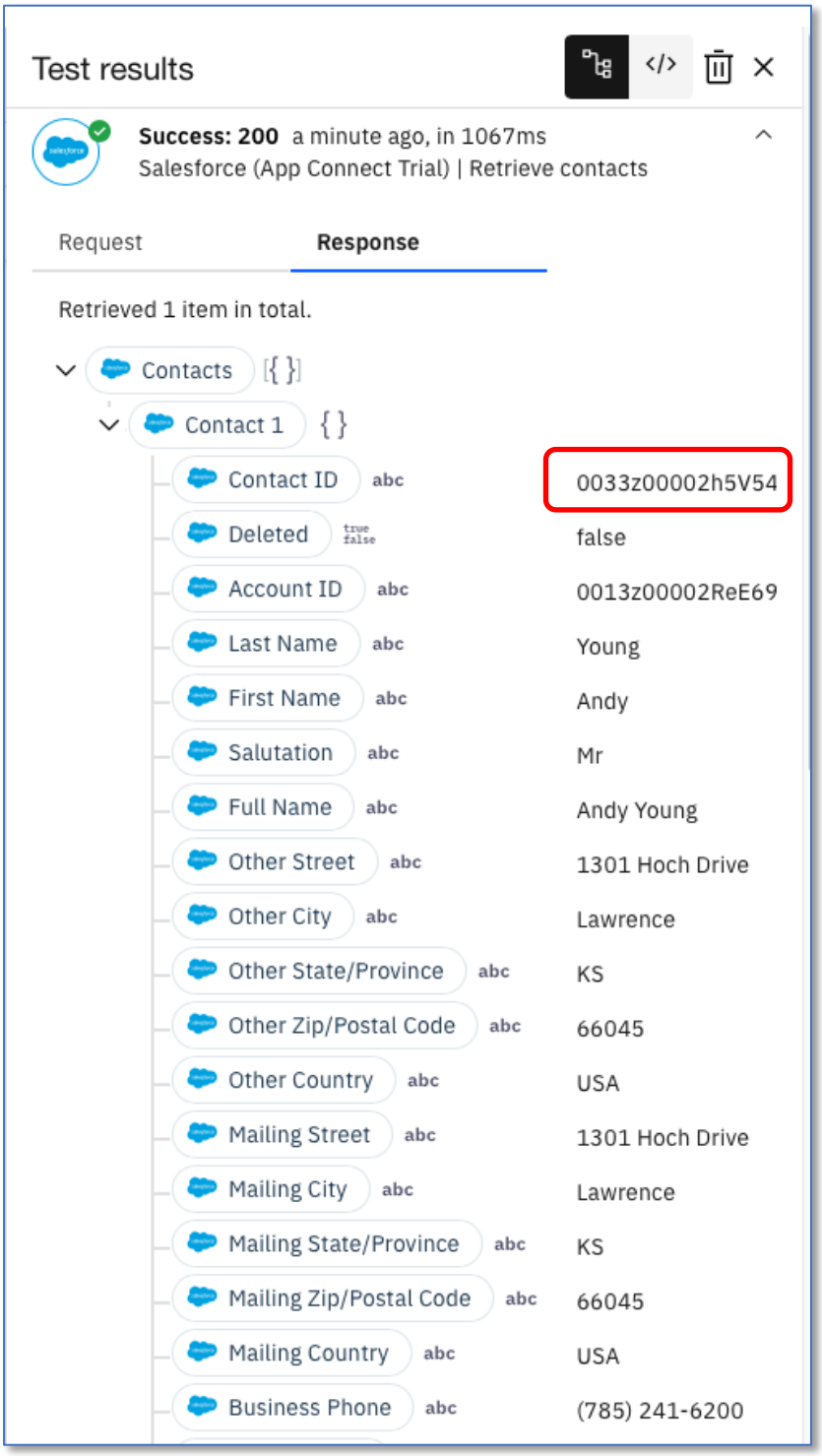
Success! Hooray, let's check our result! Click 'View details'





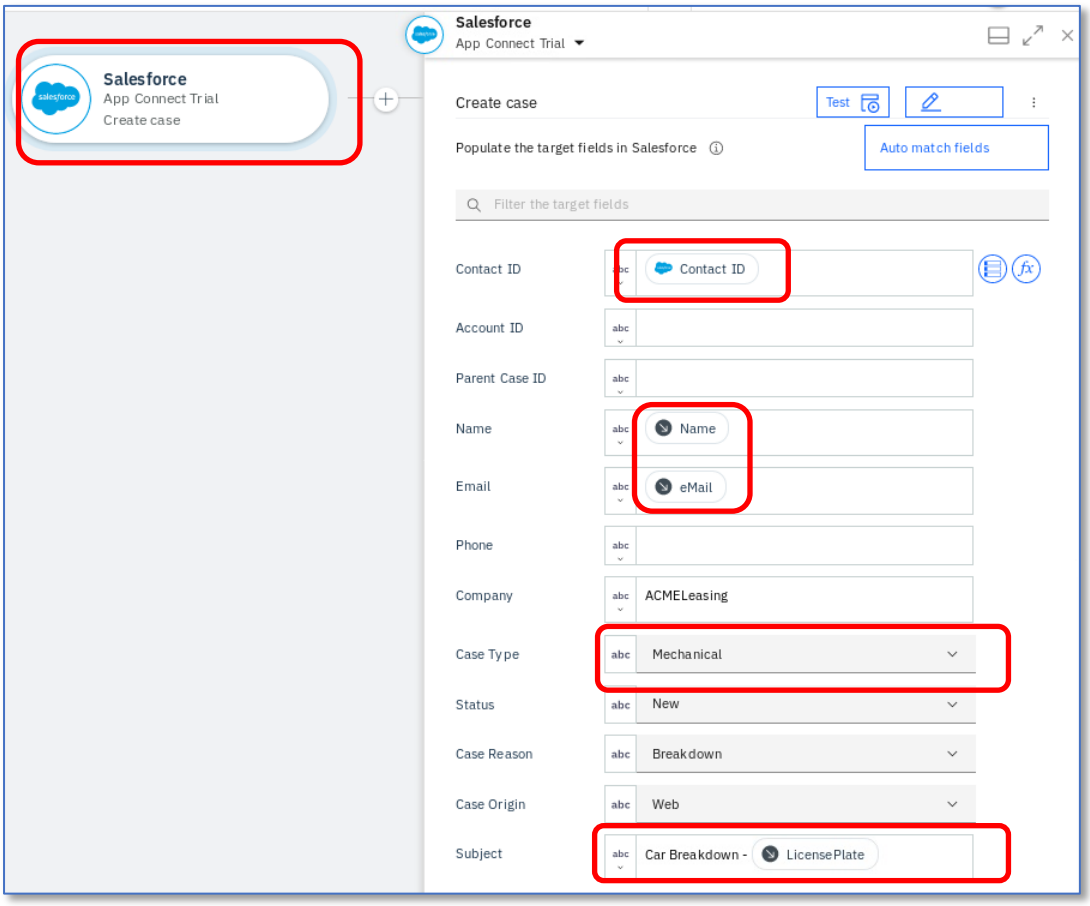
There's our test results, right in the tooling, right from the real system in the cloud. This works with all of the connectors such as Watson in our flows here. It's a great way of checking your integration calls work the way you want them to without having to test the whole flow.

Click on Contacts/Contact1 and you'll see:



All of the data back from Salesforce – in the same format you use to 'map' fields!

Now we have the ID that we need, let's create our Salesforce case. Click on the Salesforce – Create case node. Note that we just re-use the same connector but with a different operation and data.



Note that we can see that our contact ID comes from the previous 'retrieve contact' Salesforce Call. The Name and email come from the API Request.

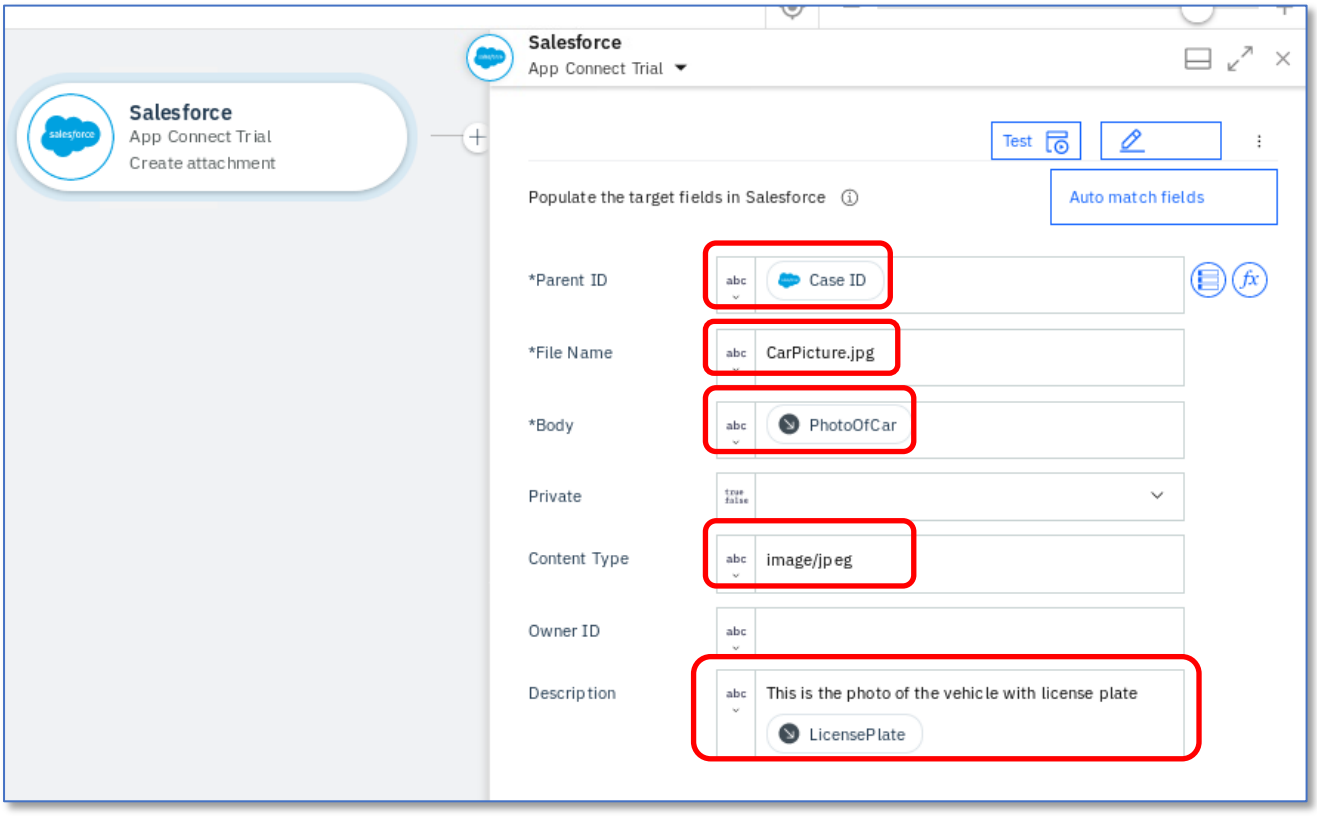
The connector 'knows' that fields like 'Case Type' have a limited number of values in Salesforce – so it automatically converts them into pull-down lists of values for you to choose from.

Also our subject. It's like a spreadsheet – we just type in what we want. No "concatenation" code, no adding strings together, no string appenders!

9.6.6 Add the photograph to our Salesforce case so we have it stored.

To add a photograph, we need to create a salesforce attachment – that's easy, just use the connector again.

Click on 'Create Attachment'

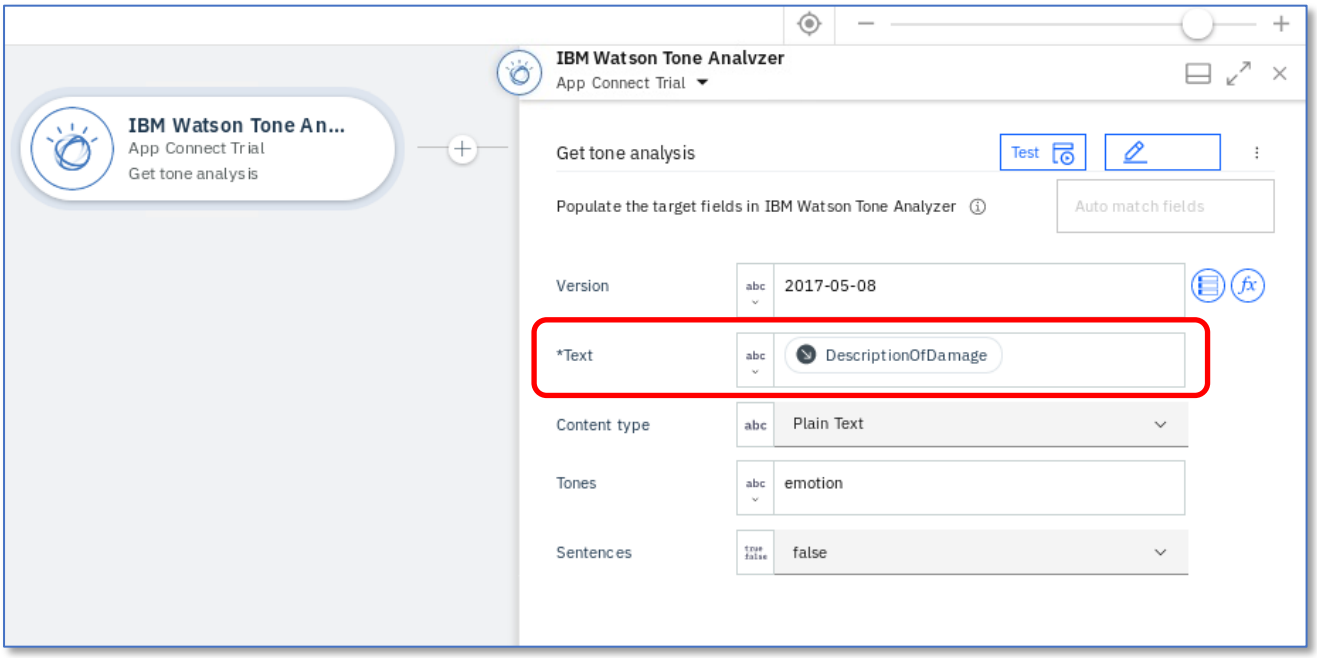


Note that we use the Case ID that is a returned value from the 'Create Case' connector call – it's been kept in the flow automatically. We send the PhotoOfCar as a base64 string and we tell Salesforce that the content Type is image/jpeg.

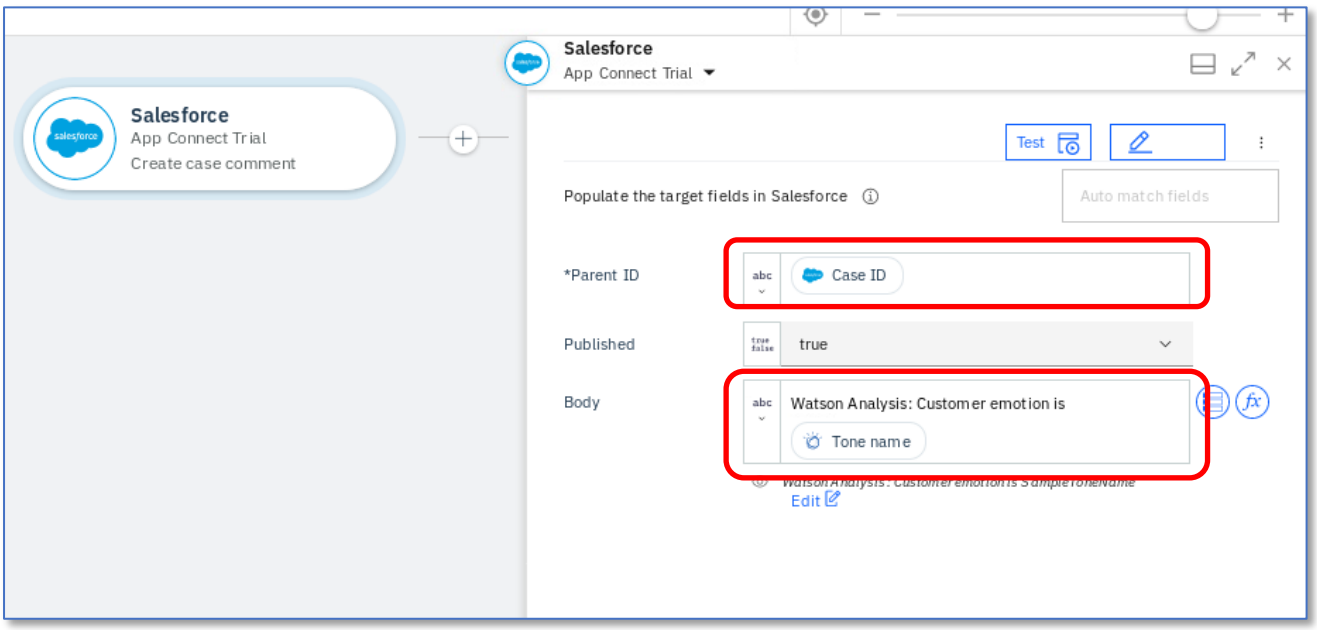
9.6.7 Analyse the description of the problem as described by the customer using IBM Watson Tone Analysis.

We store this in Salesforce for future reference – if the customer is angry or upset, we may wish to take further action or treat them more carefully.

First we'll use the Watson Tone Analyzer; Click on 'Get tone analysis'



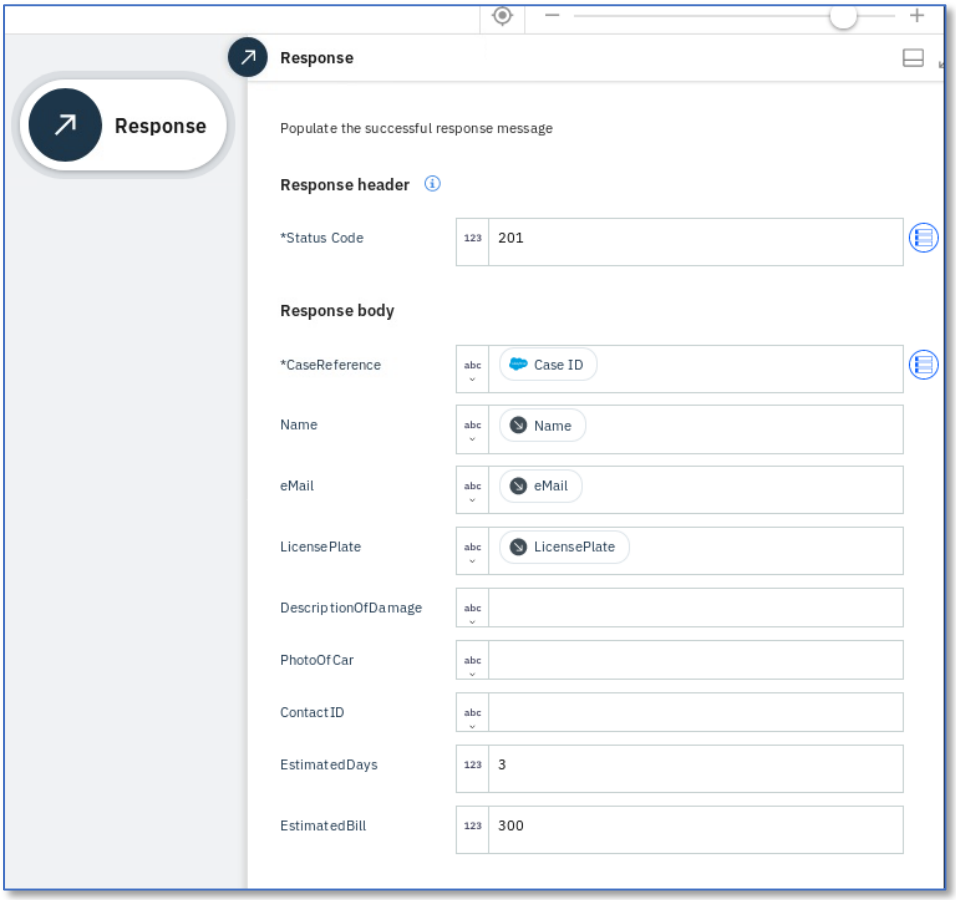
Then we'll add a comment to the case with the Salesforce connector and give it the tone name from Watson.



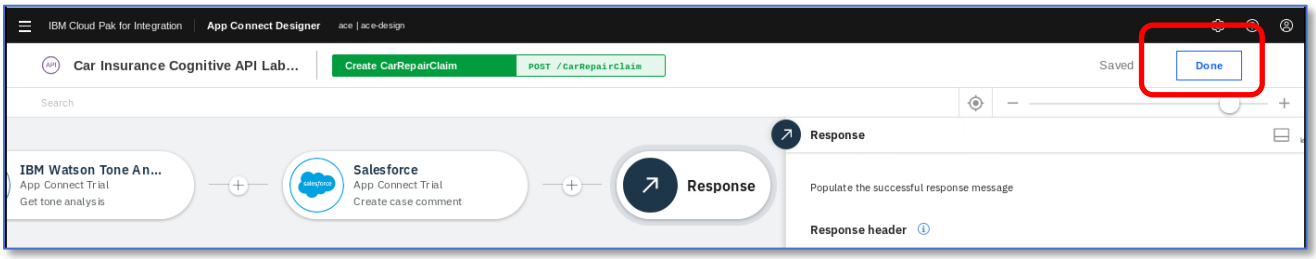
9.6.8 Send a response back to the customer with their Salesforce case reference

For future enquiries and also an estimate of how long it will take to repair and how much it will cost (These are hard coded in this lab)

Click on the Response node – we just fill in the values we want, like all the others.



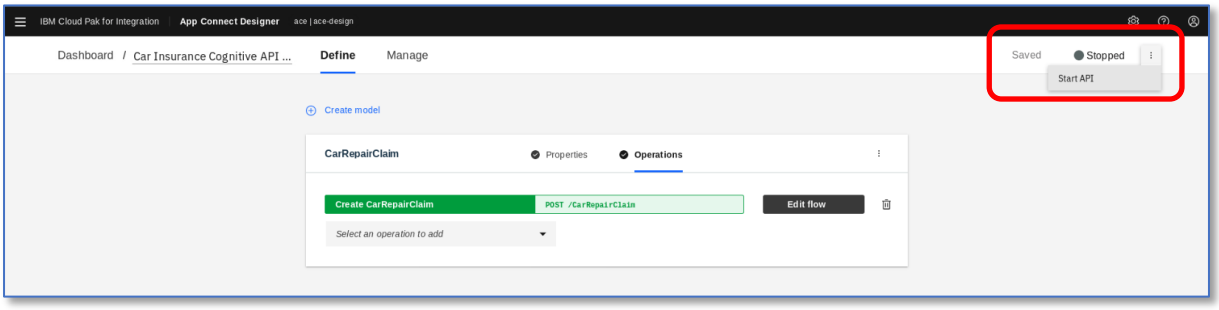
Click ‘Done’ we’ve built the flow – let’s start it!



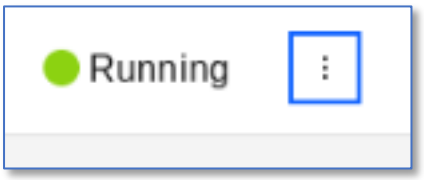
9.7 Starting the flow:

Now we've looked at the integration flow, let's start it up.

Click 'Start API' on the three dot menu at the top right:



Your API should change to a status of 'Running' like below



Now our flow is running, we need to test it.

10 Testing our API Integration Flow

Now we've built our API, we need to test it. In the course of this lab, we will want to test our APIs in three places:

- In the App Connect Designer (Where we've just built our flow)
- When it's deployed to the Cloud Pak App Connect Runtime
- When it's being called through the API Connect Gateway and Portal

If you want, you can skip testing until the end of the lab and use the API Connect test GUI to test – skip straight to section 13 'Deploying the integration flow' if you want to skip testing the API end-to-end here and do it later.

All of these API deployment endpoints will use the same data, verbs and structure – the differences will be in the endpoint and how we authenticate to the provider.

There are three variables that will change for our test cases

10.1 API Base path

This is the first part of the URL e.g. `https://host:port/Car_Insurance_Cognitive_API_Lab_Short` this will change depending on where we deploy our API endpoint e.g. in Designer, In App Connect or exposed through the API Gateway in API Connect.

10.2 APIKey / Client ID

This is the authentication method that we will use with API Connect. This is described in the request as a header of X-IBM-Client-Id:<<apikey>>

10.3 UserID and Password

This is used by the App Connect Designer to authenticate users.

11 How we will test the APIs

APIs can be tested in a number of different ways, for example using the IBM API Test and Monitor tool – available for free here: https://www.ibm.com/uk-en/cloud/api-connect/api-test

For simplicity and speed, as we're using base64 pictures, we will use simple curl scripts so that we can call the APIs from the command line – we can also use these in CI/CD pipelines if we want.

Our curl scripts start like this:

```
curl -k -u $cp4iuser:$cp4ipw --request POST   --url
$cp4ibasepath/CarRepairClaim --header "X-IBM-Client-
Id:$cp4iclientid" --header 'accept: application/json'   --
header 'content-type: application/json' --data
'{"<<DataGoesHere>>….
```

Curl is 'Client URL' – it's a way of calling an HTTP service from the command line: Let's break it down:

`curl`: Name of the command. `-k` means don't check for certificate validity

`-u`: specifies username:password to authenticate APIs that need those

`--request`: tells us what HTTP verb to use. In our case 'POST' which is HTTP for 'Create'

`--url`: Specifies the location of the resource we want to act on. In our case, our resource is the CarRepairClaim

`--header:` These are HTTP headers which add extra information to the request. `'X-IBM-Client-Id'` is how we will send the client ID when we call through API Connect Secure Gateway
`'accept: application/json'` and `'content-type: application/json'` mean that we will receive and send JSON formatted data (As opposed to XML for instance)

`--data`: This is the actual request data – in JSON format as specified above.

The parts in **BOLD** above are those parts which will use environment variables so that we can re-use the script with different values.

These are the environment variables that we want to change:

11.1 $cp4ibasepath

This is the host and port and the first part of the API path, it also includes the 'http' or 'https' part of the URL.

We will set this to
http://myserver:myport/Car Insurance Cognitive API Lab Short
(note that 'myserver:myport' will be specific to your cluster – don't type them literally)

11.2 $cp4iuser and $cp4ipw

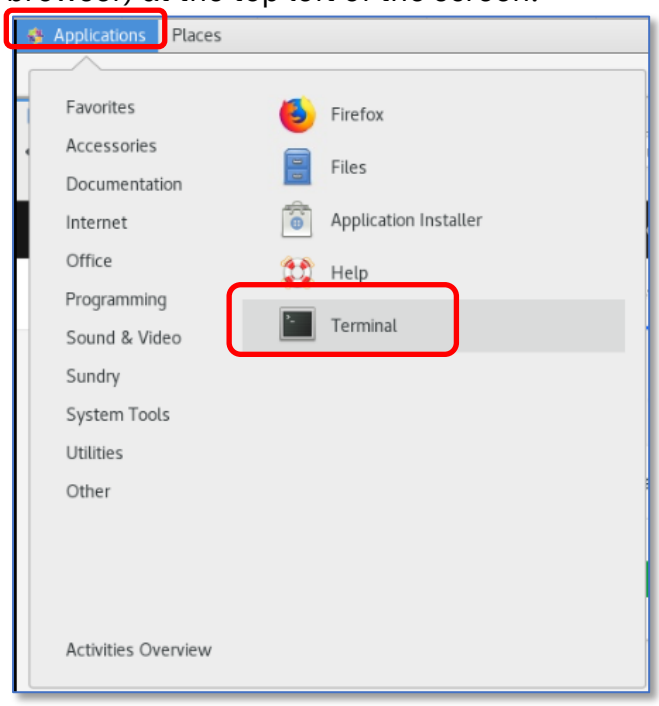
These are the userID and Password for the designer instance

11.3 $cp4iclientid

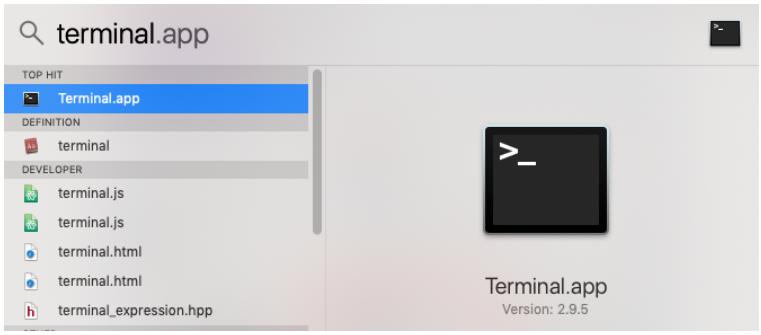
This is the clientid – this is used for authentication for API Connect.

The test scripts are pre-built on github ready for you to use

To get the scripts first go to the terminal window on your laptop: It's available in the LINUX Applications menu (not the Cloud Pak for Integration Menu in the FireFox browser) at the top left of the screen:



For a Mac, the easiest way is to use spotlight search: Do Cmd-Space and select 'terminal.app'



We will use curl to download the scripts from github to our desktop VM. Note that -o in curl means 'write to an output file' rather than display the result on the screen.

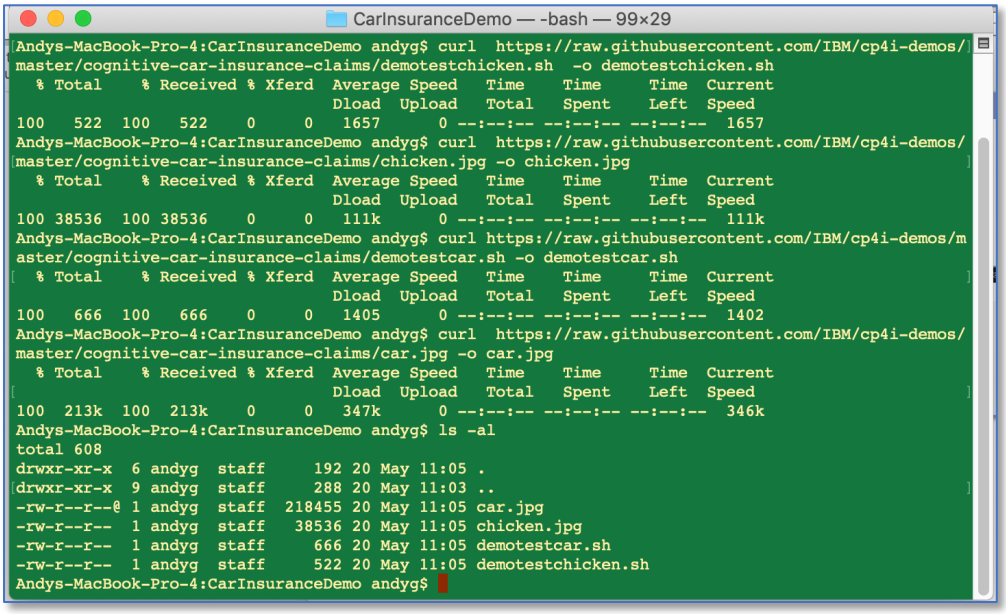
We need to download the following files from the IBM git repository:

```
demotestchicken.sh
chicken.jpg
demotestcar.sh
car.jpg
```

In the terminal window, enter the following commands:

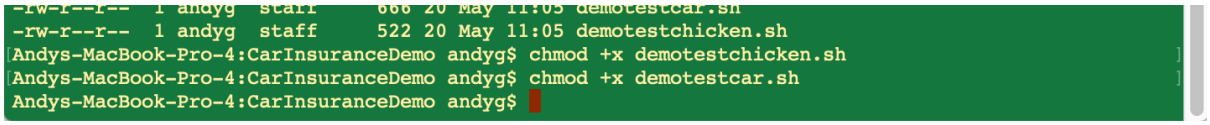
```
curl  https://raw.githubusercontent.com/IBM/cp4i-demos/master/cognitive-car-insurance-claims/demotestchicken.sh  -o demotestchicken.sh

curl  https://raw.githubusercontent.com/IBM/cp4i-demos/master/cognitive-car-insurance-claims/chicken.jpg -o chicken.jpg

curl https://raw.githubusercontent.com/IBM/cp4i-demos/master/cognitive-car-insurance-claims/demotestcar.sh -o demotestcar.sh

curl  https://raw.githubusercontent.com/IBM/cp4i-demos/master/cognitive-car-insurance-claims/car.jpg -o car.jpg
```

You should see output similar to below.



We then need to make the scripts executable – in the terminal window, enter the following two commands.

```
chmod +x demotestchicken.sh
chmod +x demotestcar.sh
```

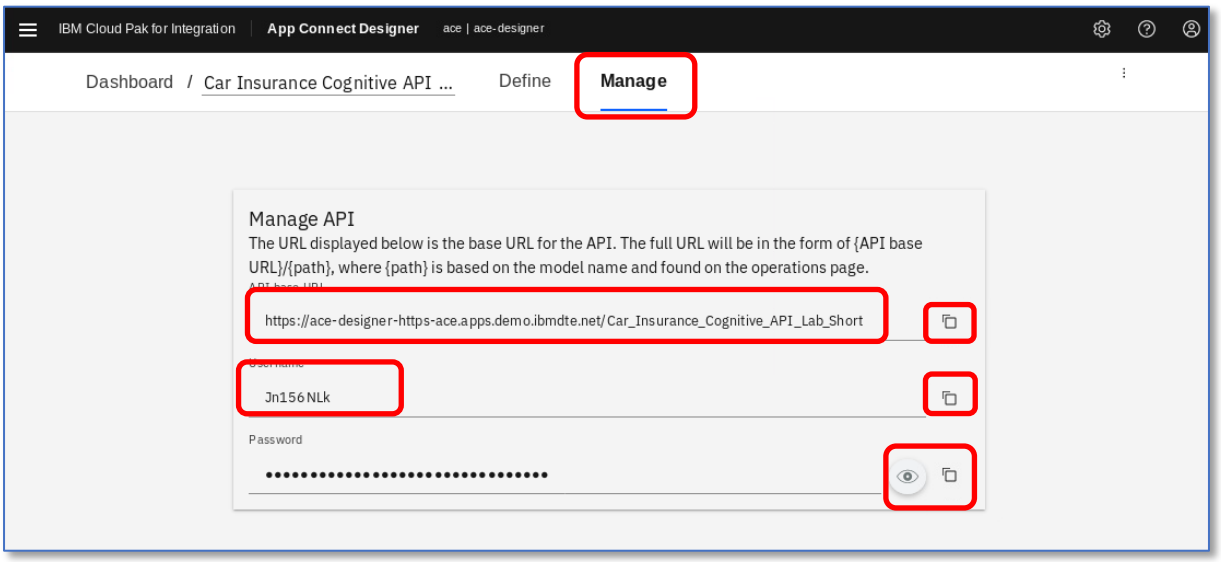


Note that these commands do not give any 'output' – there is no 'OK' or anything. Don't worry, everything is fine if you don't get any errors.

Next we need to setup the variables so that our script calls the API in the correct place with the correct credentials.

11.4 Setting Environment Variables to test in the ACE Designer

To get the credentials for the designer, we go to the 'Manage' tab in designer.



This gives us the values for running the following commands in the terminal. 'export' is unix-speak for 'set the environment variable'

```
export cp4ibasepath=<<The URL that is shown on your screen under API base URL>>
```

(Make sure you include the https:// part as well)

```
export cp4iuser=<<username on the screen>>
export cp4ipw=<<password on the screen>>
```

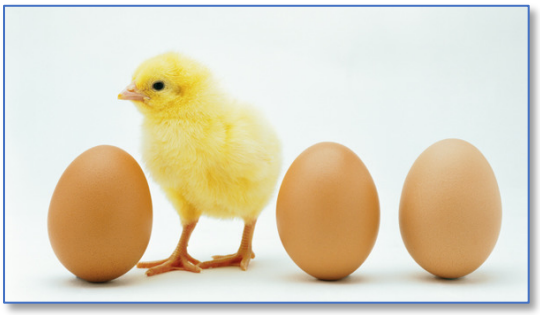
To see the password, click the 'Eye' icon. To copy it to the clipboard, click the double-square 'copy' icons.

12 Running the tests and results:

We have two test cases here:

12.1 Test 1: “Chicken Picture” – demotestchicken.sh

This test sends a picture with a chicken – there is no car in it so we should get an error
The chicken picture is in the chicken.jpg file you downloaded – you can check it’s correct if you like.



To run the test, type (including the first dot and slash) `./demotestchicken.sh`

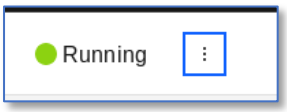
We’re going to send this request:

```
{"Name":"Vernon Barker",
"eMail":"to@epiope.my",
"LicensePlate":"tepuru",
"DescriptionOfDamage":"58",
"PhotoOfCar":"<<Base64image>>”,
,"ContactID":"8897796795006976"}
```

The expected response is something like:

```
{"error":
{"statusCode":400,
"message":"There is no car in this image, please resubmit"}
}
```

If you get ‘unexpected end of file’, double check that your API flow is started!



12.2 Test2: “Subaru SUV Picture” – demotestcar.sh

This test sends a picture with a Subaru SUV in it. There is a car in it so we should not get an error.
The car picture is in the car.jpg file you downloaded.

To run the test, type (including the first dot and slash) `./demotestcar.sh`

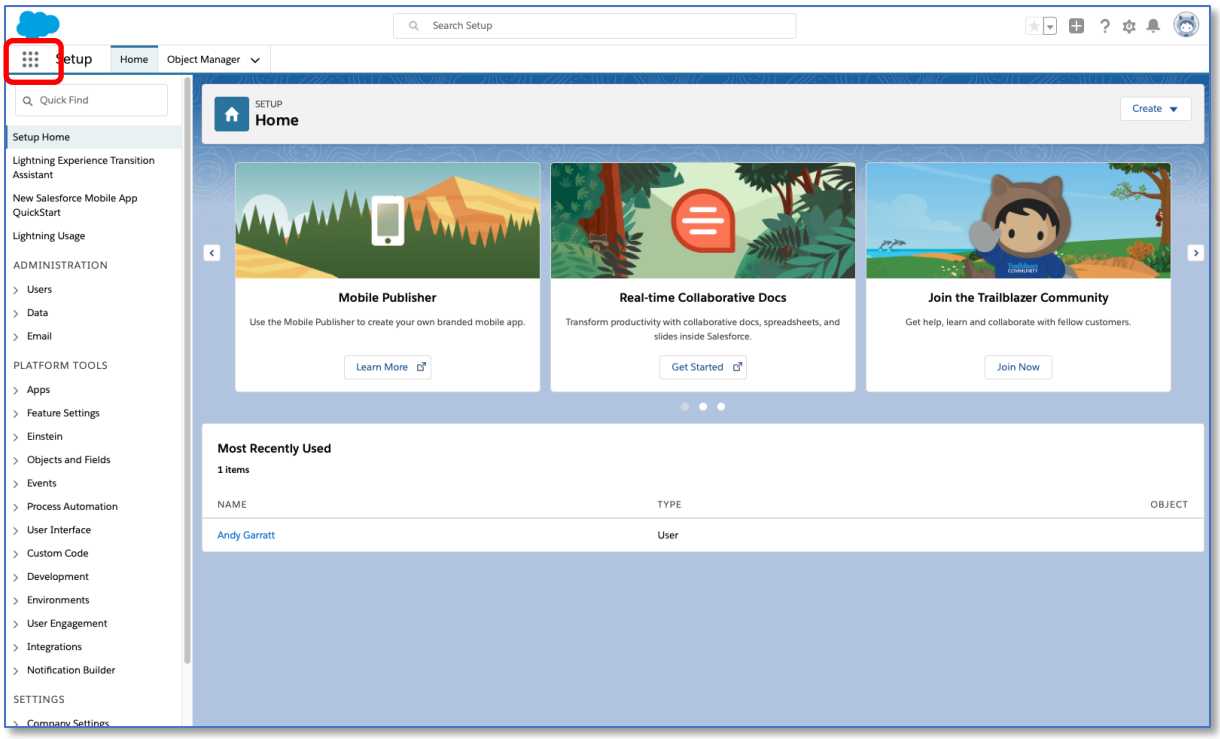
The request is:

```
{"Name":"Derek Subaru",
"eMail":"SubaruDerek@example.com",
"LicensePlate":"SUBARU1",
"DescriptionOfDamage":"You cannot see it from the outside but
the engine will not start any more. This car is rubbish and I
hate it. Fix it quickly or I will sue!",
"PhotoOfCar":"<<Base 64 picture>>”,
"ContactID":"8897796795006976"}
```

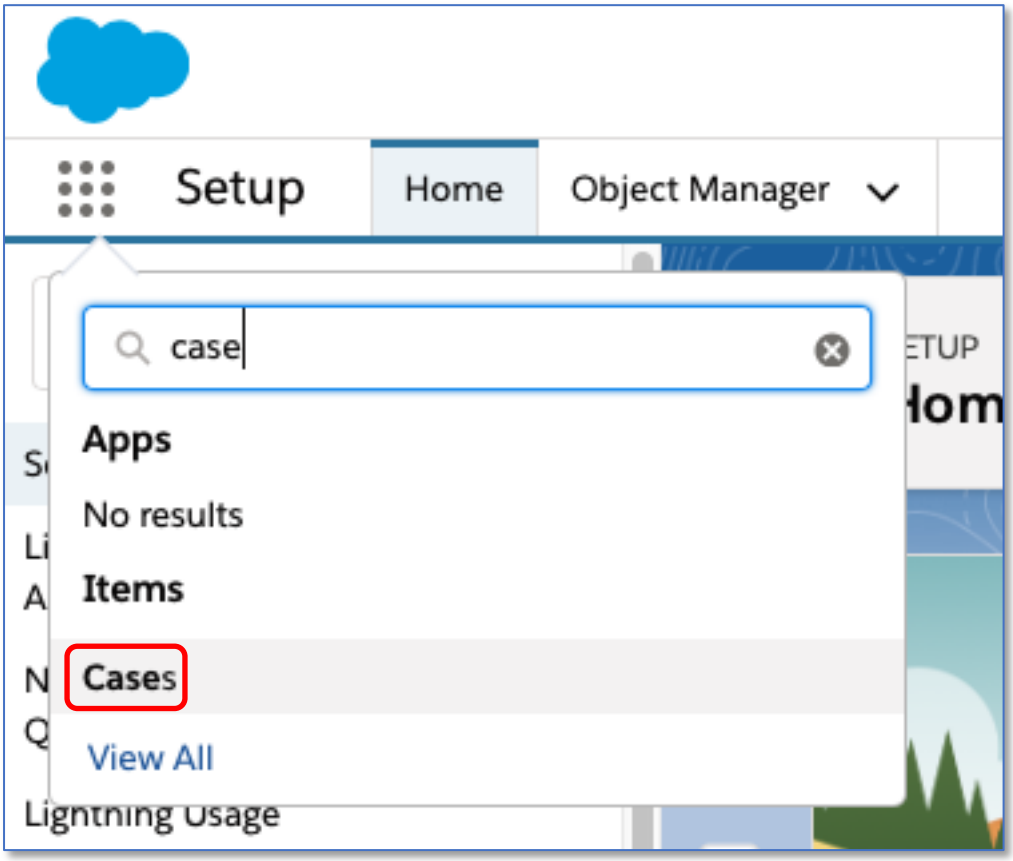
The expected response is:

```
{"CaseReference":"5003z000025uVRSAA2",
"EstimatedBill":300,
"EstimatedDays":3,
"LicensePlate":"SUBARU1",
"Name":"Derek Subaru",
"eMail":"SubaruDerek@example.com"}
```

Note how this has created a case in Salesforce – you can go to Salesforce and see the case yourself

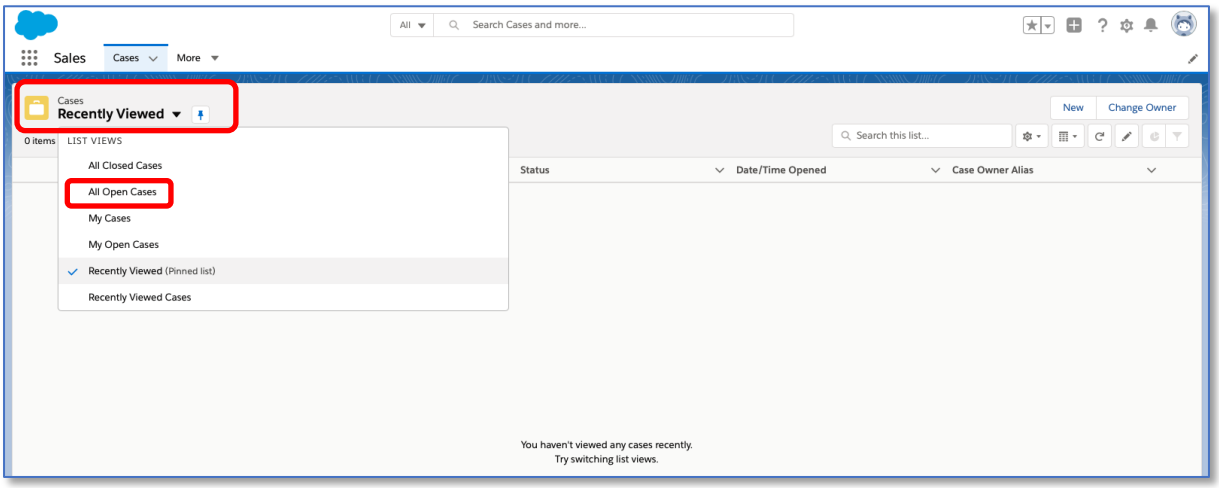


Click the 9 dots at the top left:

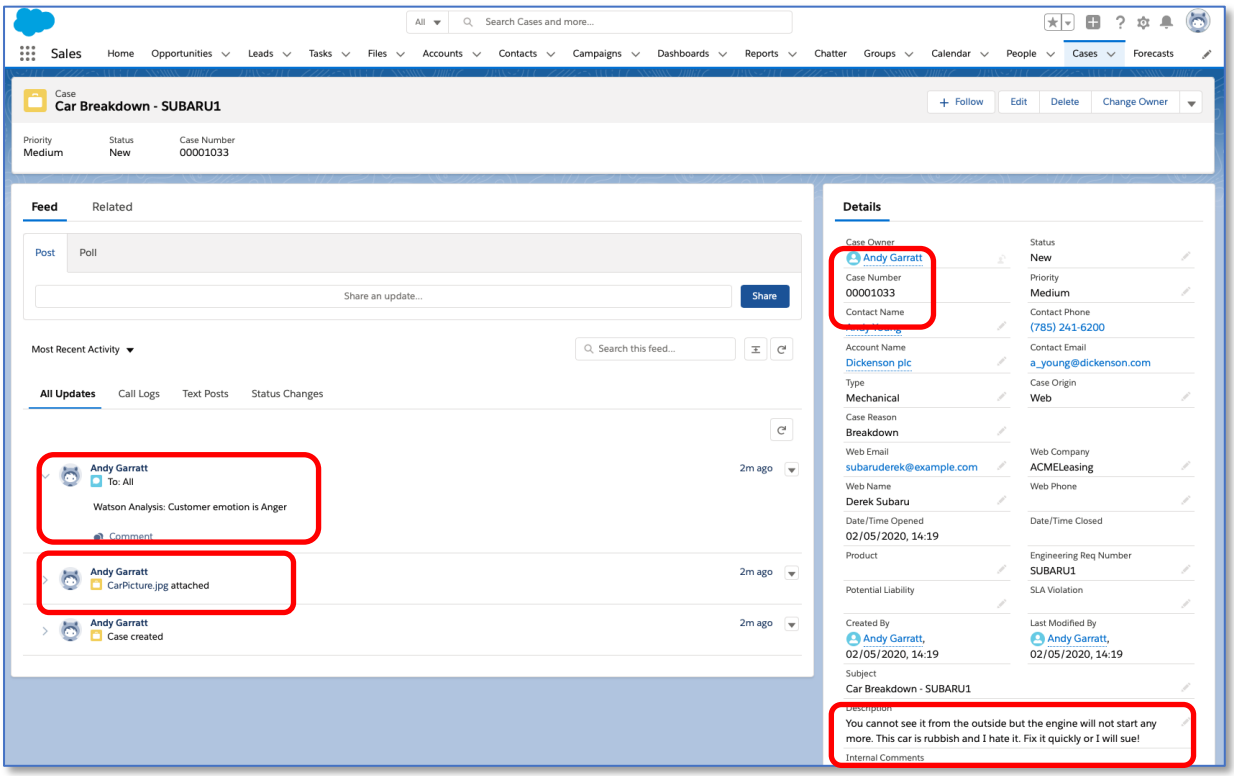


In ‘Search Apps and items, type ‘case’ then click ‘Cases’

Don’t panic that it looks empty! Notice the filter ‘Recently Viewed’ – click this to pull it down and select ‘All Open Cases’



Then you can see your case – with the Subaru photo in it!



Click on the ‘CarPicture.jpg’ to see the attached photo of the car as below:

13 Deploying the Integration flow to CP4I RunTime via the App Connect Dashboard

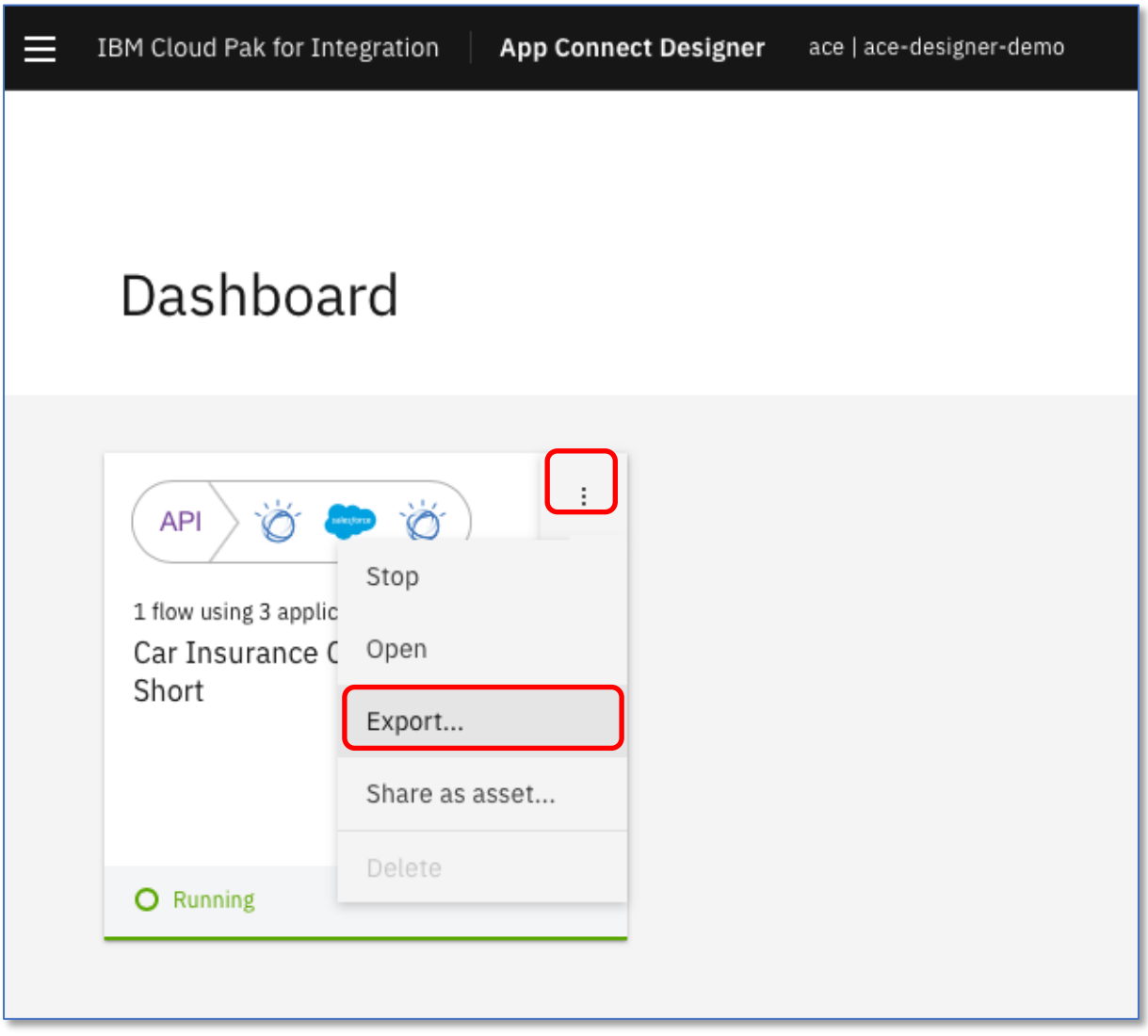
We've now got our flow running in the designer and we've tested it – now we need to deploy it 'for real' on the cloud pak runtime.

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.

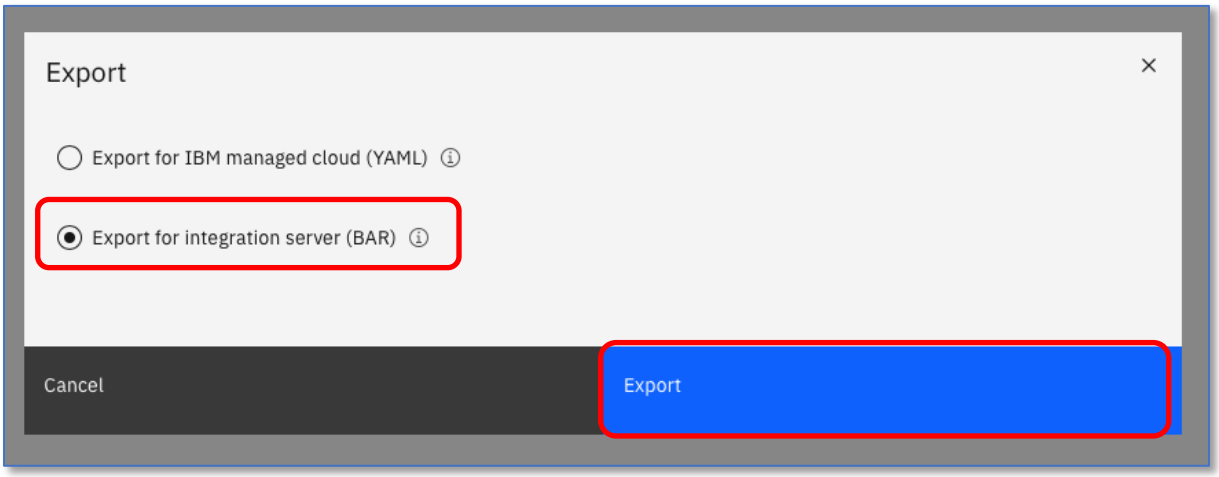
When we deploy, it will create a 3 HA replica container pods running on OpenShift – automatically.

13.1 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 'Export for integration server (BAR)' and click 'Export'



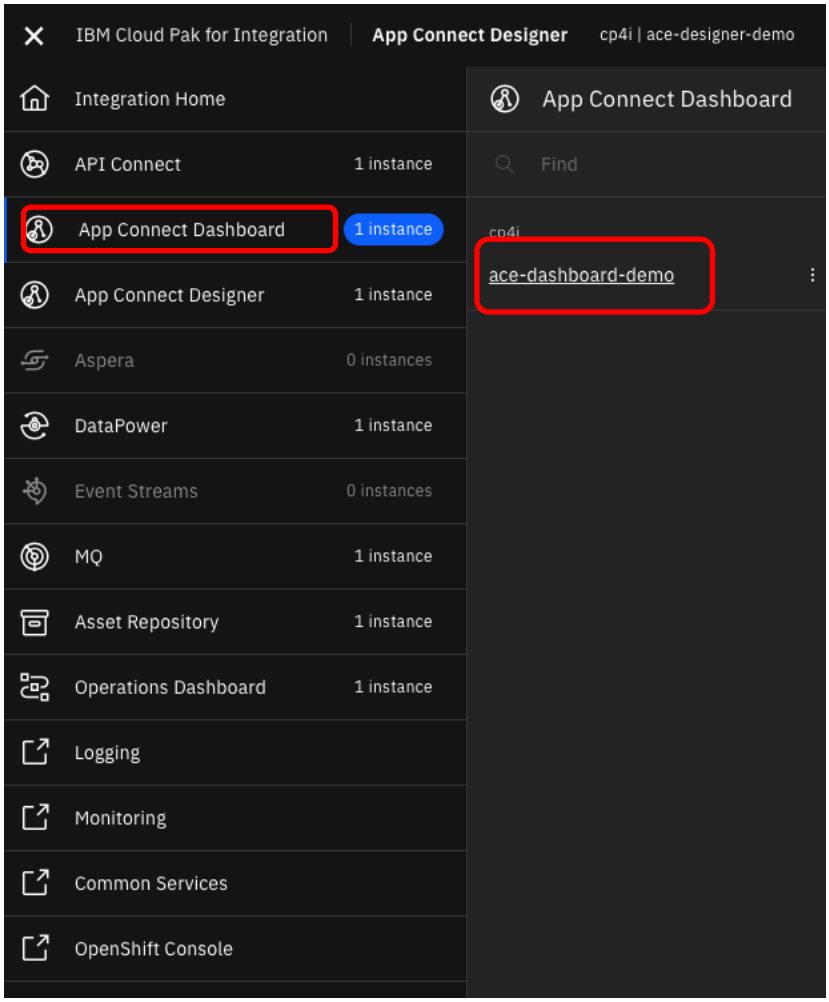
The browser may prompt you for a download location – otherwise it will place the 'Car_Insurance_Cognitive_API_Lab_Short.bar' file in the `Downloads` directory.

The 'Export for IBM managed cloud' is the YAML source for the flow. It's what we exported to git. You can import and export .yaml flows as you wish – they are source, not executables.

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

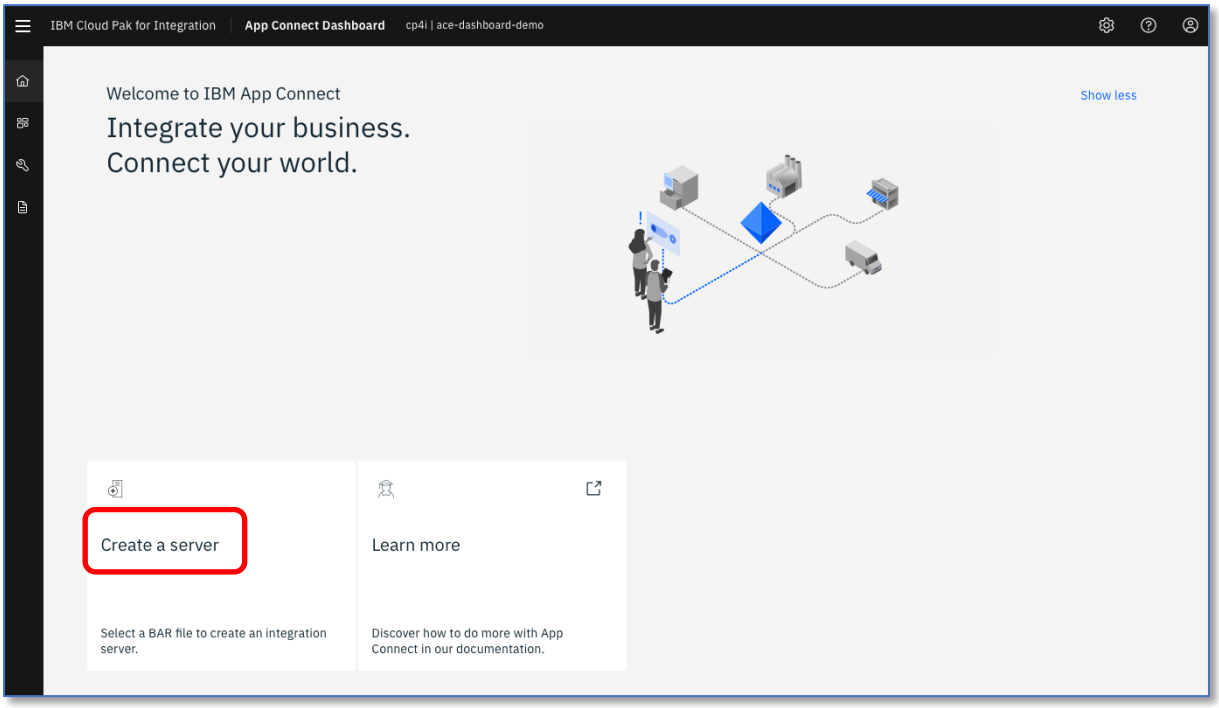
13.2 Navigating to the App Connect dashboard and importing the .bar file

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 – at the moment, there are no integrations here:

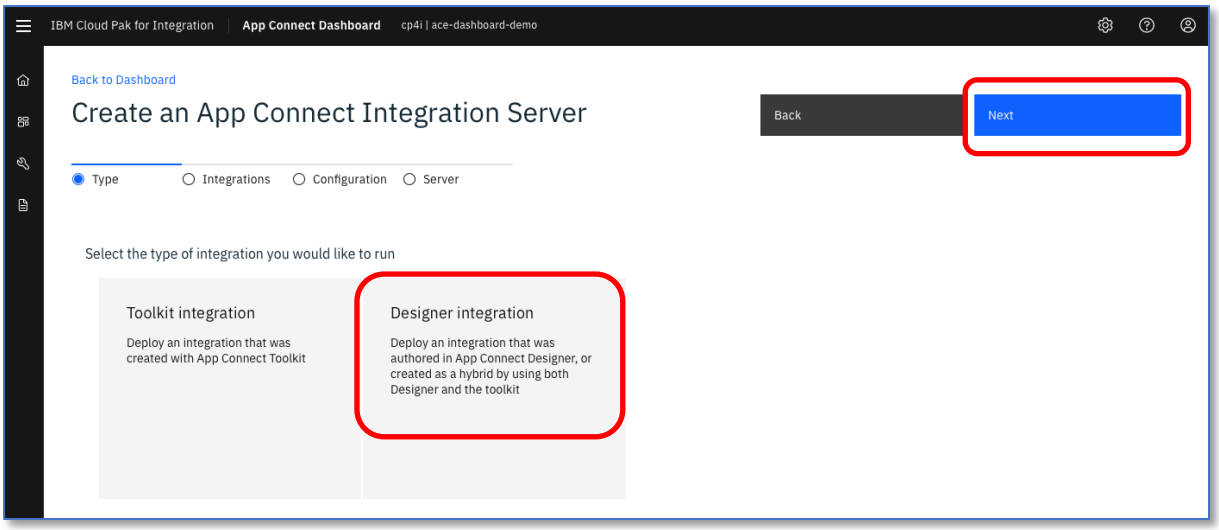
Click 'Create A Server'



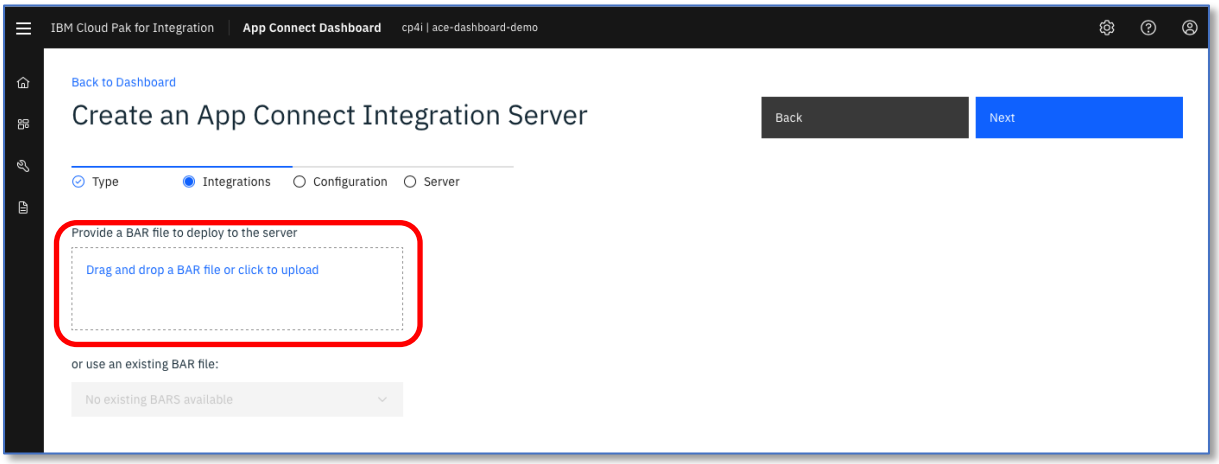
We need to create an integration server to run our integration. An integration server is a Kubernetes pod which has the containers needed to run our .bar file.

(If you're not familiar with Kubernetes terms, don't worry. We are going to deploy our integration in a multiply-redundant, scalable, highly available way)

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'

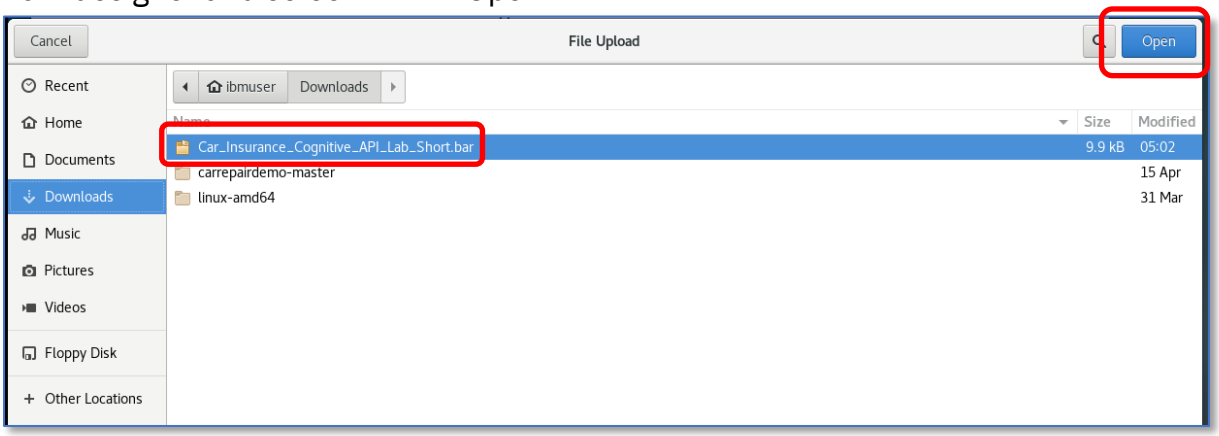


You're now prompted to upload the .bar file you exported before:

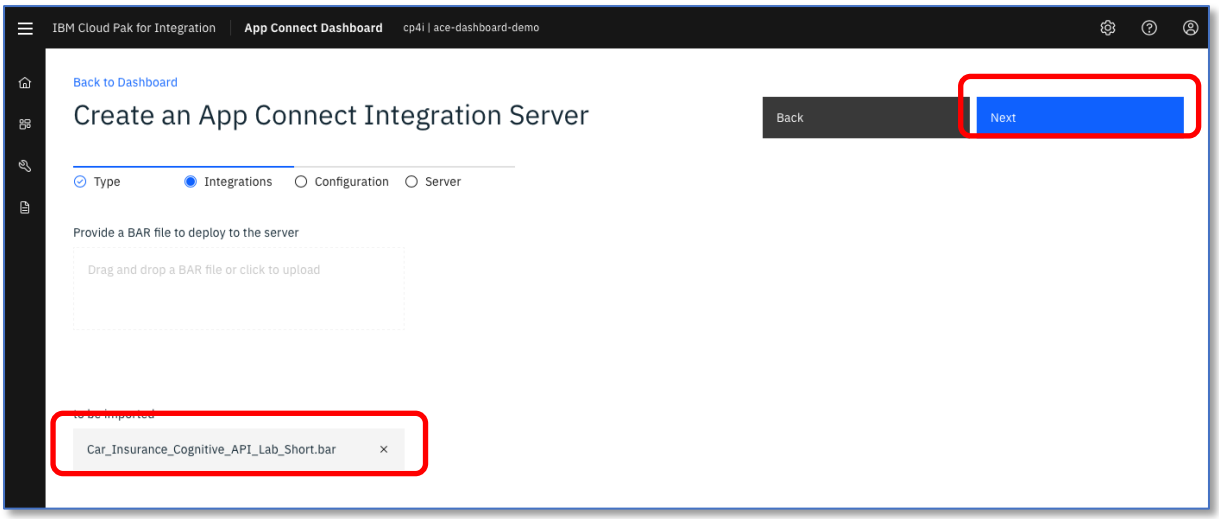


In the dialog box, click 'Drag and drop a BAR file or click to upload'

Browse to the location of the 'Car_Insurance_Cognitive_API_Lab_Short.bar' file that you exported from designer and select it with 'Open'

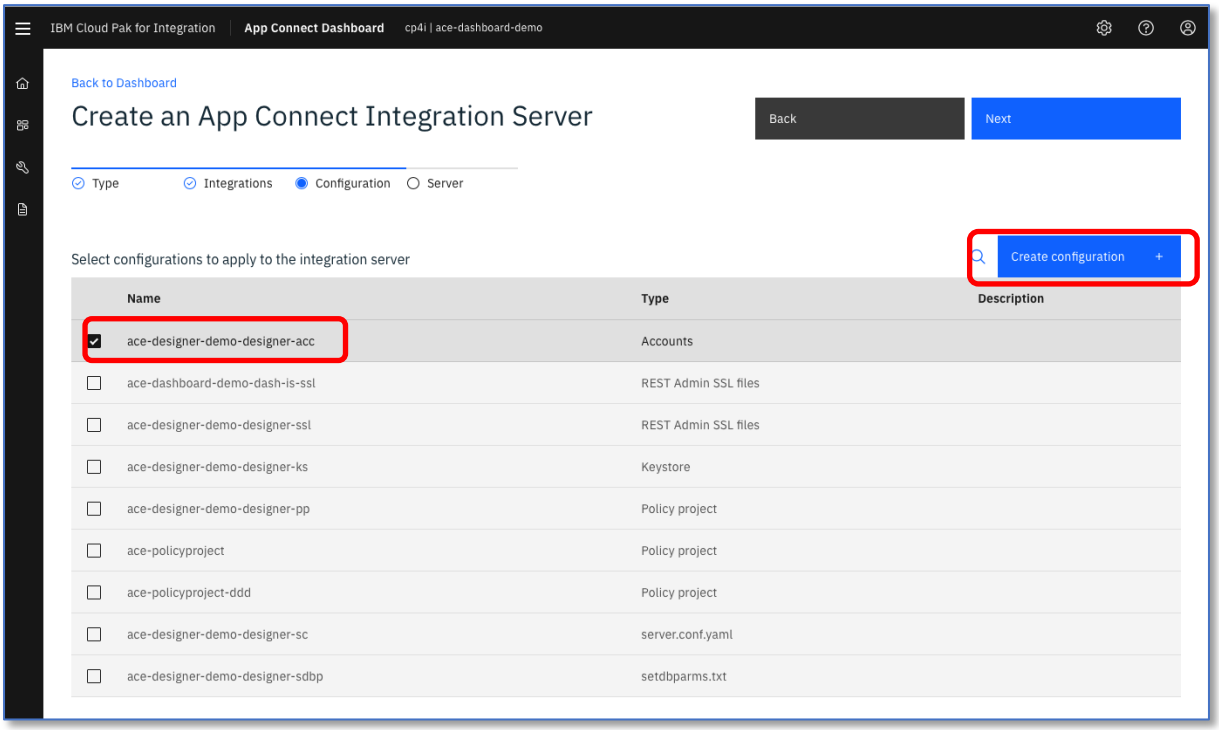


– then click 'Next' on the dialog as below:



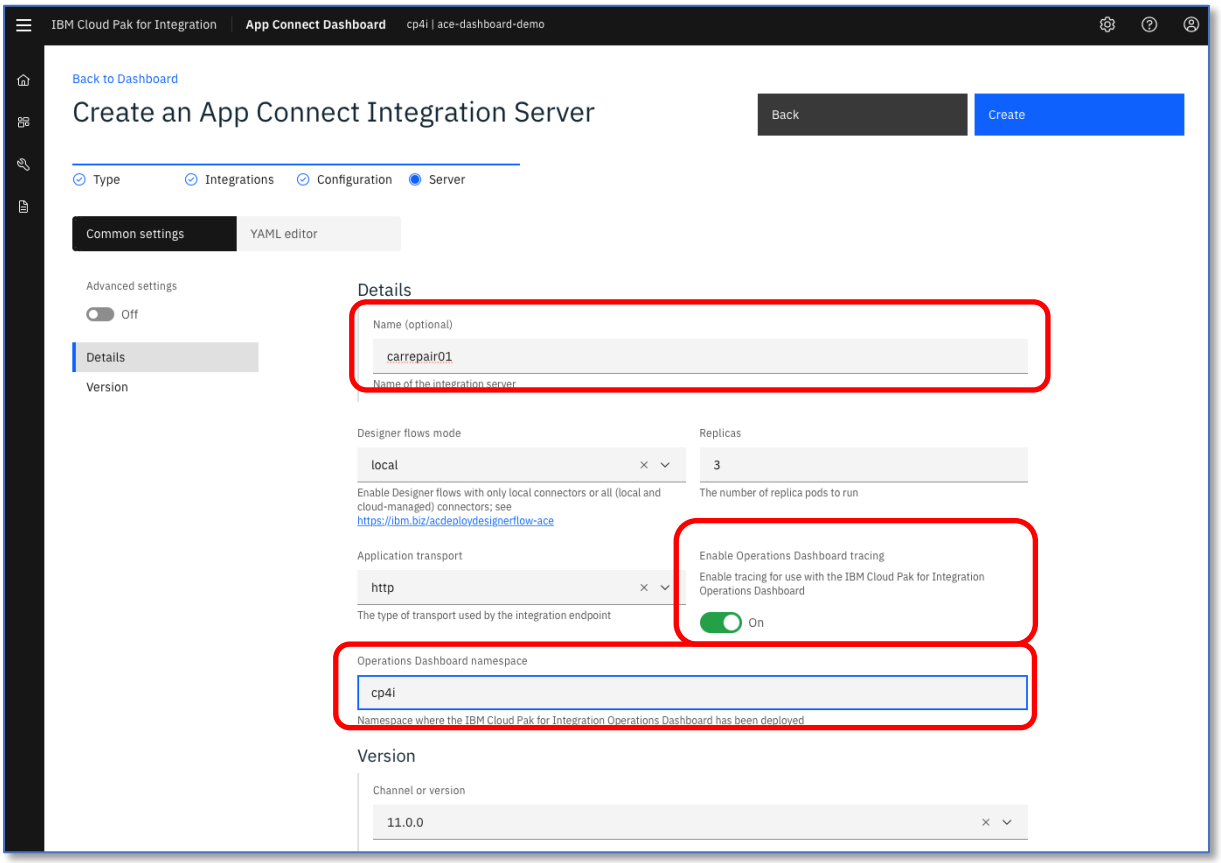
This is new from the 2020.2.1 release. The 'Download Configuration Package' has gone. Configuration data is now stored in configurations – and what's more, we can re-use the Connector Account configurations we already set up in Designer.

We need to choose which configurations we want. We want the 'ace-designer-demo-designer-acc' Accounts. Select that as below and click 'Next'
(note: You don't need to click 'create configuration' here)



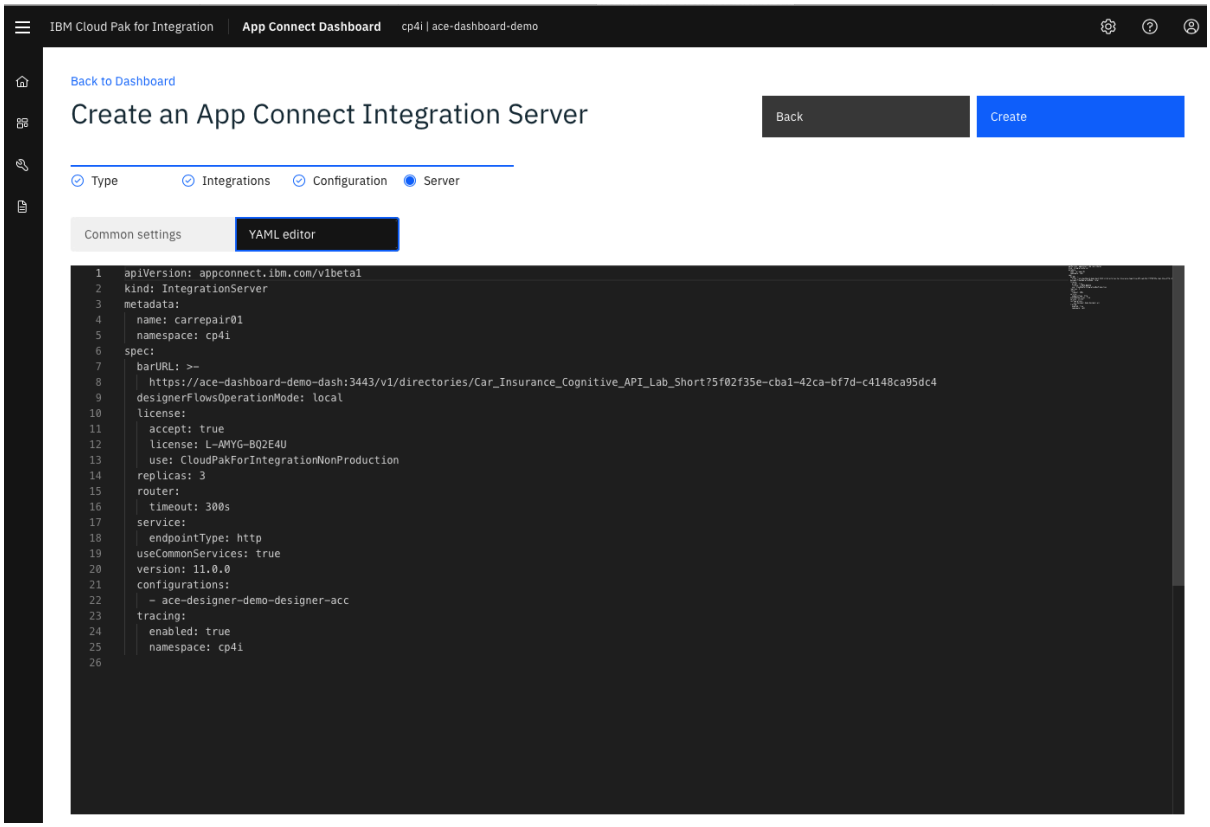
You're now on the last screen! Enter a name for our integration server – we're going to use '`carrepair01`' – if you see errors with your name, one key thing is that the name must be lower case.

Set 'Enable Operations Dashboard tracing' to 'On' – we want to be able to use tracing when we test our flows. The Operations Dashboard namespace is 'cp4i' (the same name you gave for your 1-click install)



IMPORTANT: At this point, there is a known issue in 2020.2.1.1 of the cloud pak – we need to apply a patch to the integration server!

Click on 'YAML Editor" and you'll see the YAML configuration that the GUI is creating:



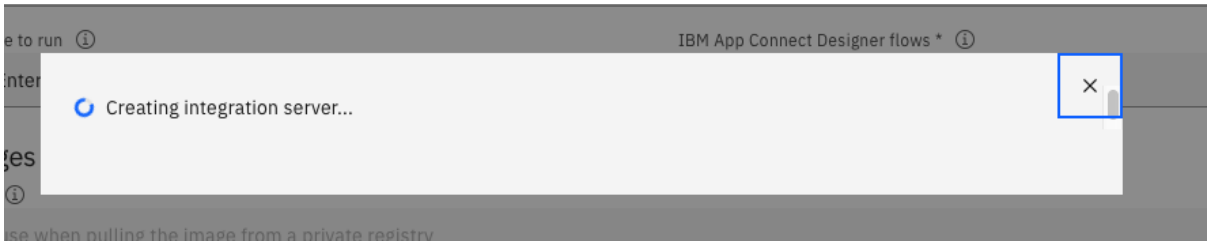
You'll need to add in the following code at the bottom. Note that this is YAML and the spaces are VERY important – for example there are two spaces before the word 'pod'

```
  pod:
    containers:
      runtime:
        image: cp.icr.io/cp/appc/ace-server-prod:11.0.0.9-r3-tfit33963-amd64
```

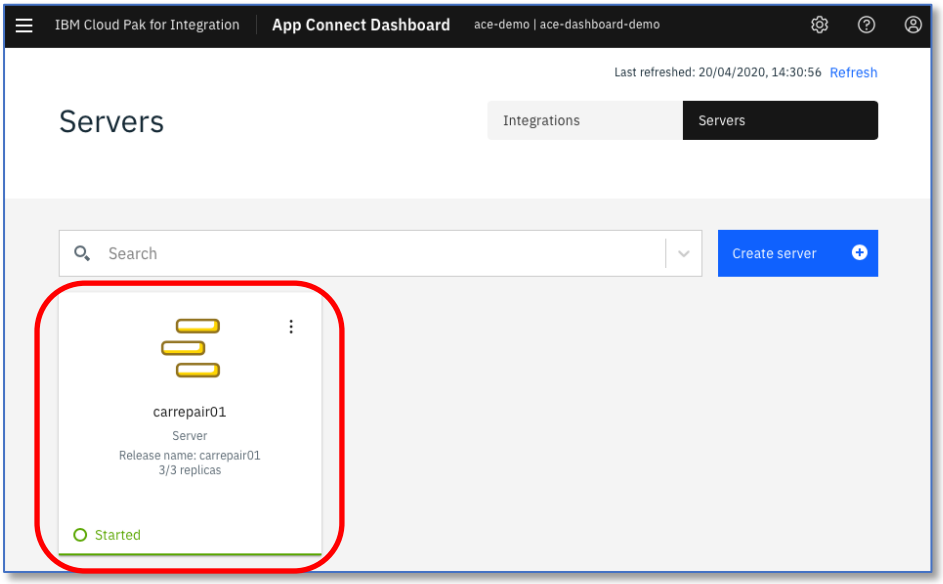
You'll need to ensure it's added with exactly the right spaces – like below:

Now click ‘Create’

You’ll see:

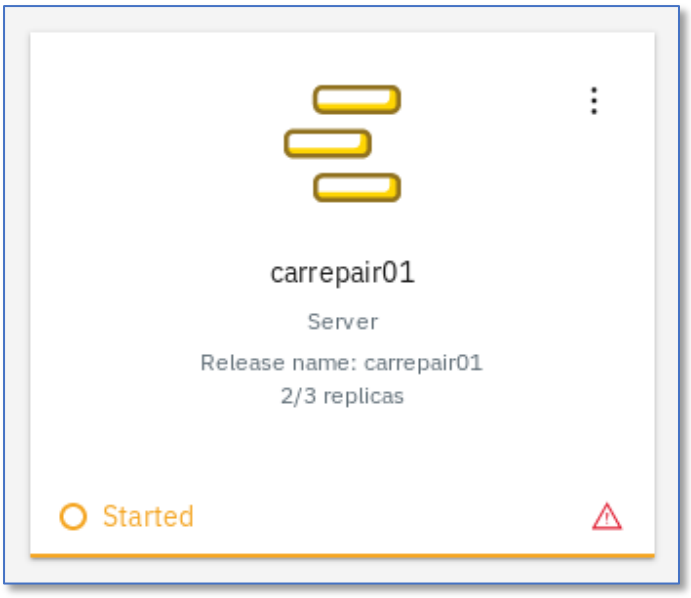


When you get refreshed screen, you should see this in your dashboard:



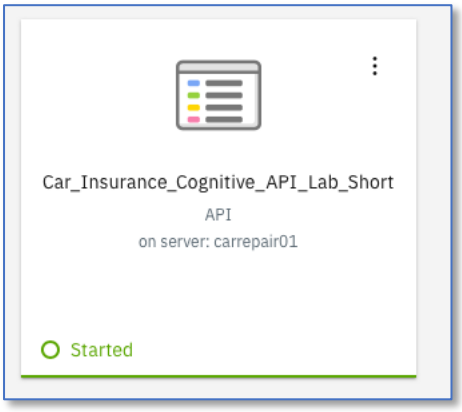
DON’T PANIC! You may find that you initially see what looks like an error – this is the cloud pak spinning up 3 pods of the integration server – it won’t show a green tick until all the pods are running. Give it a couple of minutes or so and refresh your browser. You can leave it in this state for a bit and get on with the next part of the lab if you wish.

Tip: If you know OpenShift, you can go to the deployments and see it spin up the pods.

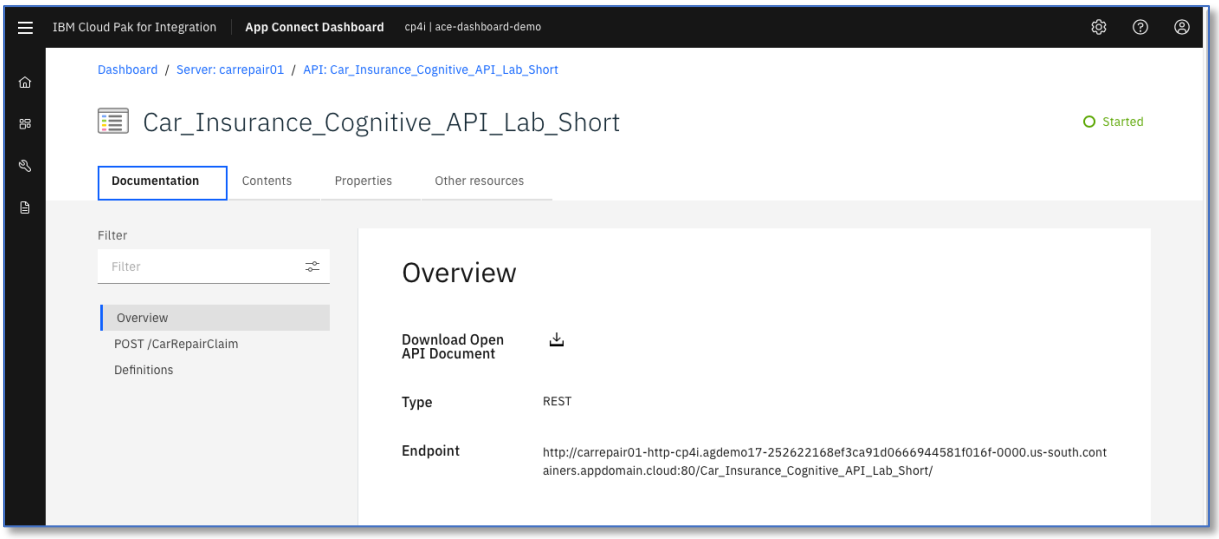


At this point, the integration is running on the cloud pak however, it can't actually connect to anything – it doesn't have any credentials to connect to Salesforce or Watson.

Click the tile and you'll see the following:



Click again, and you'll drill down further and see the following:



You can see the REST operation, the base URL and you can even download the OpenAPI (also called swagger) document.

We can test this if we wish! (It's not compulsory...). Use the same curl scripts that we used for designer, but change the URL to point to the REST API Base URL given in the UI.

(Not that you'll need to make sure you have the `/CarRepairClaim` at the end of the URL).

This means we now have our integration running on our cloud pak server.

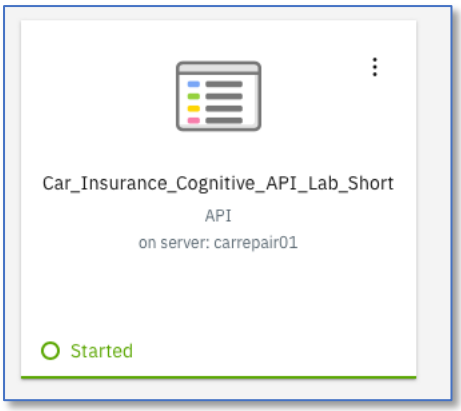
Now would be a good time to test it again – good job we have re-usable automated test scripts!

(Note: If you're familiar with earlier versions of the cloud pak, you may be wondering where all the 'download integration package' parts have gone – these have been replaced by the configuration resources – it's now much easier!)

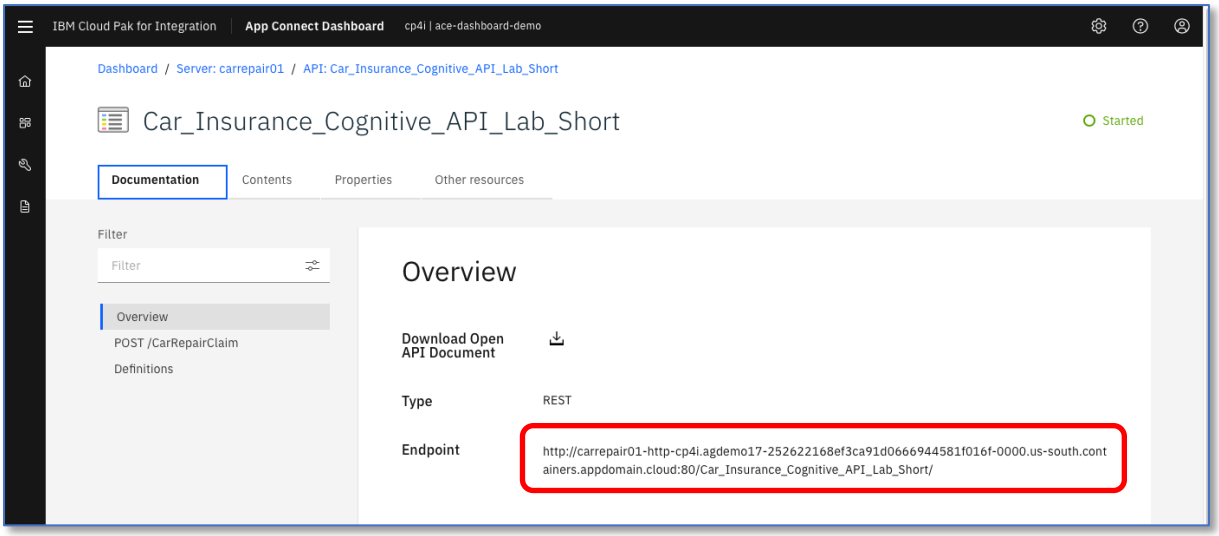
13.3 Testing the flow on the CP4I runtime:

We'll use the same scrips we used to test designer – we'll just change the variables to point to the ICP4 runtime.

Click on our carrepair01 integration tile on the dashboard – you'll see this:



Click on this and you'll see:



The 'Endpoint gives us the base URL variable we need.

We just need to set the cp4ibasepath variable correctly:

In the terminal, run (All one line)

```
export cp4ibasepath=<<The Endpoint URL from the screen, including the http part>>
```

To run the tests, we do the same as we did last time:

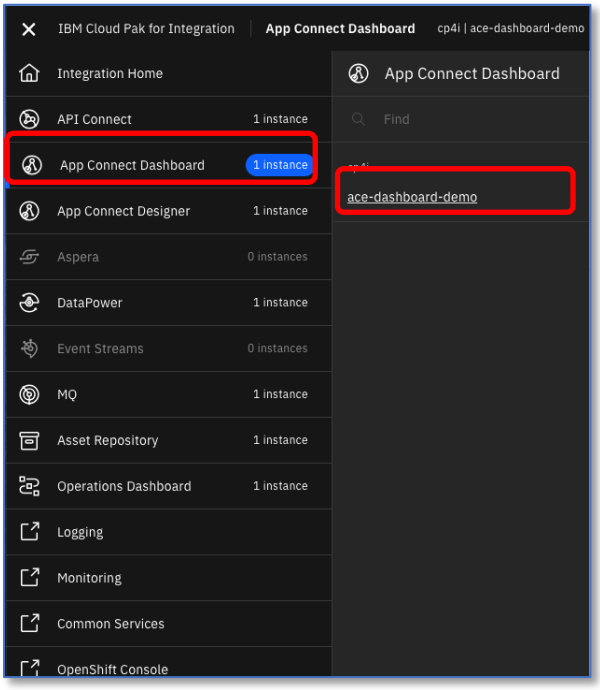
```
./demotestchicken.sh
./demotestcar.sh
```

(if you can't find the scripts, check you're in the right place – they may be in the home (your user name) directory, or in the Downloads directory – either is fine.

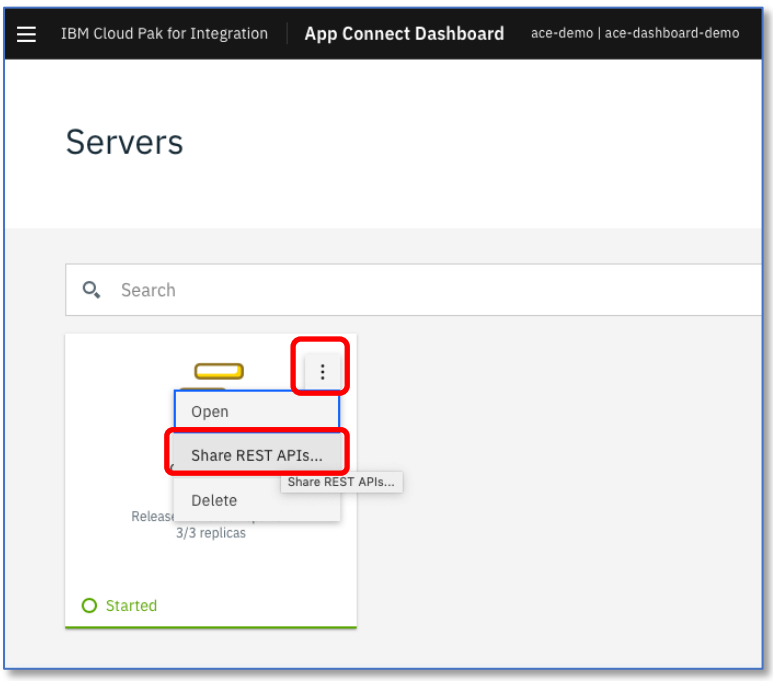
demotestchicken.sh will give an error as there is no car in the picture, but demotestcar.sh should create another case in Salesforce – go check if you wish.

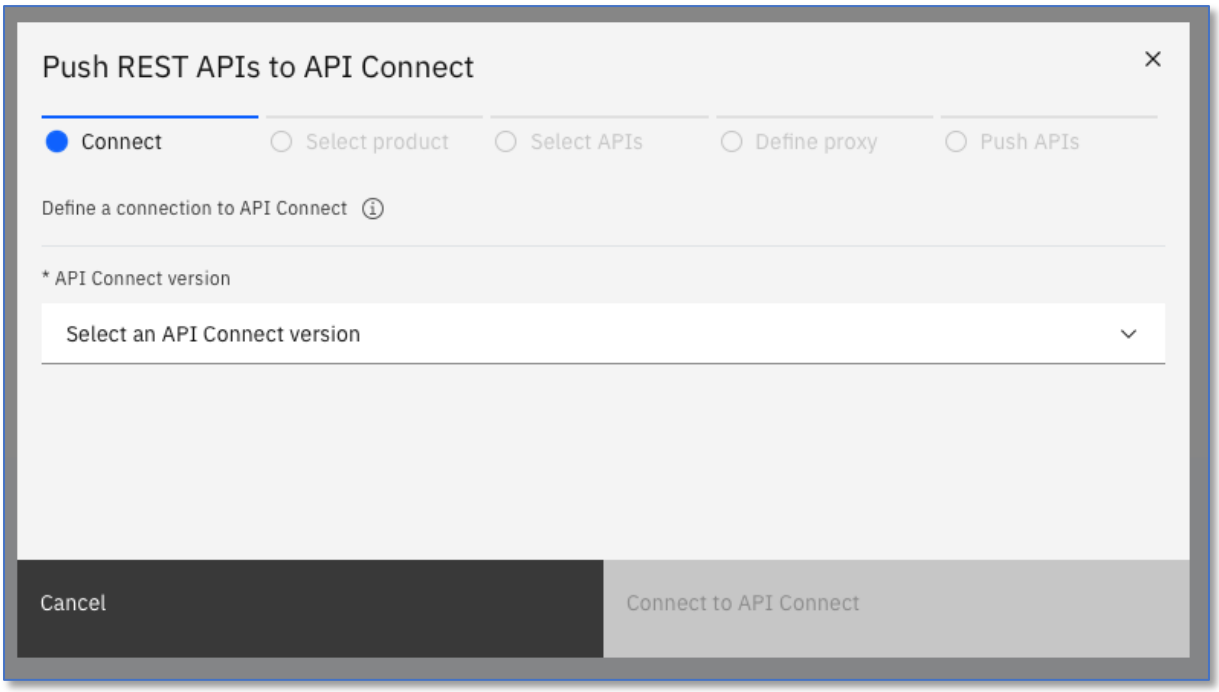
14 Pushing our API to API Connect to be managed:

Go to the App Connect Dashboard:

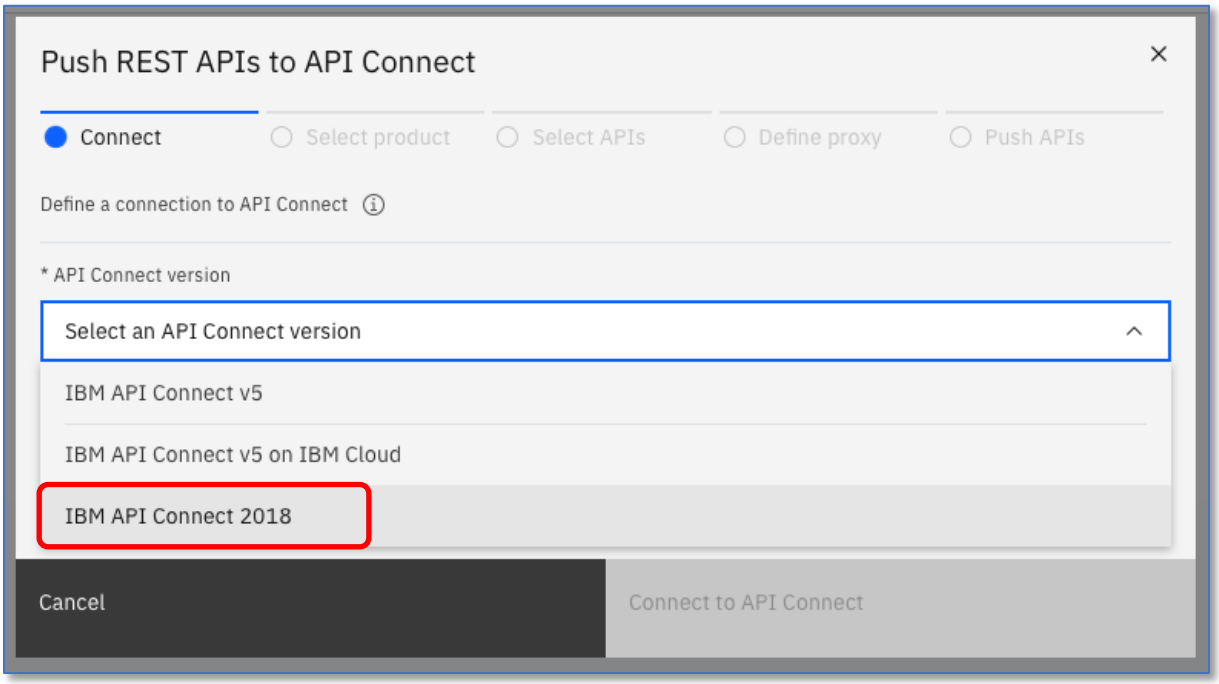


On the Servers screen, on the carrepair01 tile, click the '...' menu and click 'Share REST APIs...'

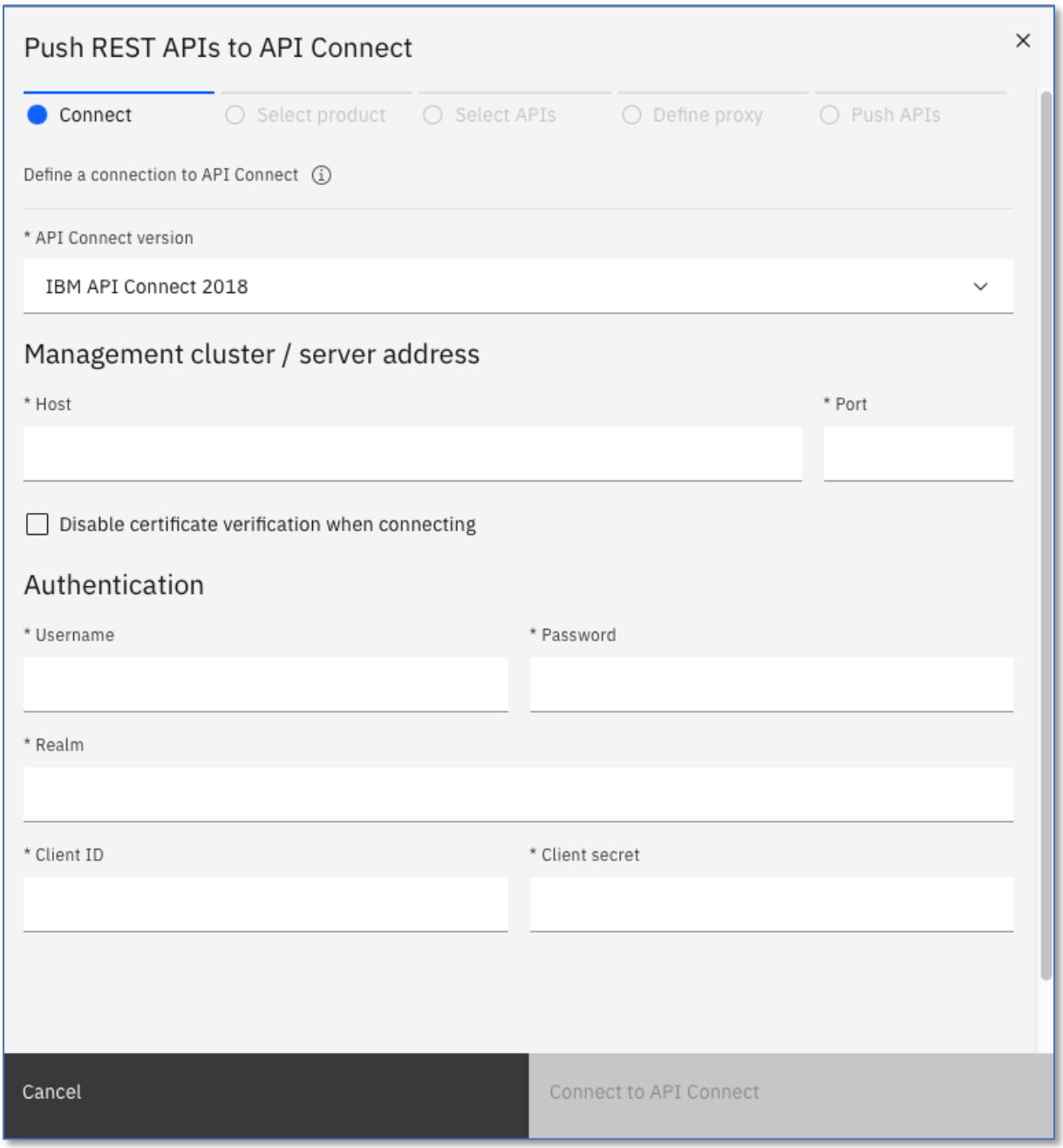




You'll need to be able to specify an API Connect Version – select 'IBM API Connect 2018'



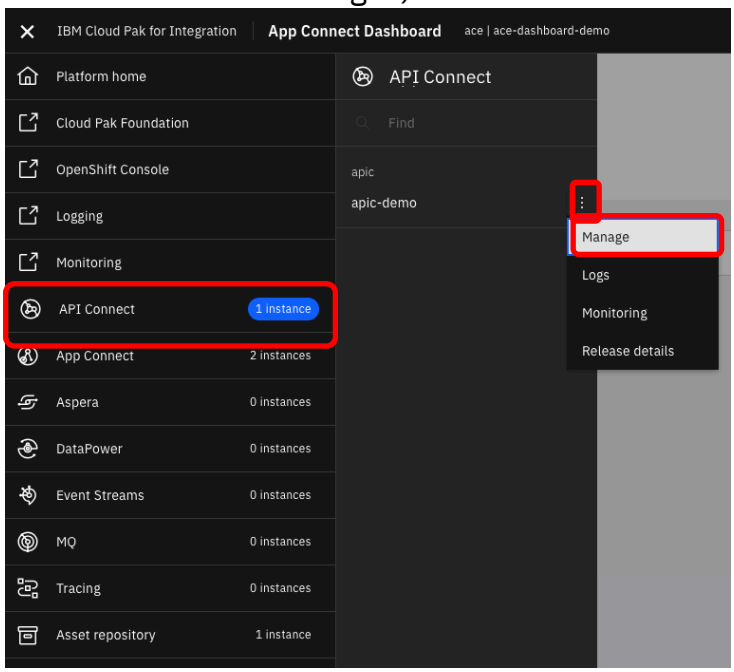
Now you'll see the following:



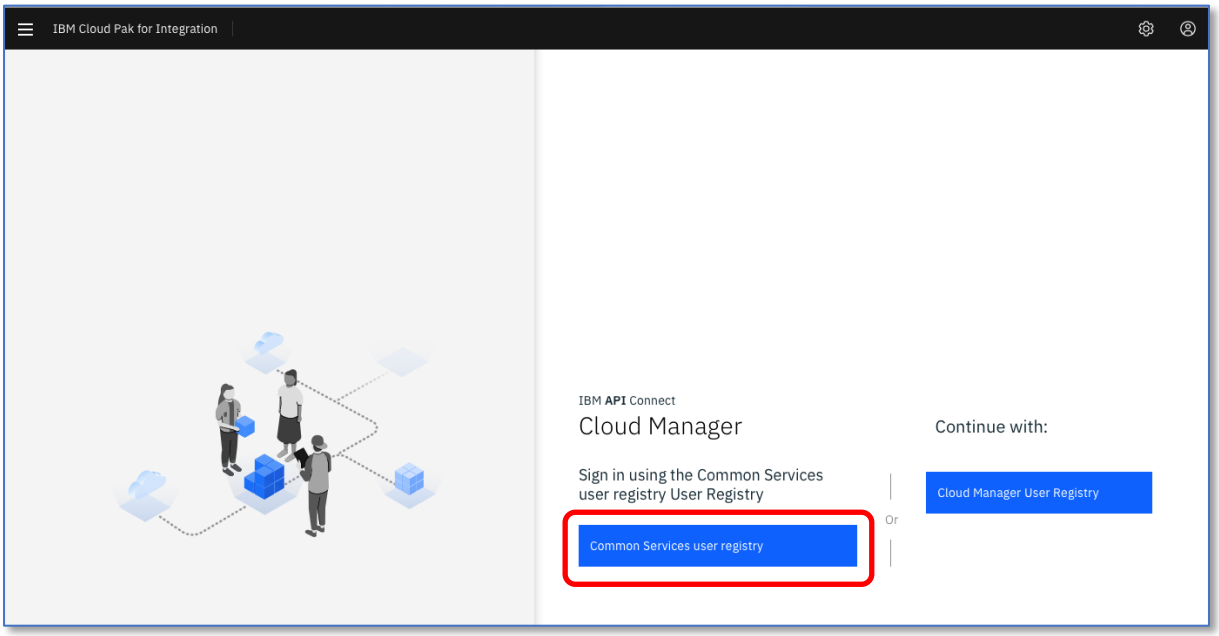
Why do we need to fill in these fields? App Connect can push APIs to API Connect anywhere – on cloud, on premise, on different clusters, inside or outside of the cloud pak, so we need to tell it where to push the API to: Let's look at the fields one by one:

14.1 Host:

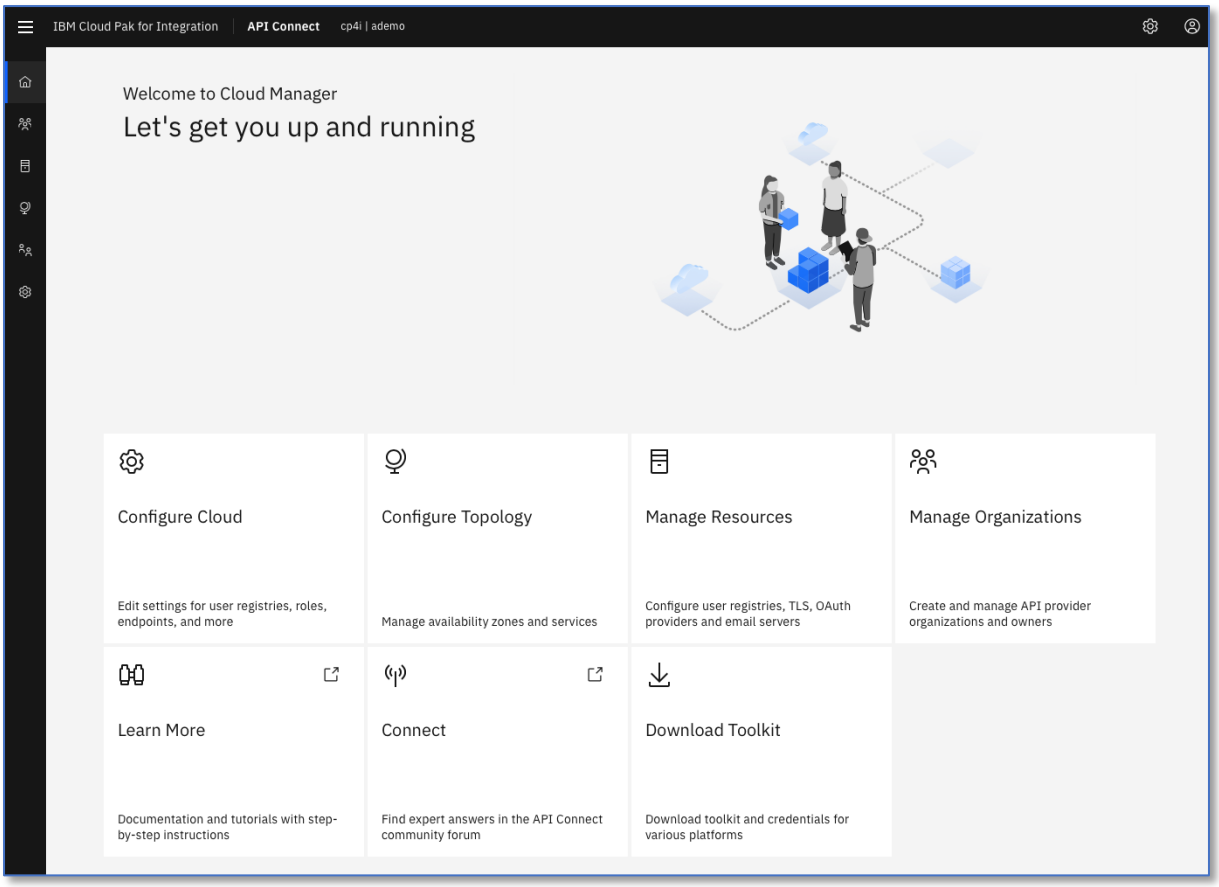
This is the host of the API Connect Management Server. How do we find out what that is?
Go to API Connect Cloud Manager using the menu below: (don't forget to click on the three dots for manage..)



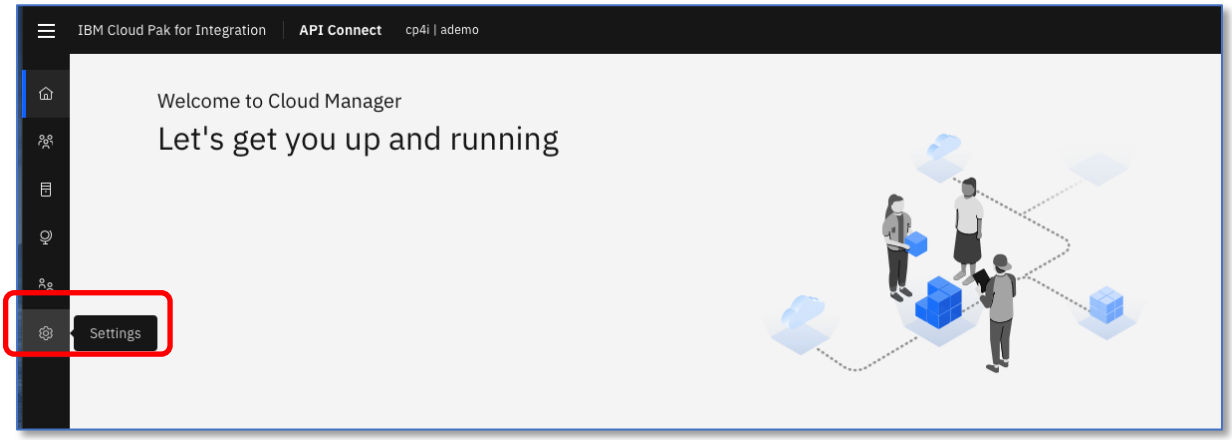
At this point, log in with the IBM Common Services user registry – it should log you in automatically. If it doesn't, the user id (admin) and the password are the same as you used to log into the cloud pak home page.



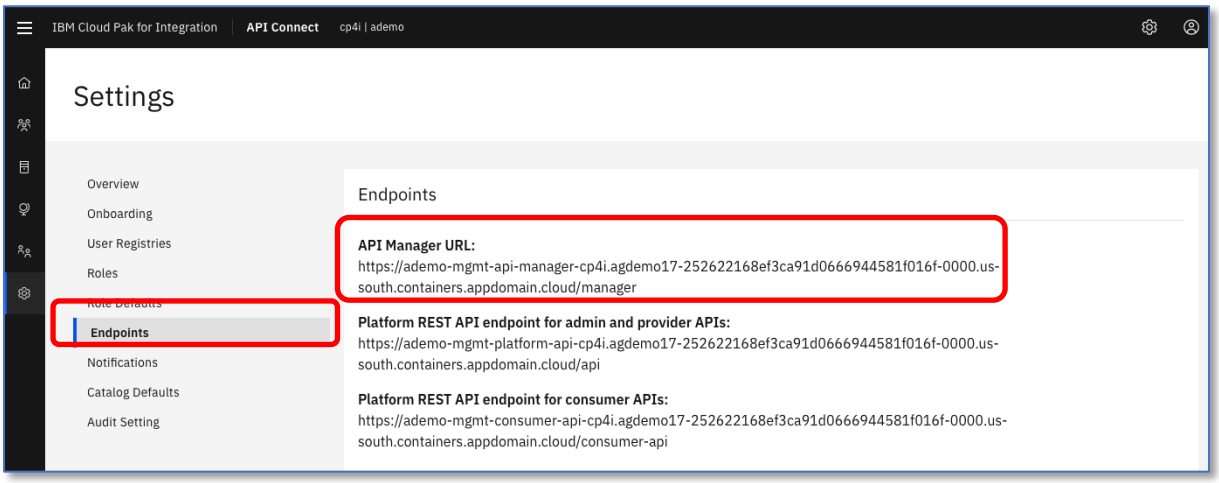
You'll then get to the API Connect Cloud Manager Screen:



Use the ‘Cogwheel’ Menu and Click on the ‘settings’ menu as below:

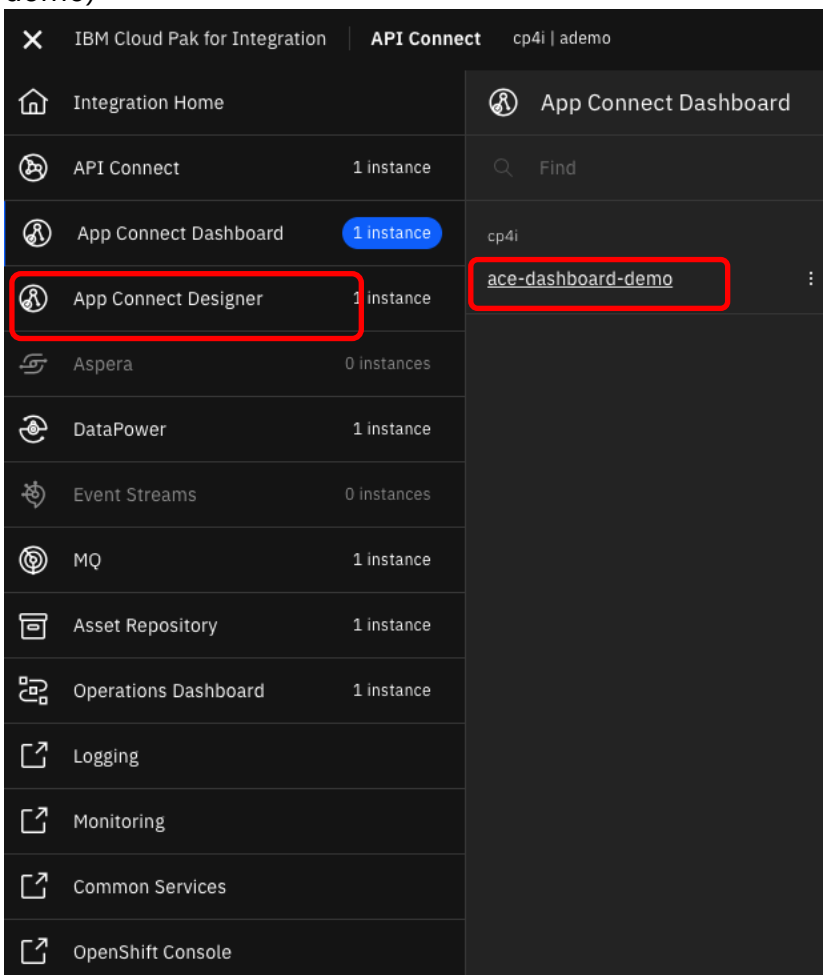


Then click ‘Endpoints’: You’ll see the endpoint you need as the API Manager URL. Copy this value, you’ll need it in a minute.

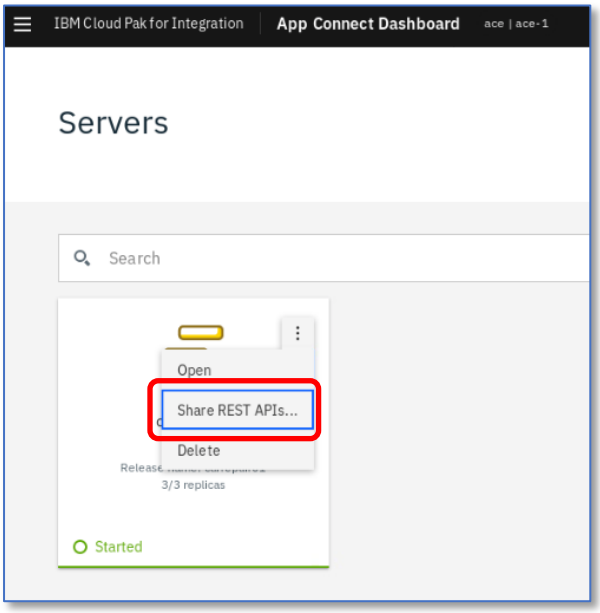


14.2 Pushing the API to API Connect

Go back to the App Connect Dashboard (Menu-> App Connect -> ace-dashboard-demo)

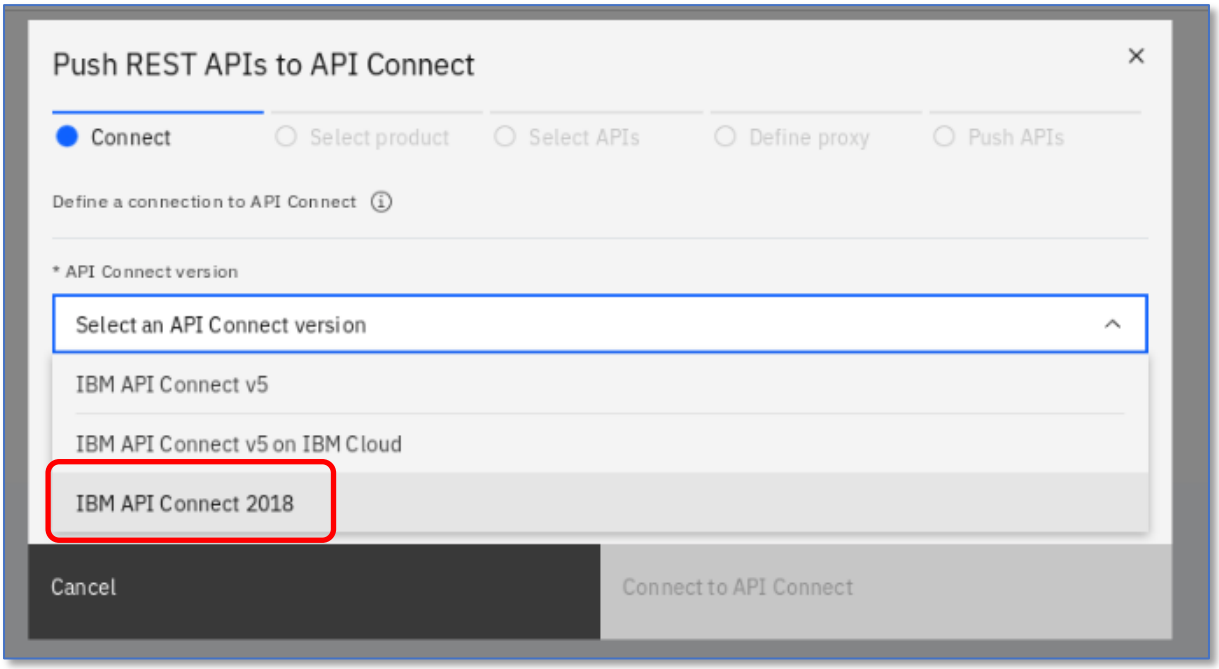


You'll see our integration running. Click on the three-dot menu (...) and choose 'Share REST APIs'



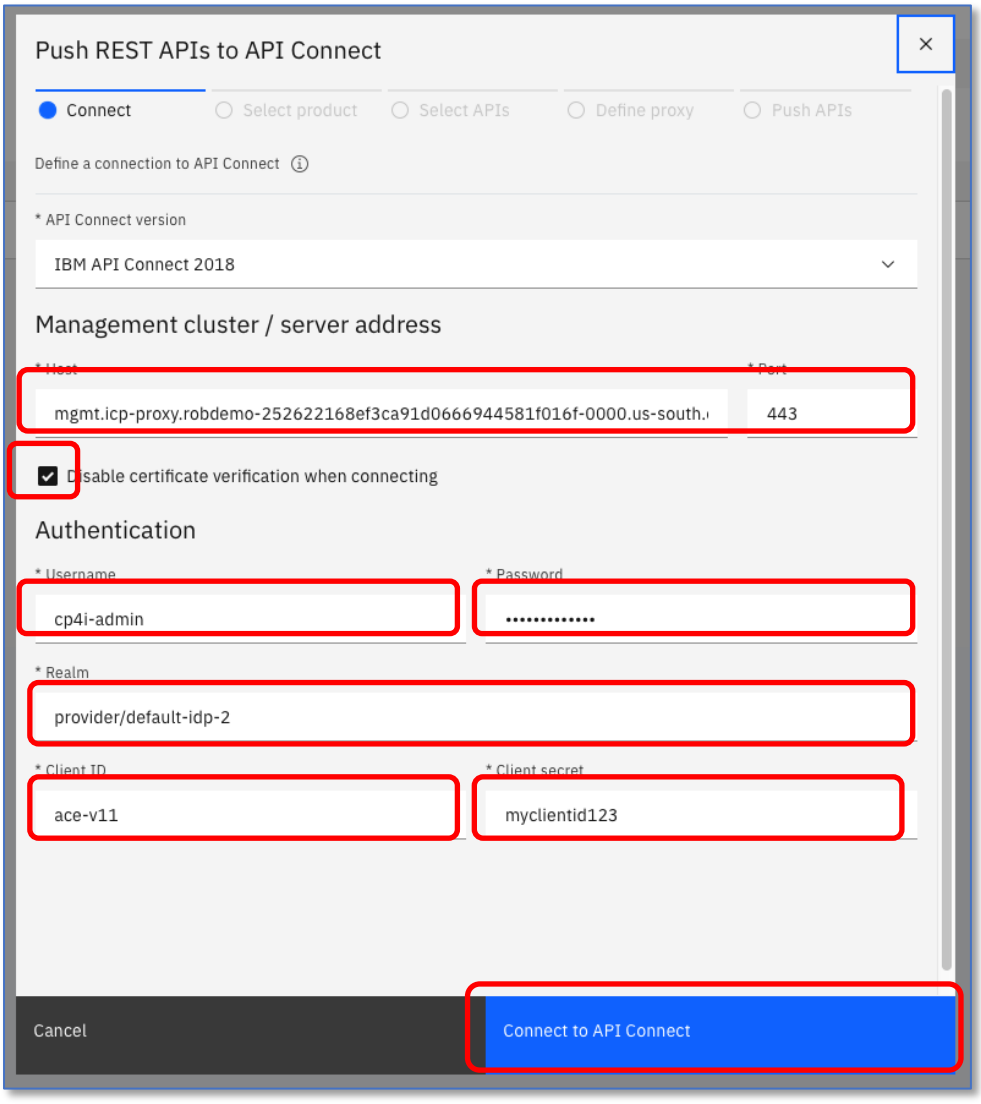
You'll see the Push REST APIs to API Connect dialog box. We're going to take the APIs in App Connect and push them to APIC where they'll be created in the correct provider organisation, catalog and product.

On 'Select an API Connect version', select 'IBM API Connect 2018'



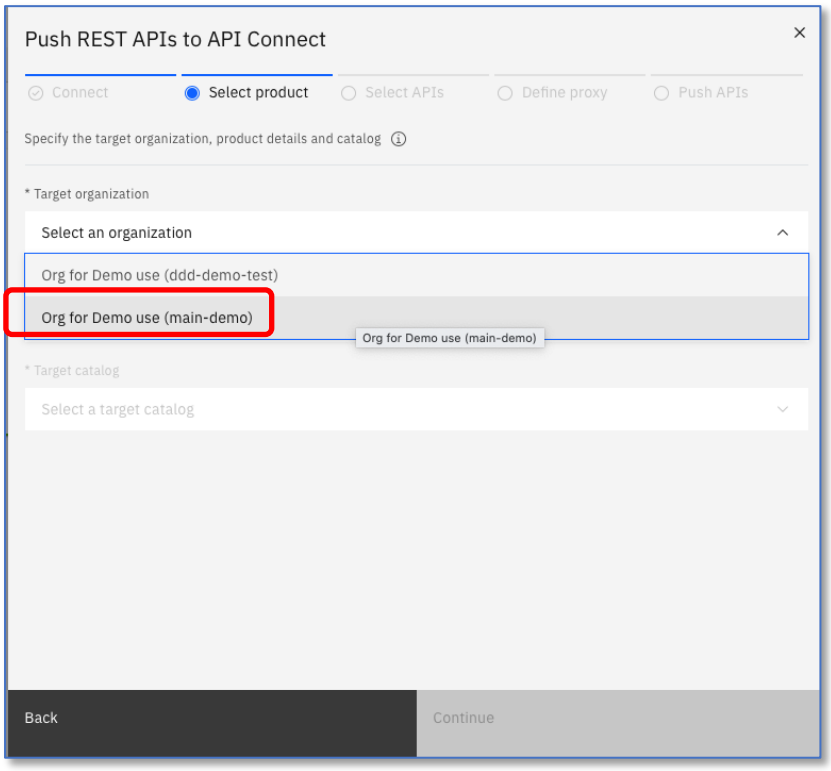
Ok, now we have the dialog box we needed our credentials for. They are as follows:

- Management Server:
  The address saved from the GUI above. Not there is NO ‘https’ and NO /manager at the end. For example, one of ours was `mgmt.icp-proxy.robdemo-252622168ef3ca91d0666944581f016f-0000.us-south.containers.appdomain.cloud`
- Port: `443 (https)`
- Disable certificate validation: `true (ticked)`
- Username: `cp4i-admin`
  (this is set up as part of the 1-click installation ‘install demos’ option)
- Password: `engageibmAPI1`
  (this is set up as part of the 1-click installation ‘install demos’ option)
- Realm: `provider/default-idp-2` (This is the internal name for API Manager User Registry – it will usually be this)
- ClientID: `ace-v11`
- Client Secret: `myclientid123`

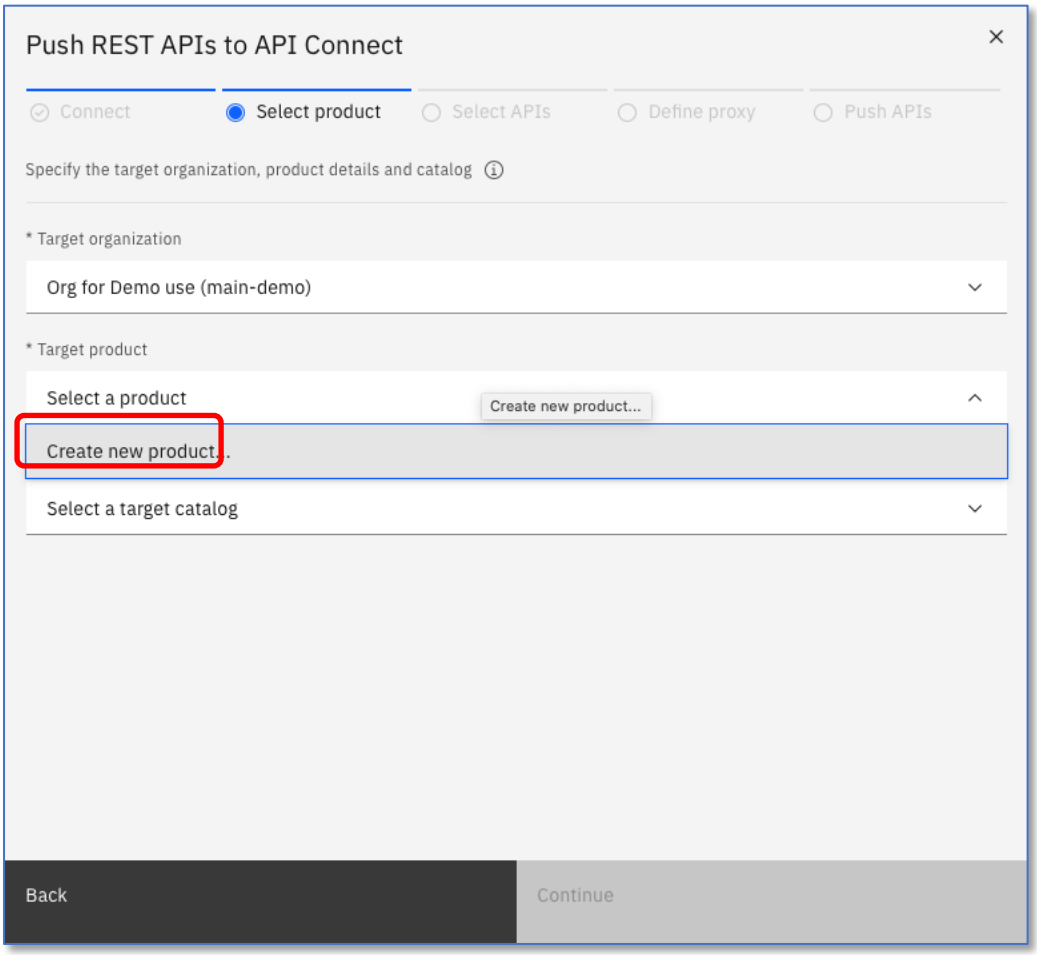


At this point, we're connected to API connect!

Select an organization: Select '`Org for Demo use (main-demo)`'. (This is the provider organization that has been created for you in APIC as part of the 1-click 'install demos' configuration.)



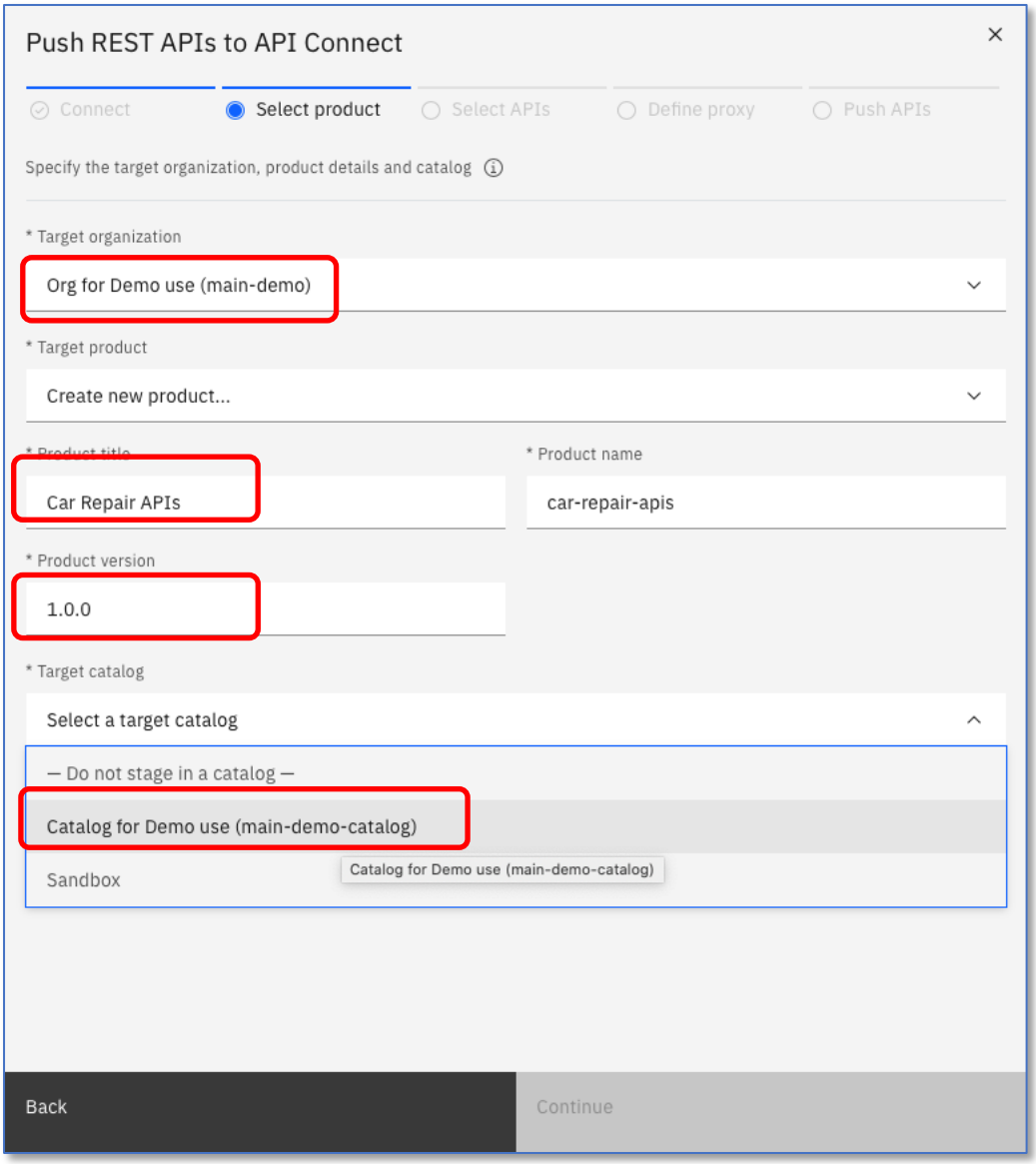
Select a product: ‘Create new product’



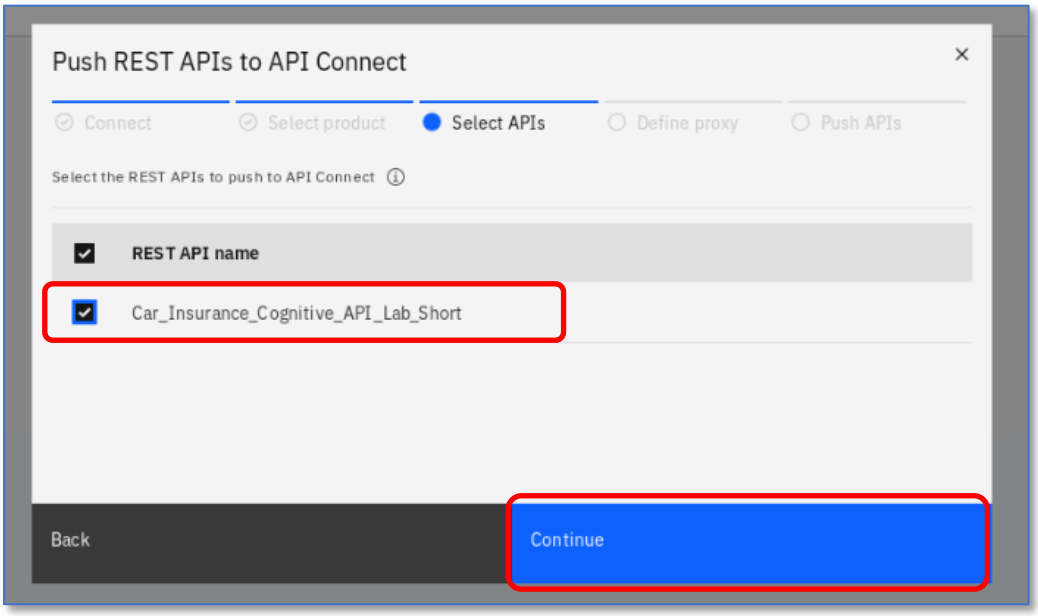
A product is a way of grouping together APIs. Consumers subscribe to products rather than individual APIs.

Enter a product name '`Car Repair APIs`' and a version '`1.0.0`' will do.

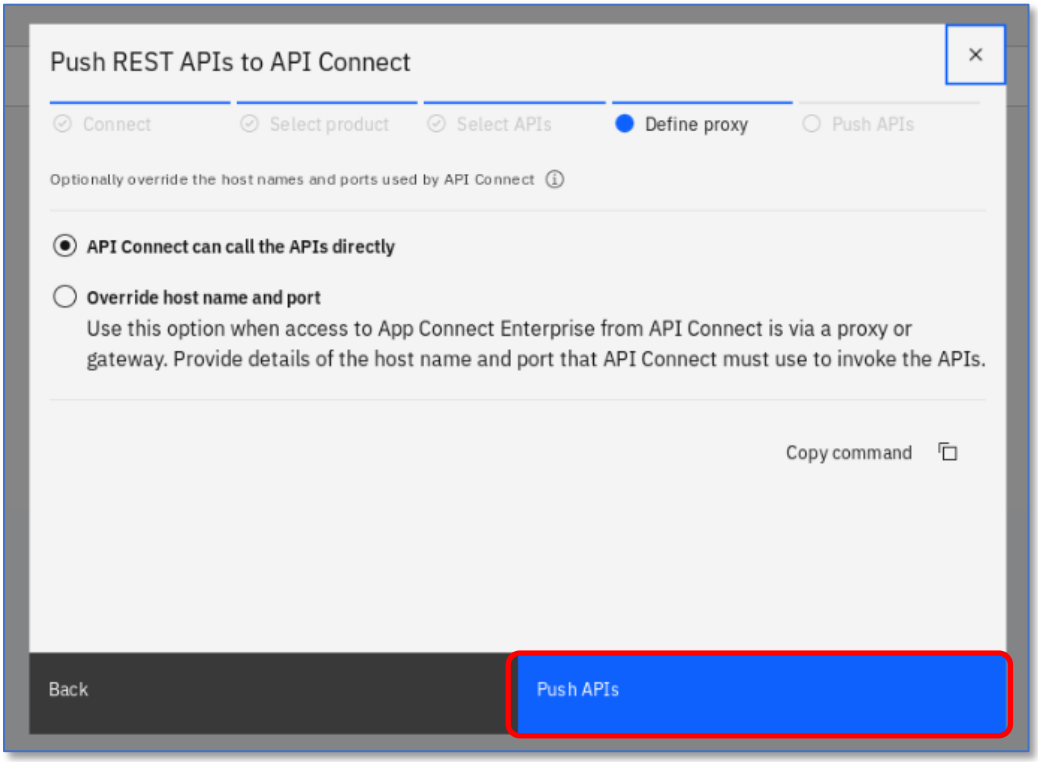
Select our '`Catalog for Demo use (main-demo-catalog)`' as our catalog and click 'Continue'



We only have one API – but we could select as many as we wish.



Select our API and click ‘Continue’

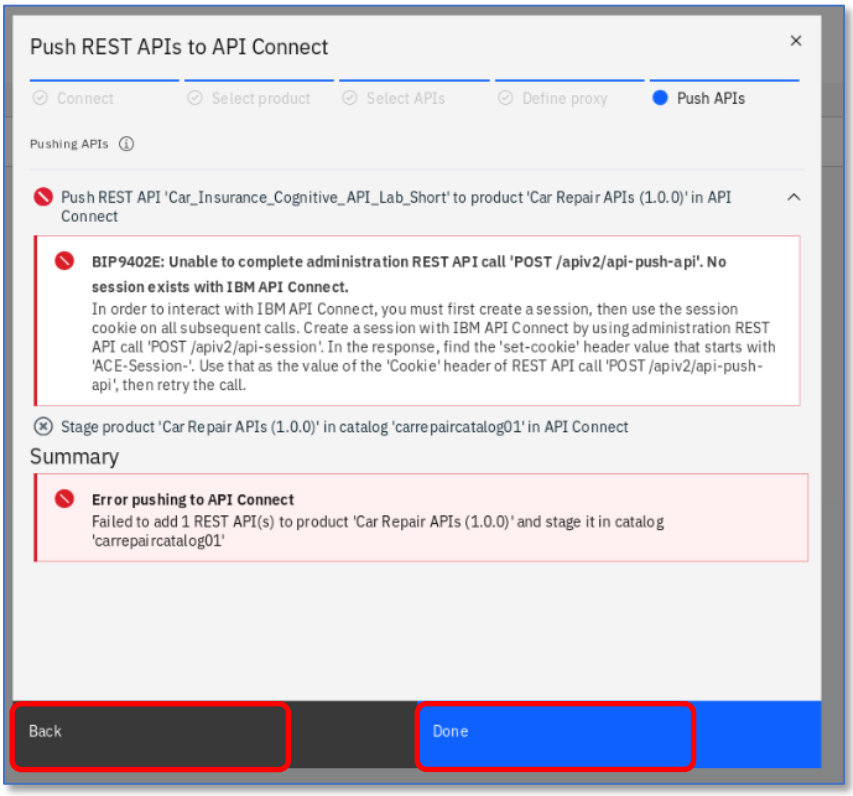


API connect can publish our APIs to use a proxy between API Connect and App Connect – this is used for remote gateways with proxies and load balancers in the way.

As we’re calling locally on the cloud pak, we’ll call our API directly.

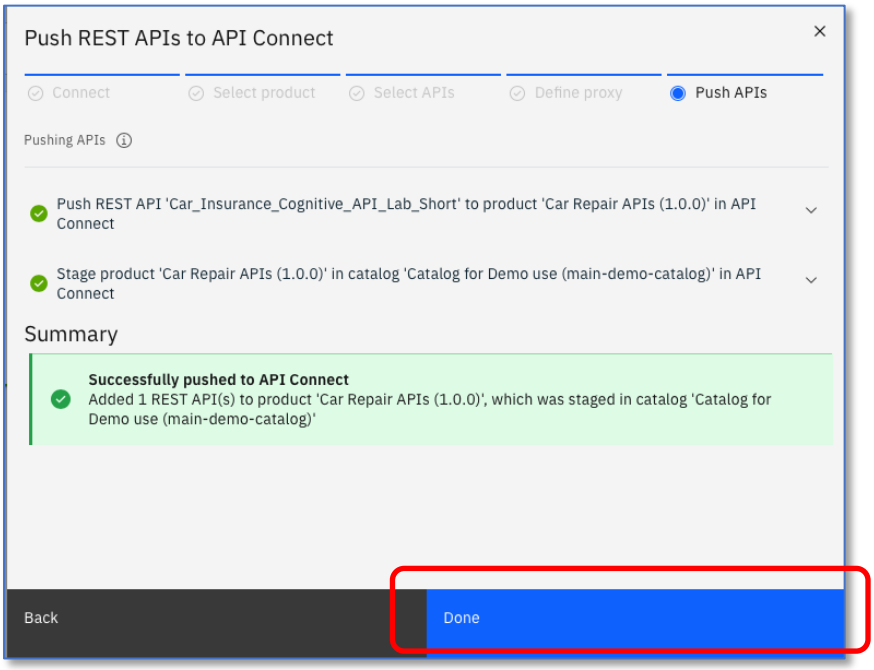
Click ‘Push APIs

Due to the lab environment, you may get this or similar:



If so, you may have to click ‘Back’ and ‘PushAPIs’ again or even several times. This is a lab environment issue….

When it’s successful, you should see:



Click done! OK, let’s go look for our API in API Connect.

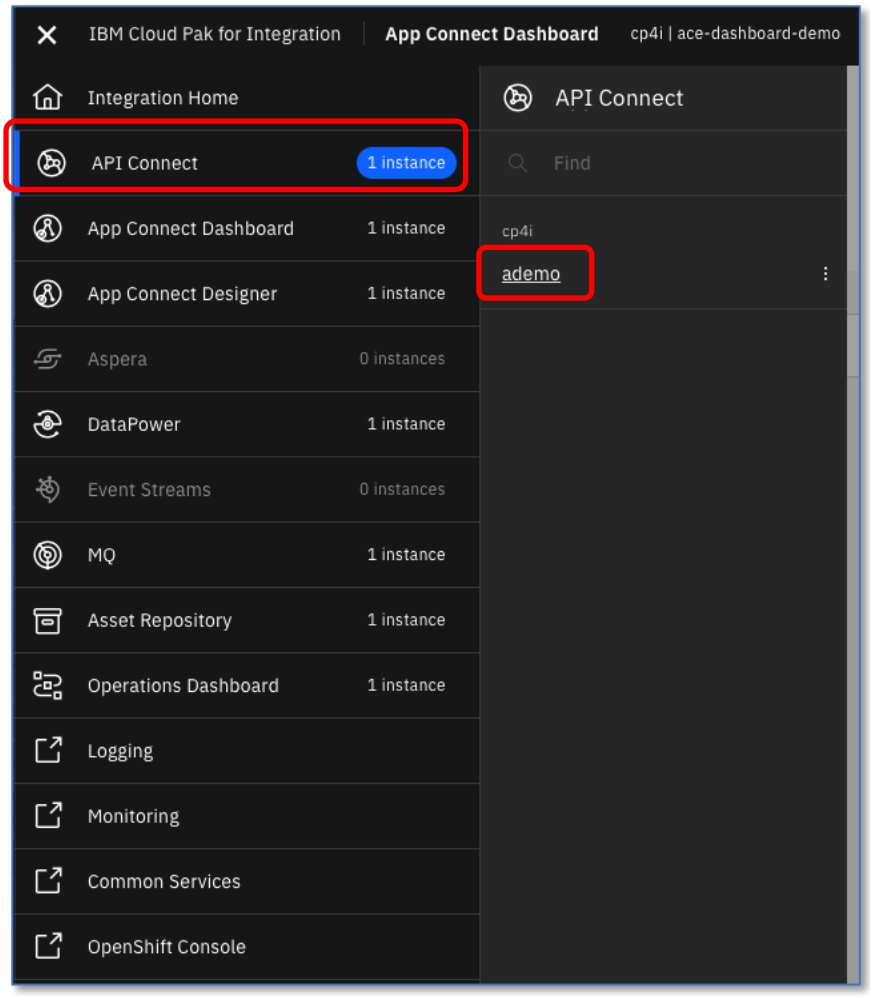
15 Managing our API using API Connect

Now our API is in API Connect, let's go there and do some API Management.

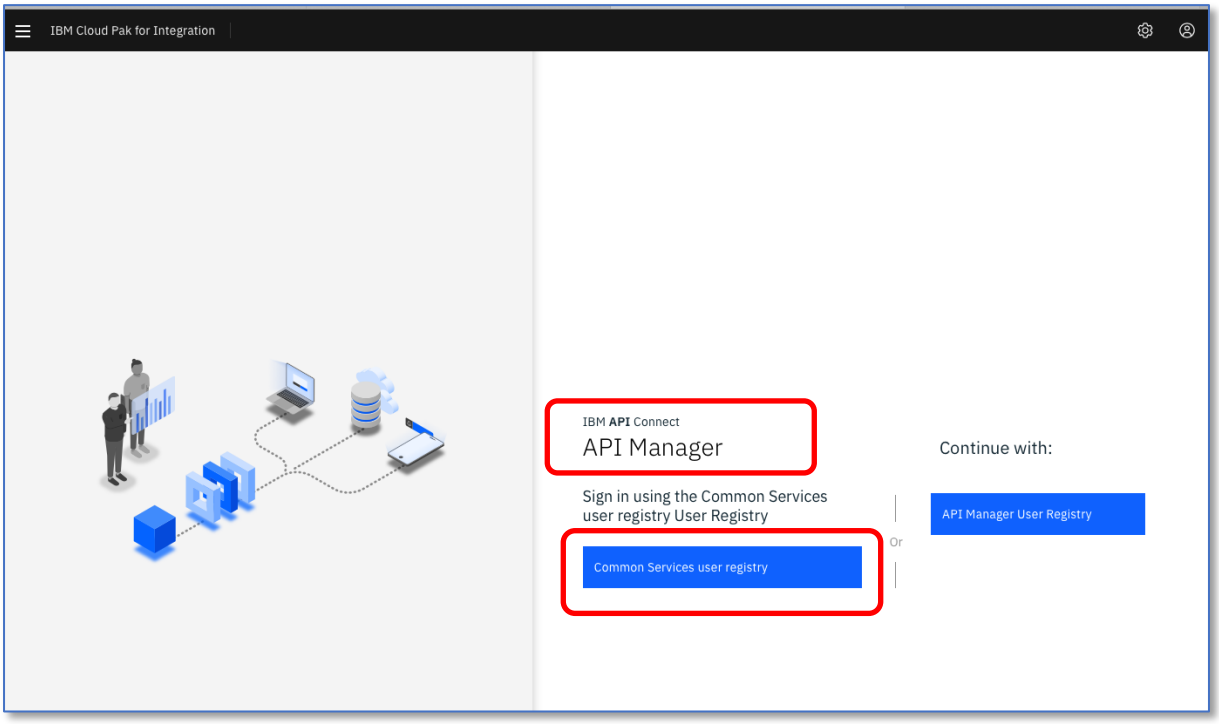
We want to be able to add security, define some rate-limiting plans and publish the API to a secure gateway.

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..



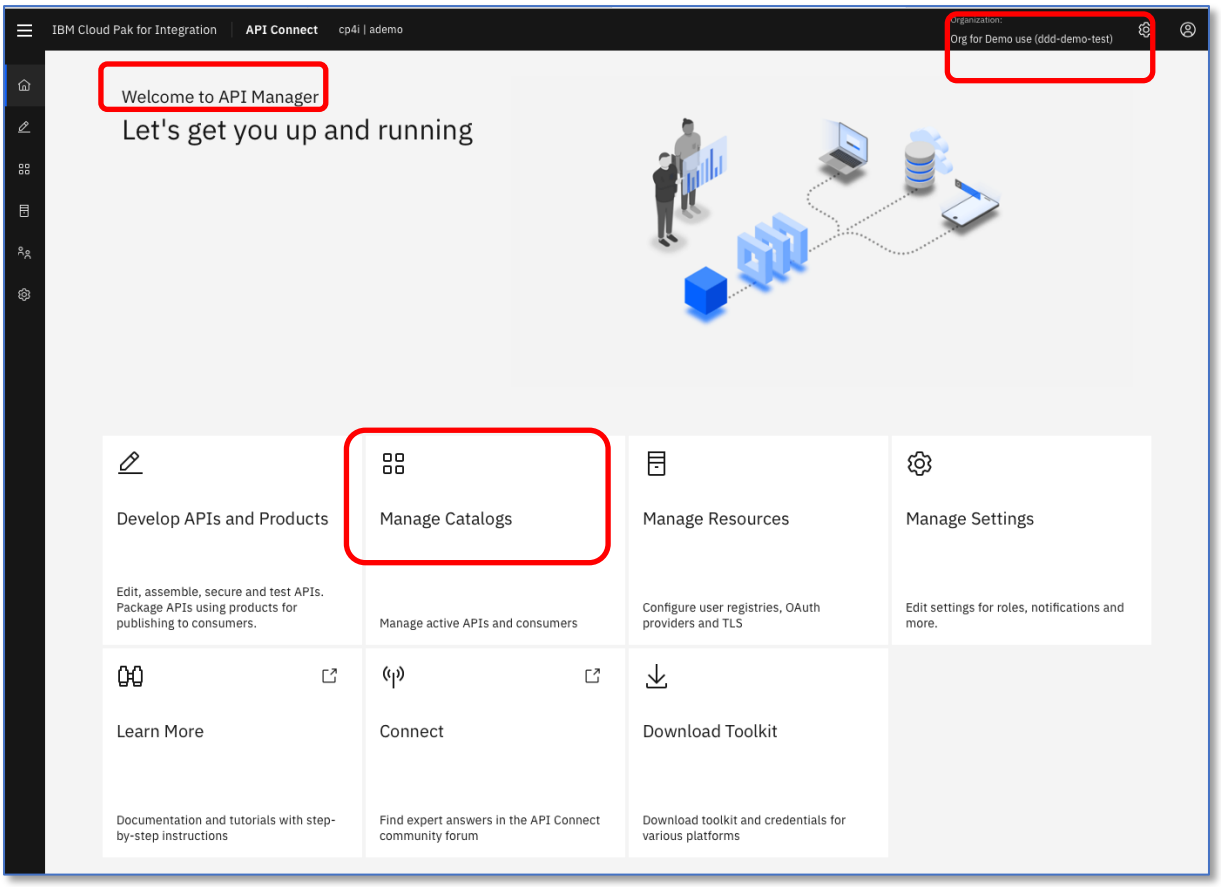
You'll be asked to log into the API Manager. Click on 'IBM Common Services user registry'. You should be logged in automatically using SSO. If not, use 'admin' and the same password you used to log into the cloud pak home page.



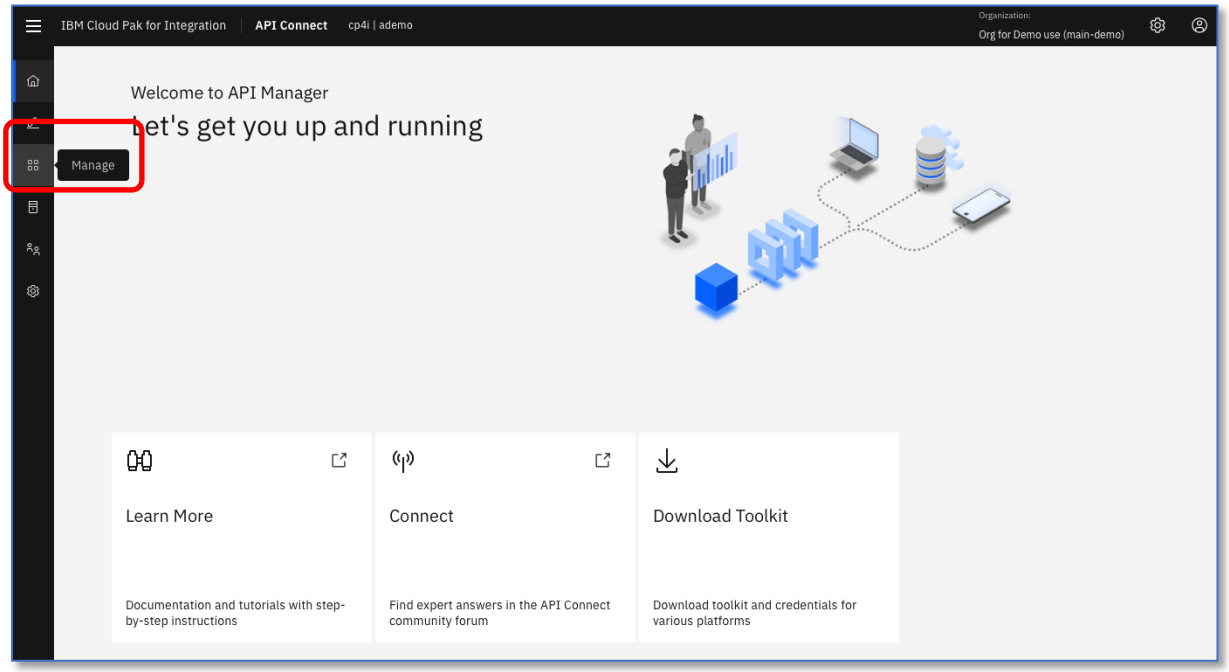
Make sure it says ‘Welcome to API Manager’.

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

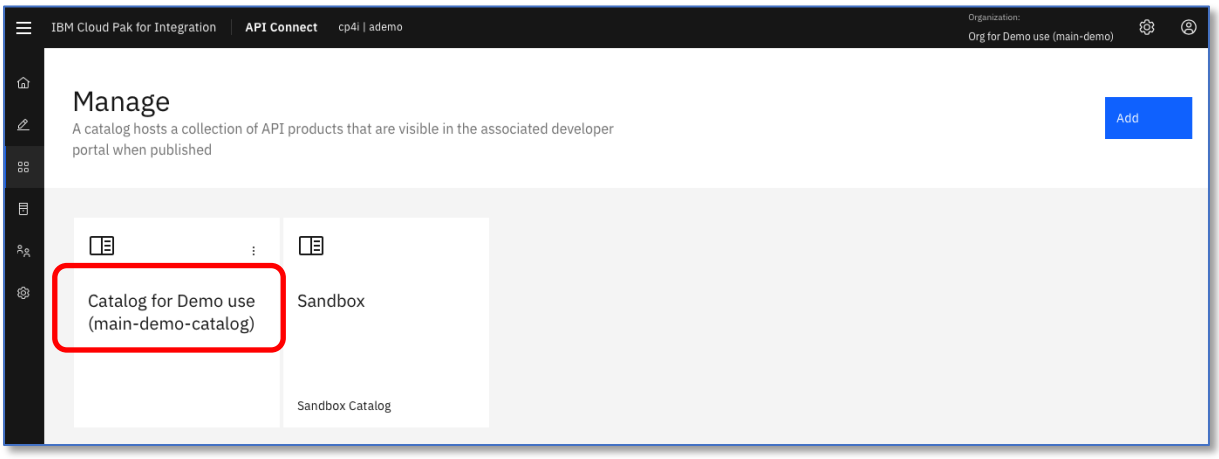
Click “Manage Catalogs”:



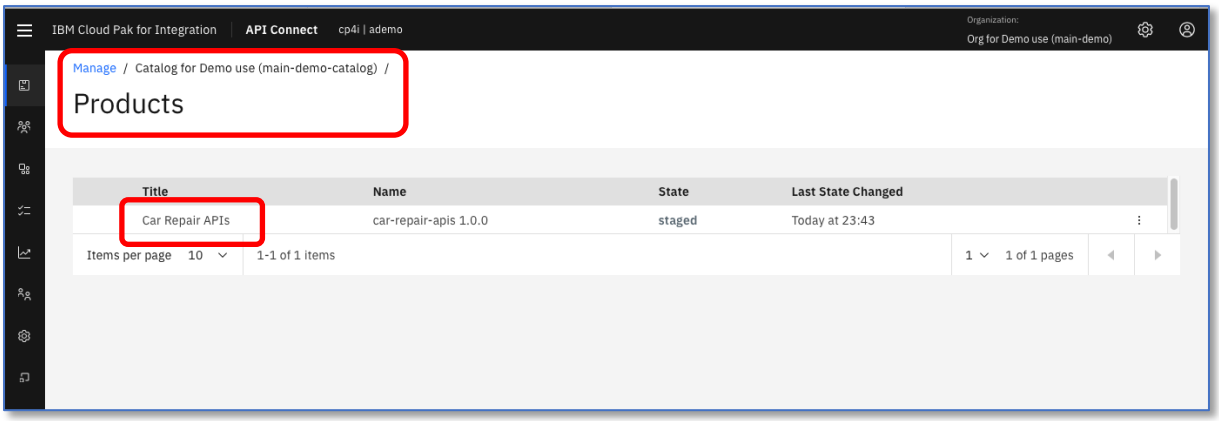
(If you get a screen like below and can't see 'manage catalogs' – click the 'manage' menu on the left)



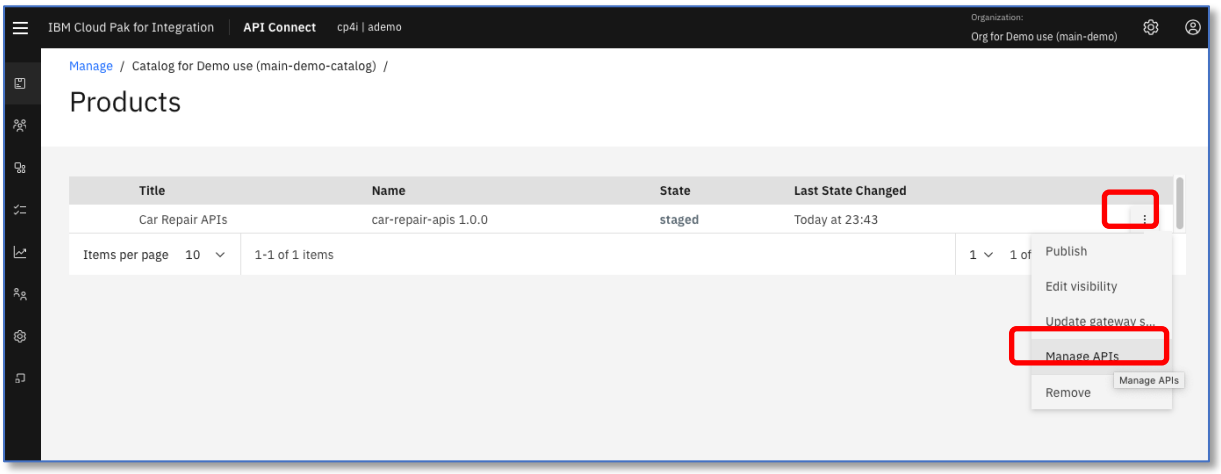
Click on the “`Catalog for Demo use`” catalog



You can then see the Product that we named in the ‘Push to APIC’ dialog box. Has been created and staged in the catalog for us.

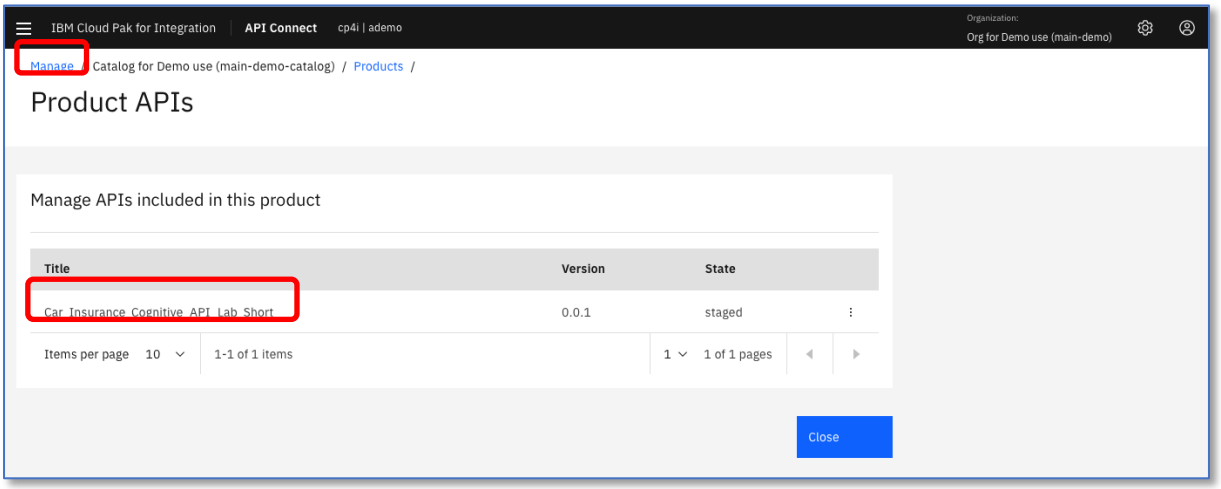


Click ‘Manage APIs’ in the three dots (...) ‘overflow’ menu on the right

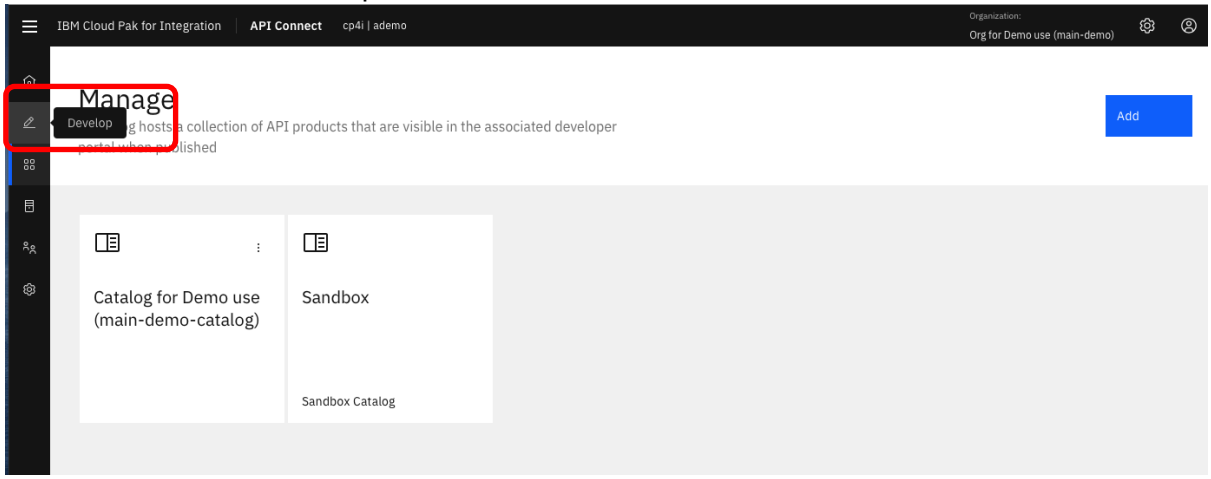


We can then see our API that we pushed already created for us inside the product: All of this is the integration between ACE and APIC.

Click in ‘Manage’ to return to the ‘Manage Catalogs’ screen.



Then click on the ‘develop’ menu

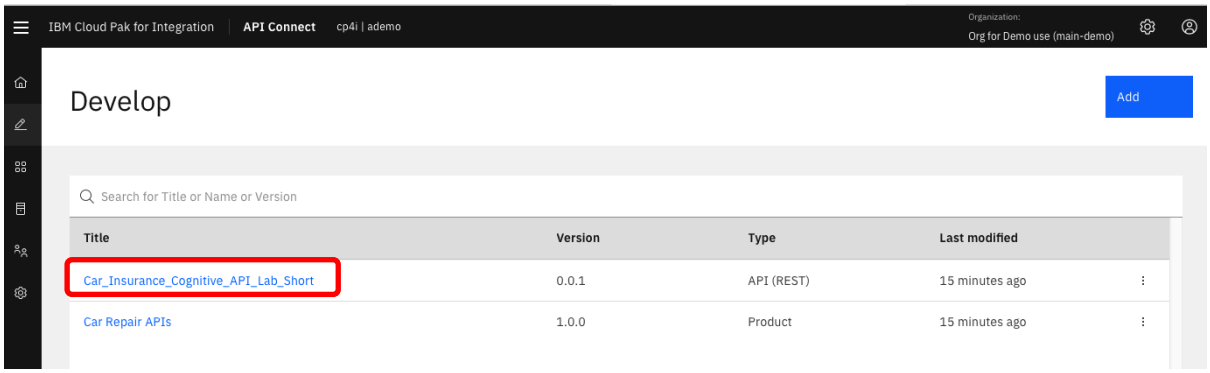


15.1 Adding a security Policy

Now we need to add security to our API. We can define various security policies, but we're going to keep it simple for this demo and use an API key.

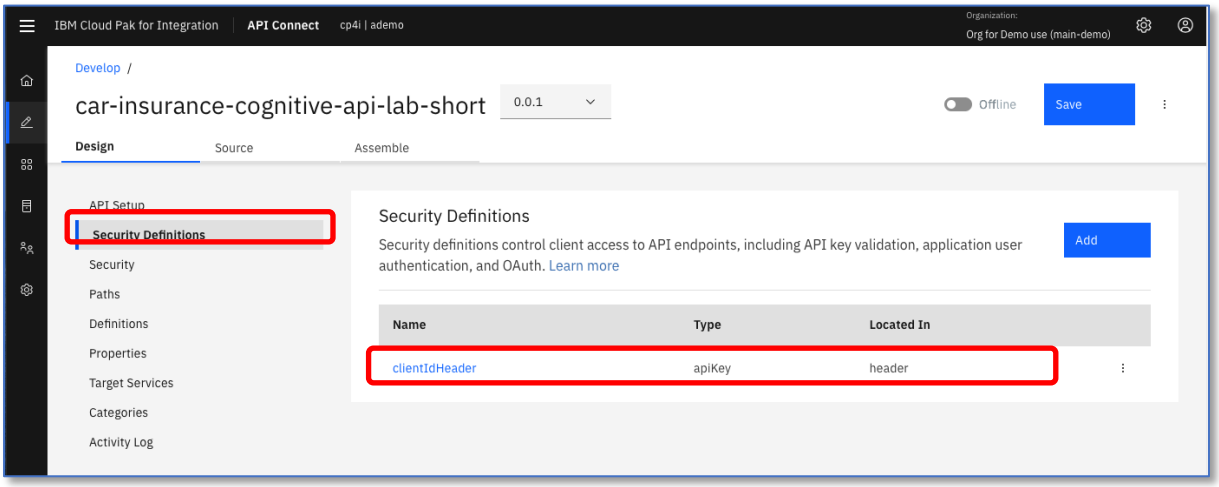
You'll see our API and our Product in the 'Develop' Screen.

Click on the `Car_Insurance_Cognitive_API_Lab_Short` API.



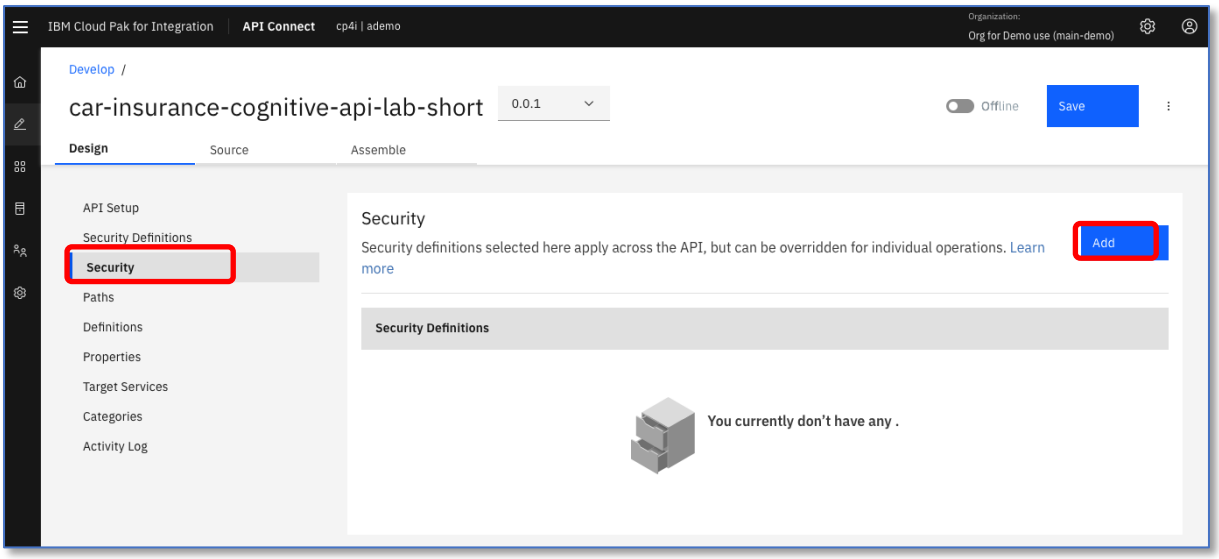
Here we can change a lot of things about the API in API Connect – we don't have time to cover them all here:

Click on 'Security Definitions'



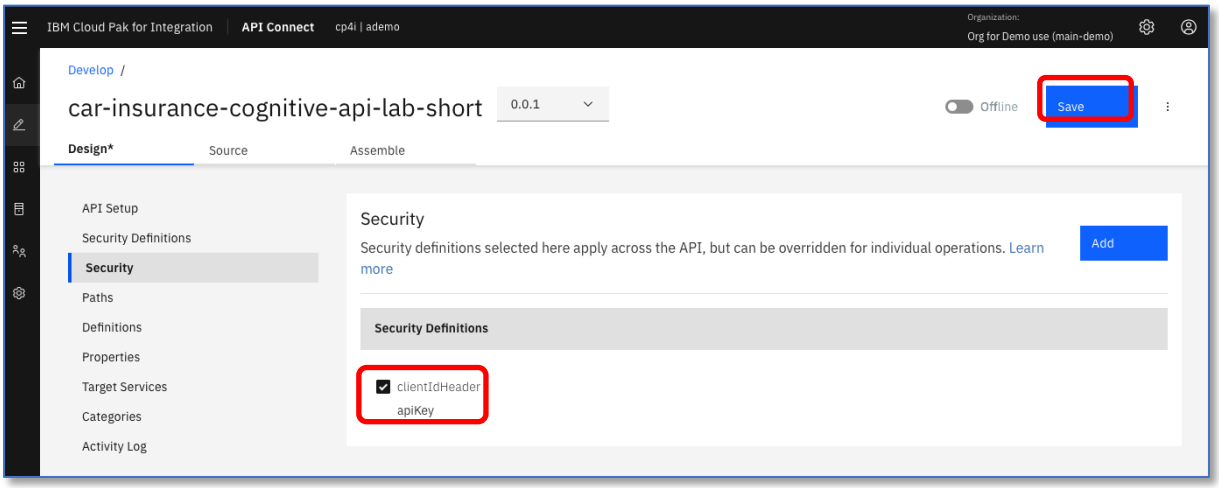
You can see that we have a pre-defined security definition of 'clientIdHeader' of type apiKey located in the Header.

Now click on 'Security'

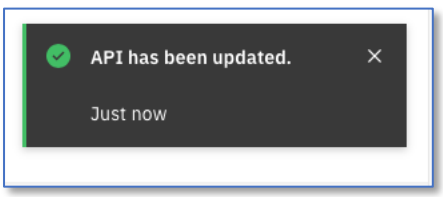


You can see that we have no current security definitions applied to our API. Let's add the clientIDHeader security definition.

Click on 'Add' (top right)



You'll see the clientIdHeader as a possible security definition. Click the check box to select it, then click 'Save'. You'll see the API update notification.

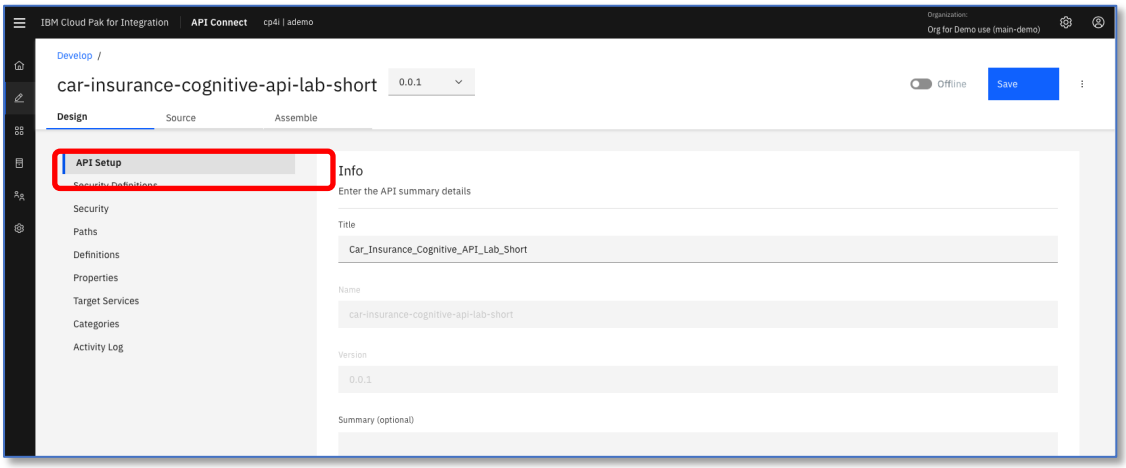


Remember that we needed API keys when connecting to Watson Services and Salesforce? This is exactly the same thing. They all protect their APIs with API keys.

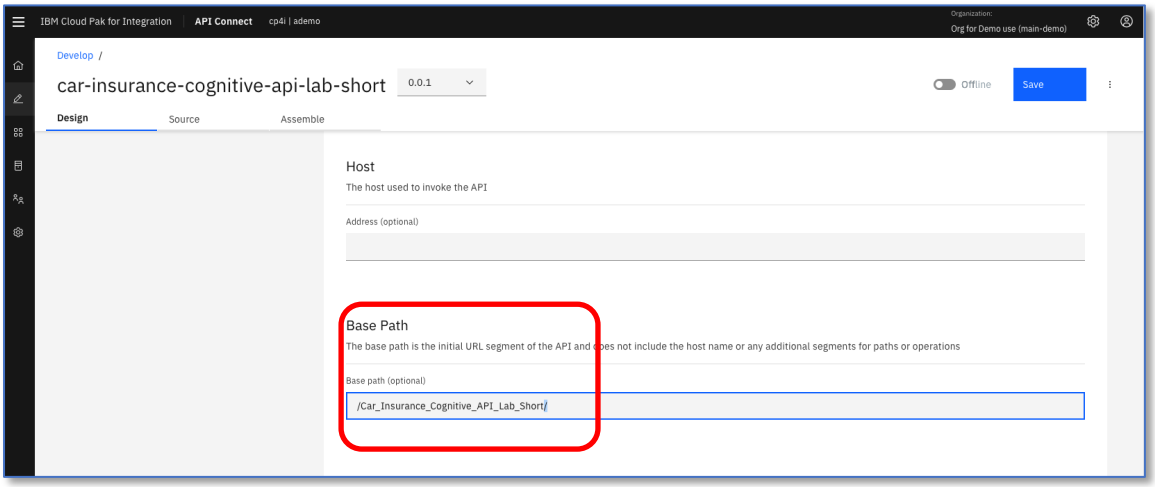
We've added a security policy to ensure only authorized consumers (those who have a valid API key) can consumer our API, we need a rate plan to make sure they can only call it as often as we allow them to.

IMPORTANT: WE NEED TO UPDATE THE API HERE! PLEASE READ.

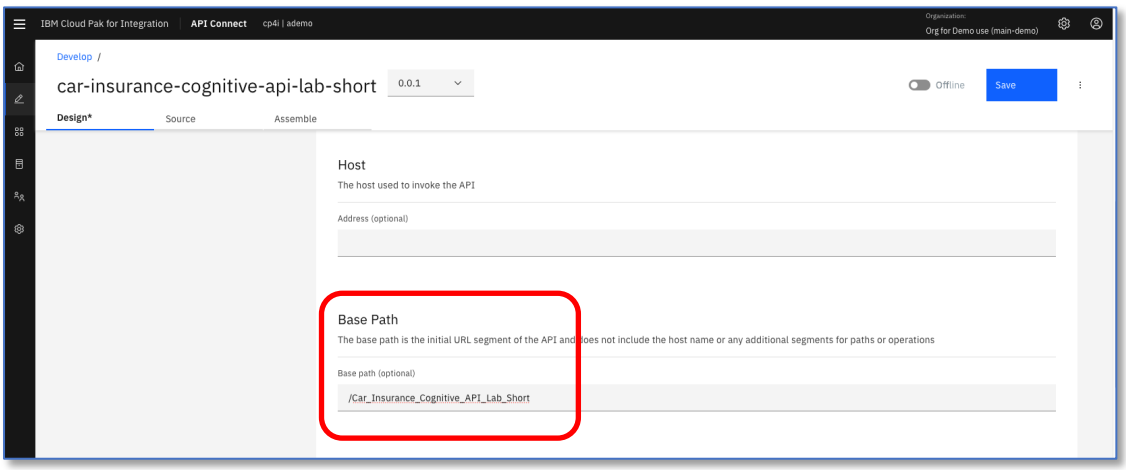
There is a configuration change we need to make to the API. Click on ‘API Setup’:



Now scroll down to ‘Base Path’. You’ll see the base path has a trailing slash (at the end) – remove this: So this

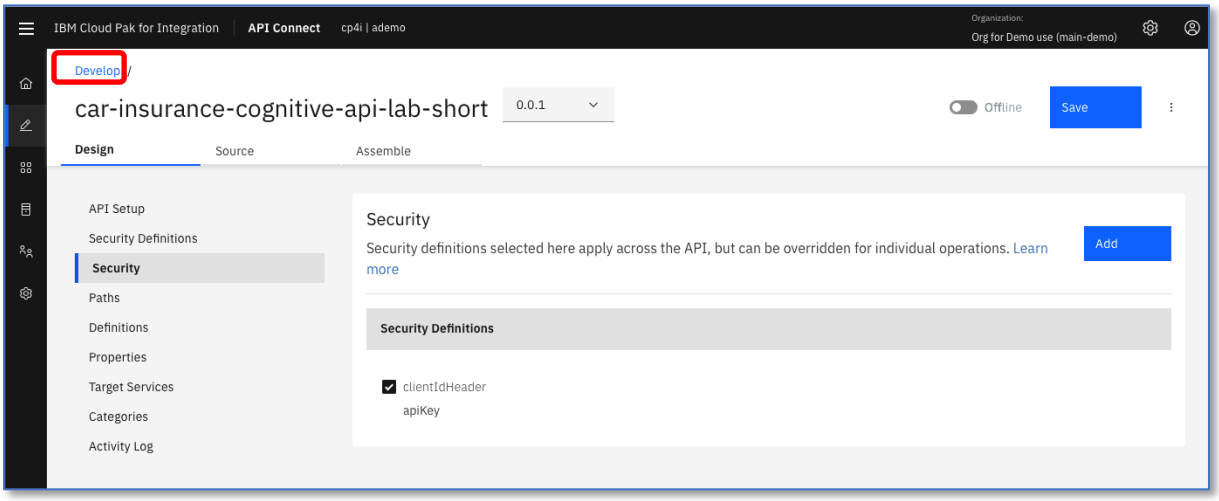


Becomes this like below: Then click ‘Save’



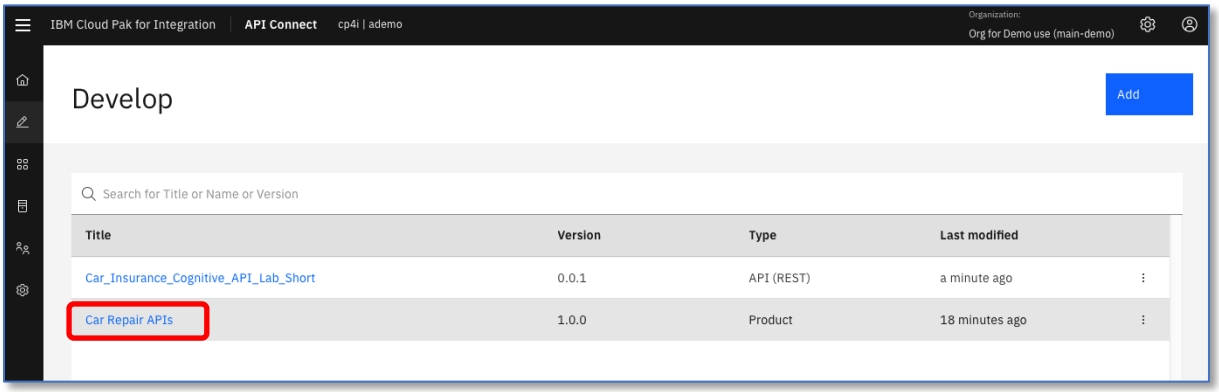
15.2 Adding a rate limiting Plan

Go back to the ‘Develop APIs and Products’
(click on ‘Develop’)



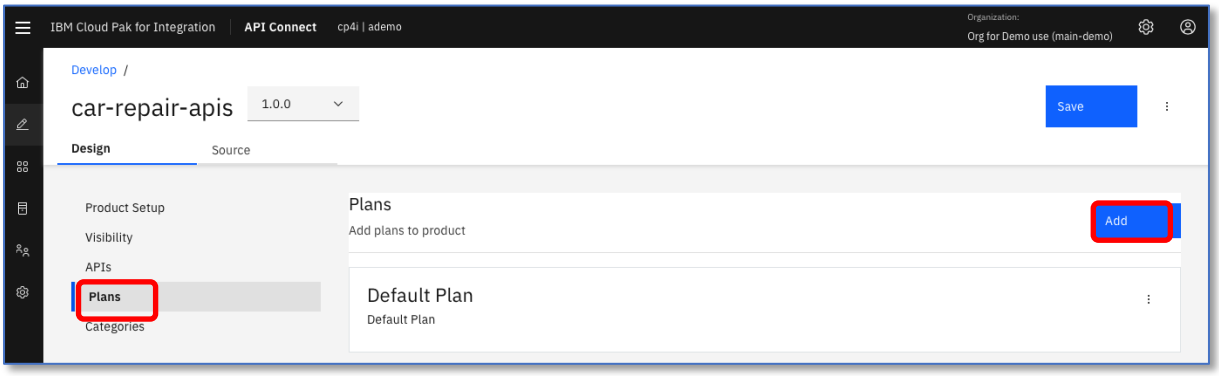
Security is applied to APIs. Rate limiting is applied to products.

Click on the ‘Car Repair APIs’ product



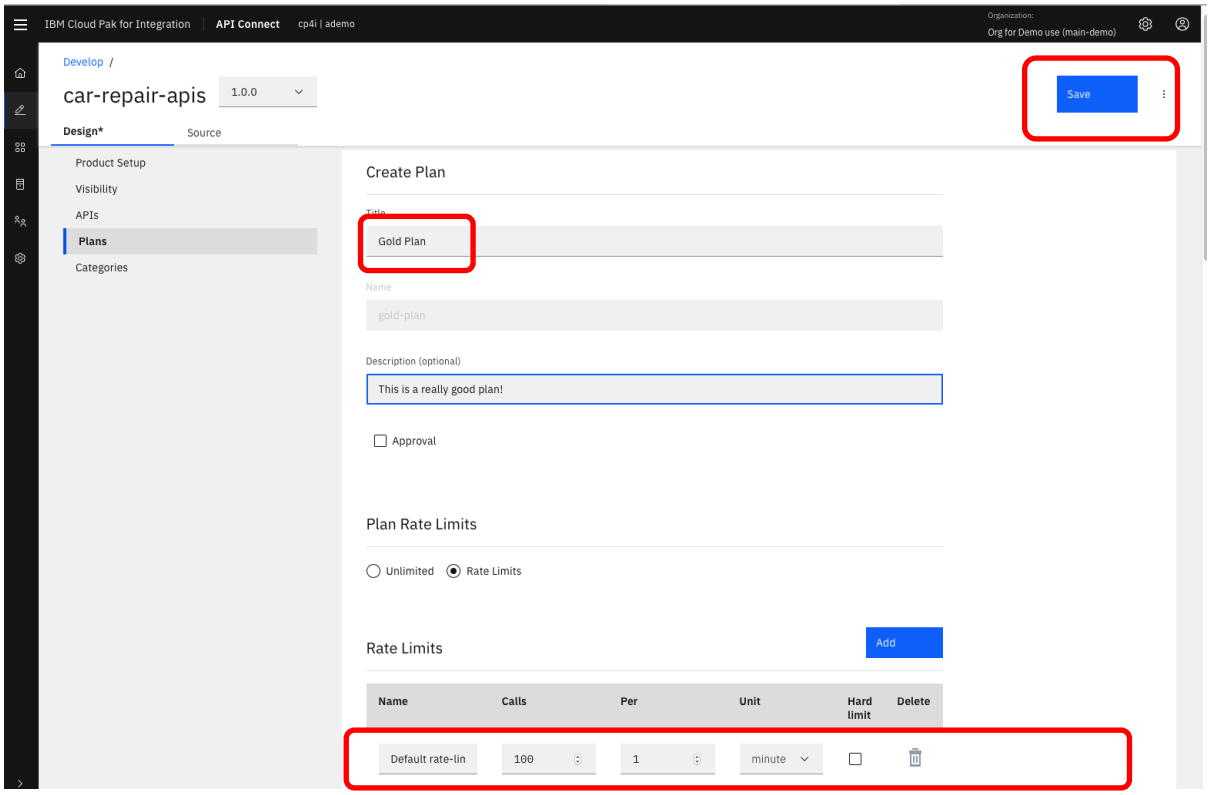
Rate Limiting is done using plans. Click on ‘Plans’

You’ll see that there is a default plan already created. We want to add another custom one. Click ‘Add’



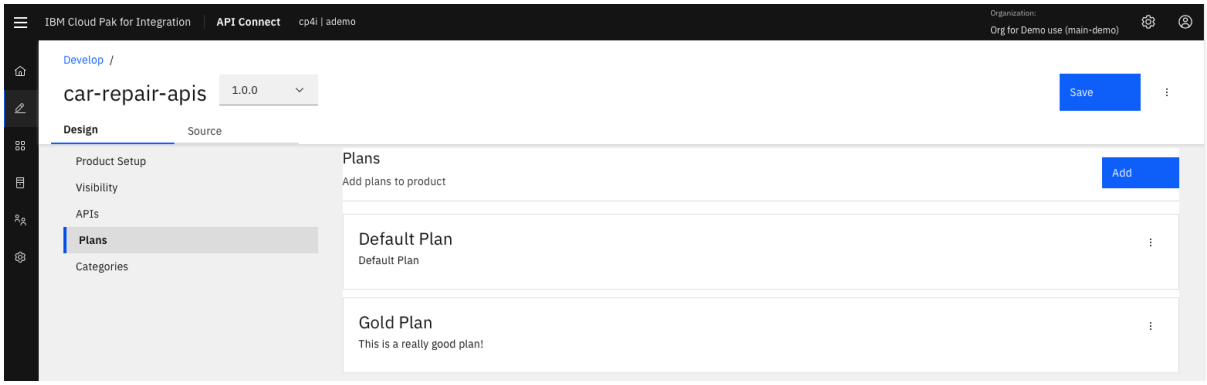
Now give our new plan a name (e.g. “Gold Plan”) and give it a rate limit – for example, here it is 1 call per minute: If you want to add more limits/bursts/plans etc, go ahead.

Then click ‘Save’



You should now be able to see both plan our new plan (Gold plan) and the default plan.

You can have multiple plans for different consumers – you can add approval steps for consumers when they sign up – or you can allocate them plans as a provider.



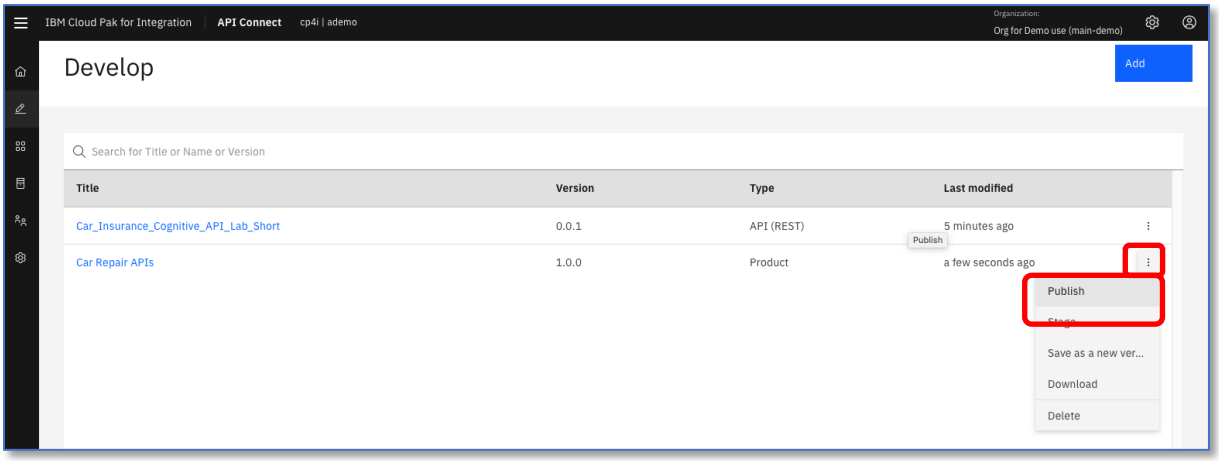
Now we have an API and a product, we want to publish it so that we can discover and call it!

Important: APIs/Products Can exist in multiple catalogs. Here we are developing our APIs and products.

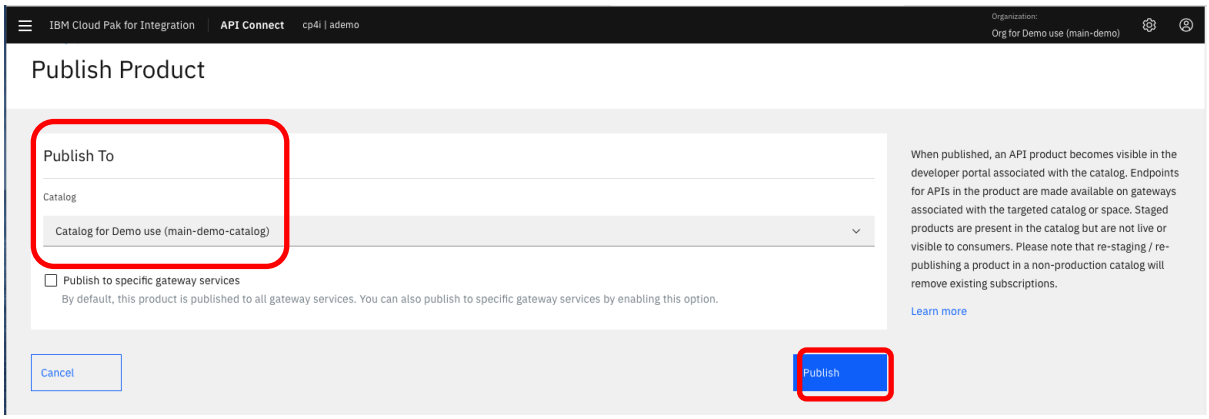
Our product and API were already staged in our “Catalog for Demo use” – we could have just published them but we chose to add security and plans. We must now republish to update them.

Go back to Home/Develop (Click ‘Develop’ on the top left)

Now click on the three dot overflow menu by the ‘Car Repair APIs’ product and click ‘publish’

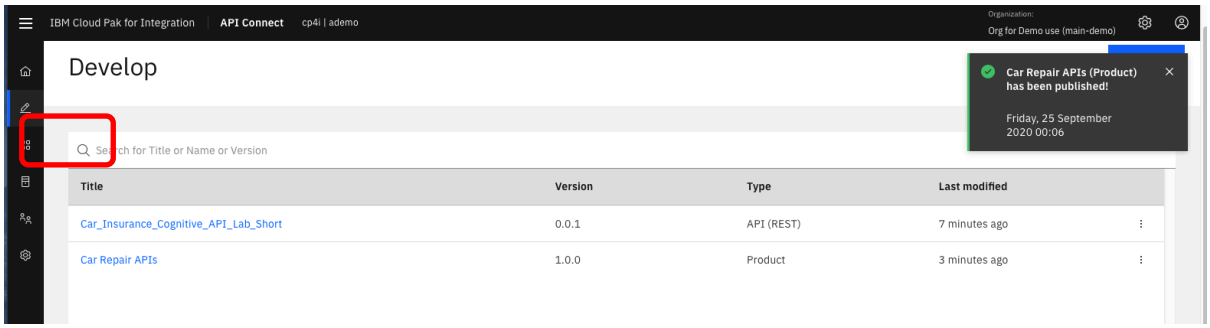


You’ll be prompted for a catalog to publish to – select ‘Catalog for Demo use (main-demo-catalog)’

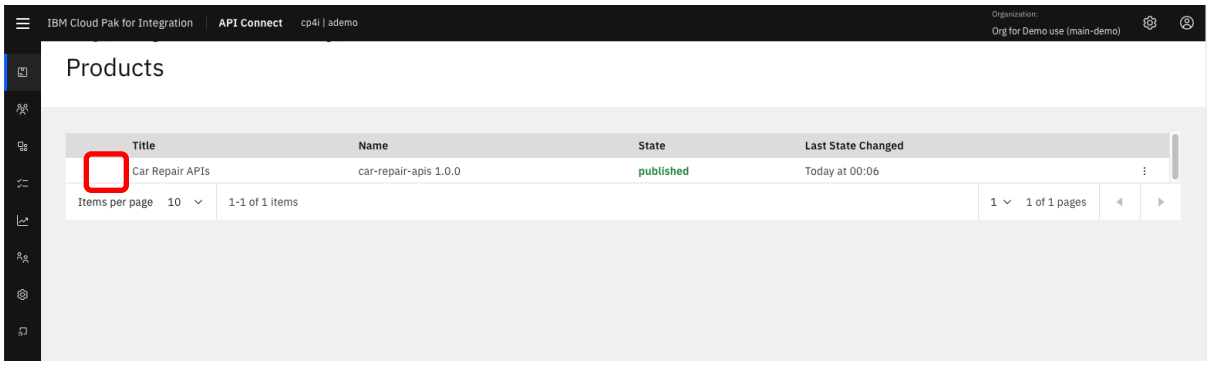


We only have one gateway installed so we can leave the checkbox blank – click 'Publish'

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)

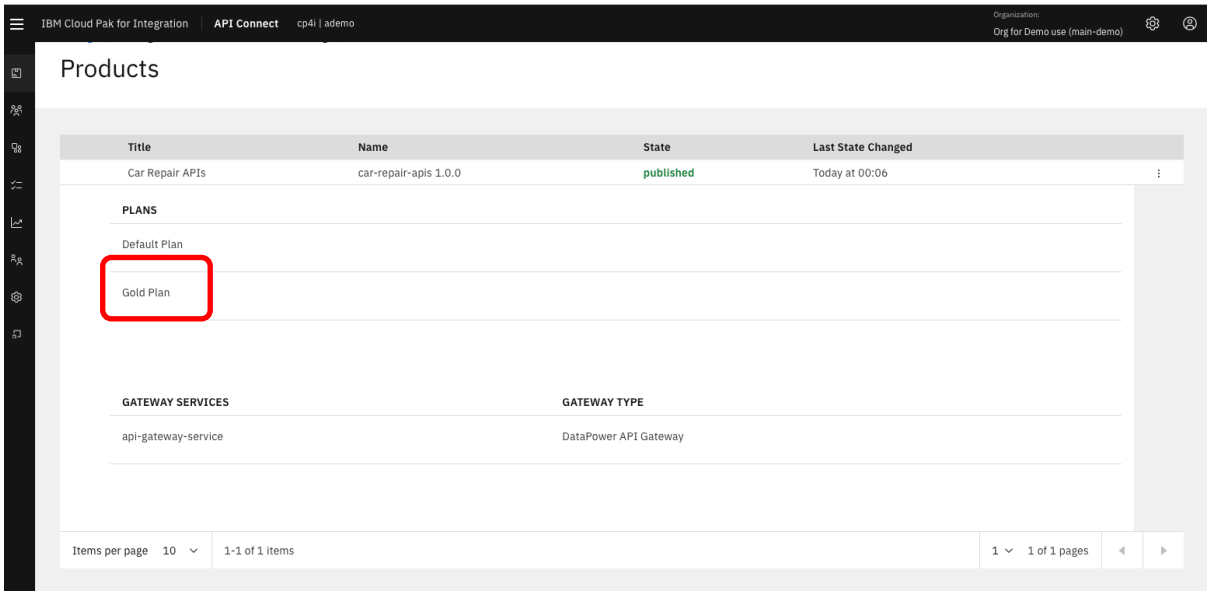


Product Status is 'published' – click on the twisty next to the 'title'



You can see our plans added into the products. 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.

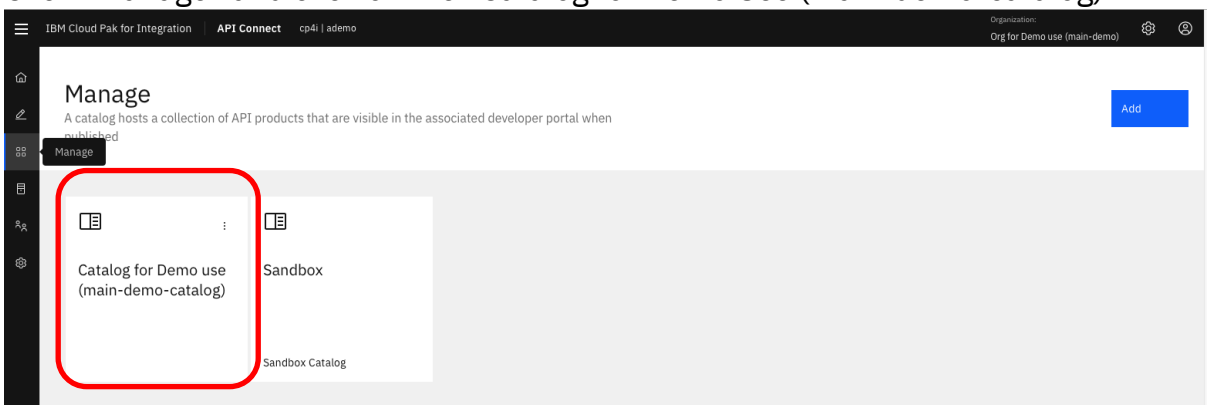

IBM Cloud Pak for Integration
API Connect
cp4i | ademo
Organization:
Org for Demo use (main-demo)
Products
Title
Name
State
Last State Changed
Car Repair APIs
car-repair-apis 1.0.0
published
Today at 00:06
PLANS
Default Plan
Gold Plan
GATEWAY SERVICES
GATEWAY TYPE
api-gateway-service
DataPower API Gateway
Items per page 10
1-1 of 1 items
1 of 1 pages

16 Creating a Portal

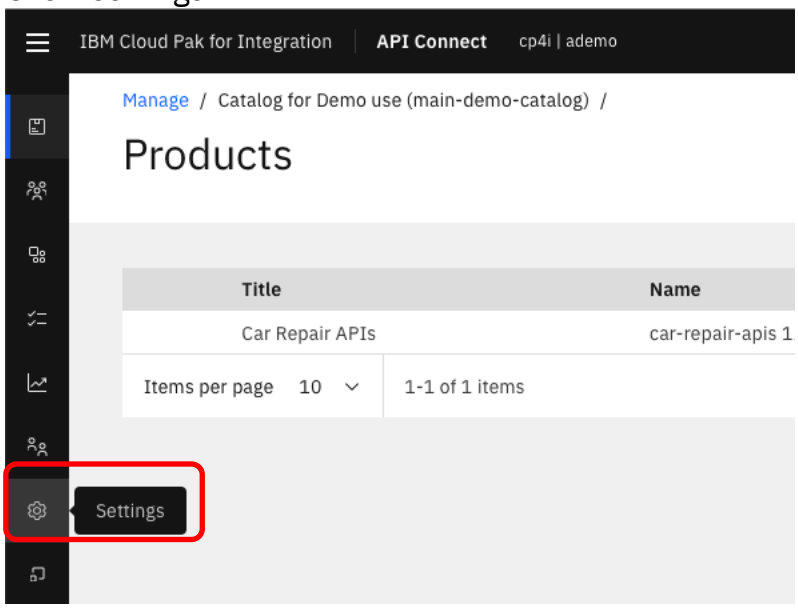
Now that we've published our API, we need to make sure that our API consumers can discover it and use it.

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.

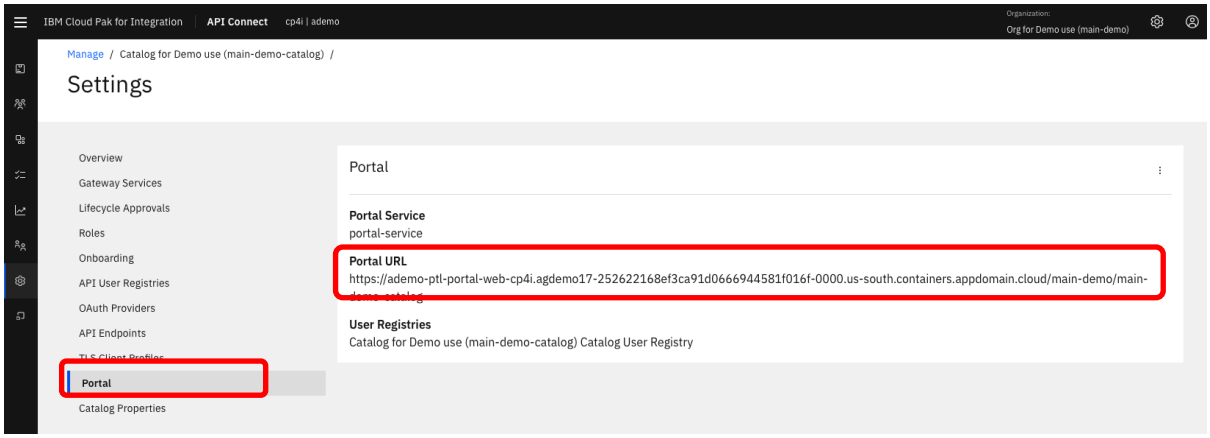
Click "Manage" and click on the "Catalog for Demo Use (main-demo-catalog)"



Click ‘settings’



Now click “Portal”



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

Take a note of the Portal URL – we’ll be needing this to sign in later.

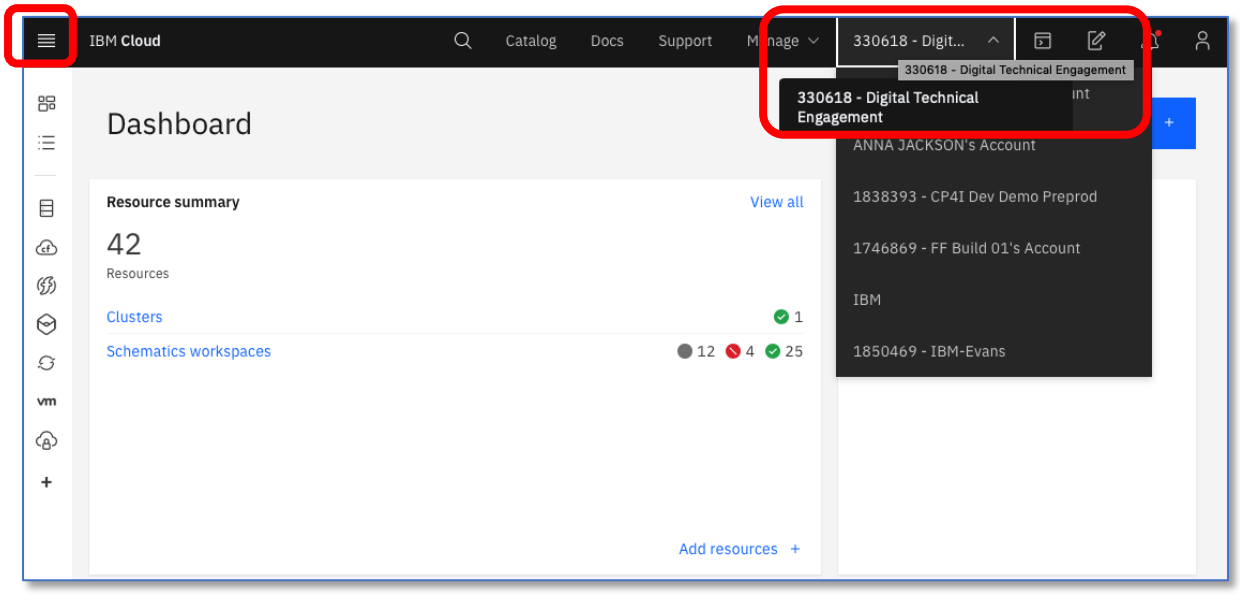
Normally, if you create a portal site yourself as part of configuring API Connect, API Connect will send an email to you with a 1-time password reset link so that you can set up the ‘`admin`’ account for the portal.

As we’ve created the portal site for you as part of the 1-click demo install, you won’t have that eMail (actually, you might – check your mailtrap mailbox TODO).

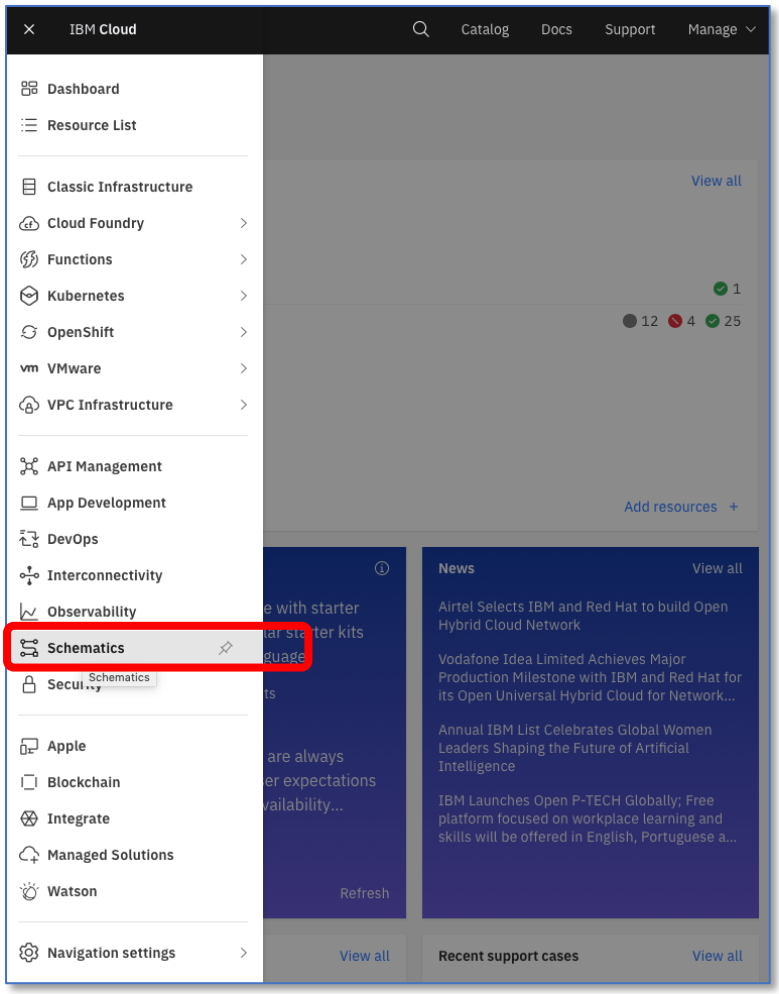
We have put the reset link in the installation logs…to find them to do the following:

Go to https://cloud.ibm.com (best to do this in a separate tab) and sign in with your IBM id:

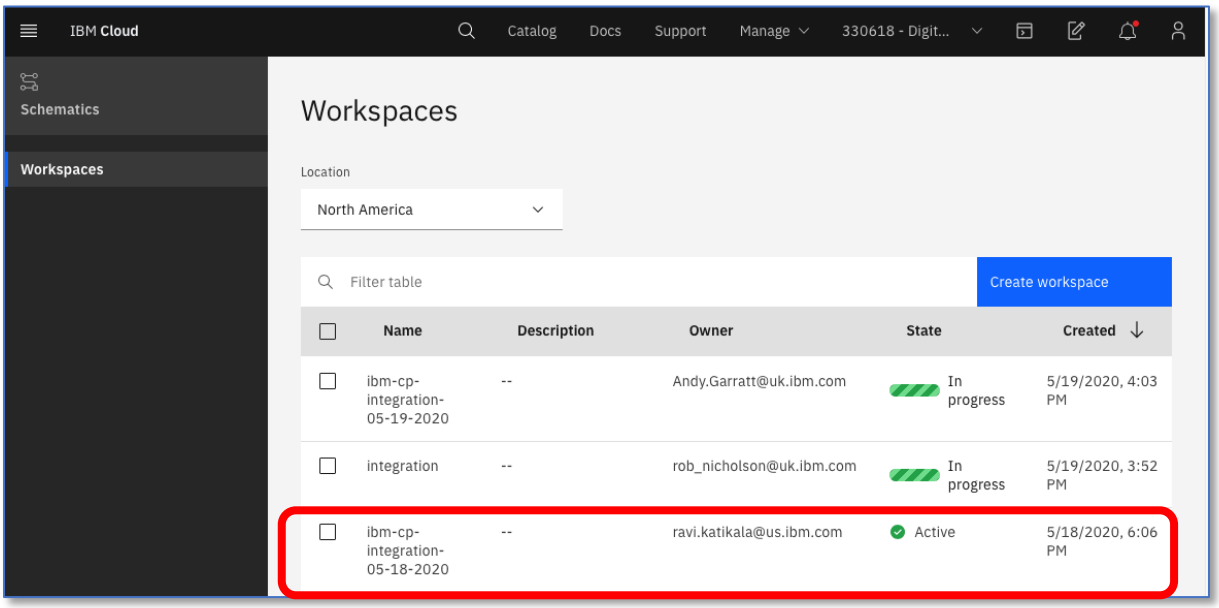
Go to the ‘Account’ pull down and make sure you’re in the ‘DTE’ account: If you’re not pull down and click on the DTE account to change it.



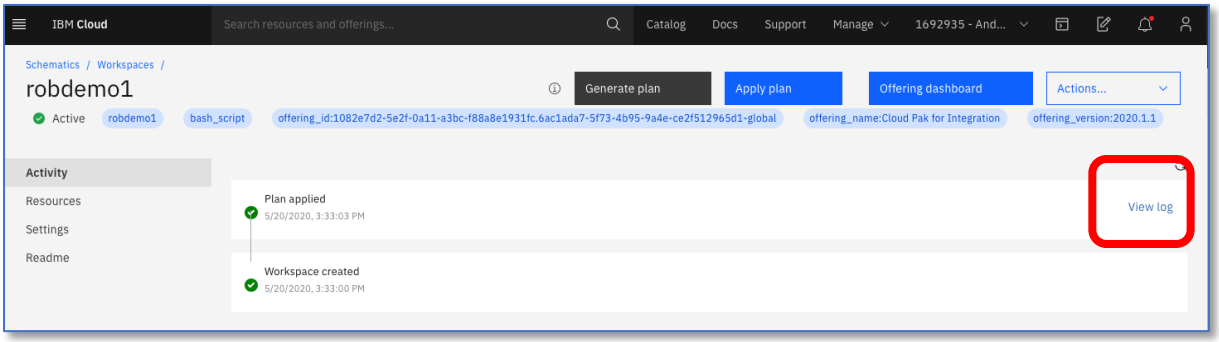
Next, click on the ‘Hamburger’ menu and click ‘Schematics’



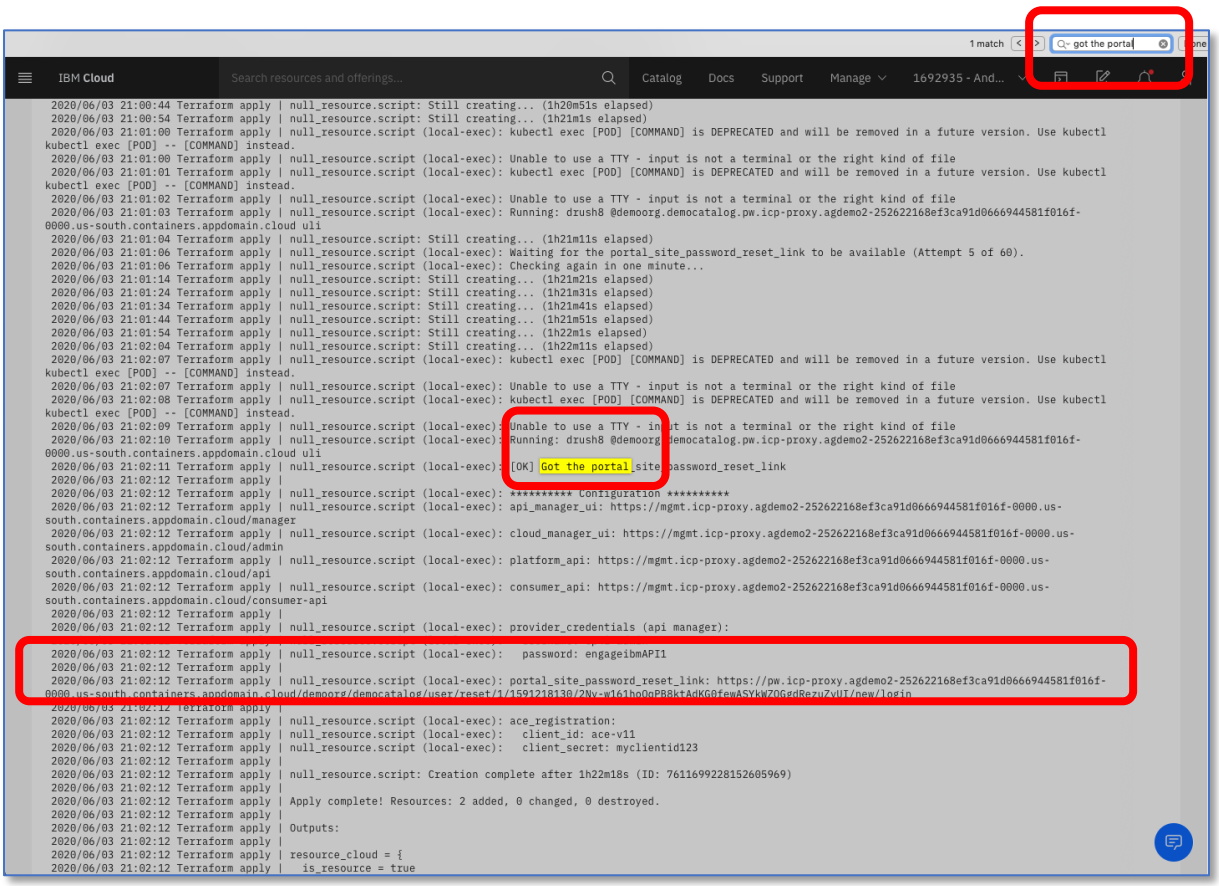
Find your Workspace (This is a shared cloud account, even though you will get your own ROKS cluster – look for your email to see which one is yours). Click on it (your state should be ‘Active’



Now click on ‘view log’

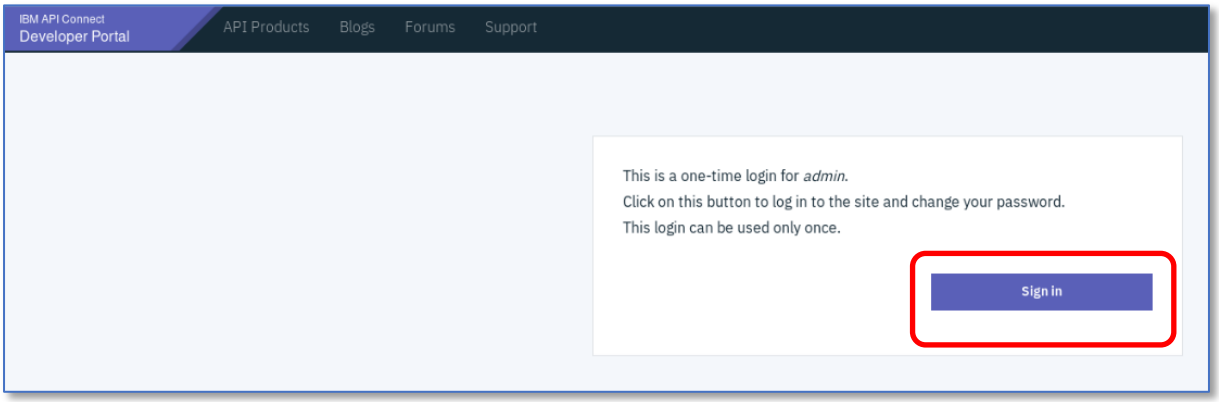


The link is right near the bottom of the log – search for ‘Got the portal’ in your browser and you’ll find it:

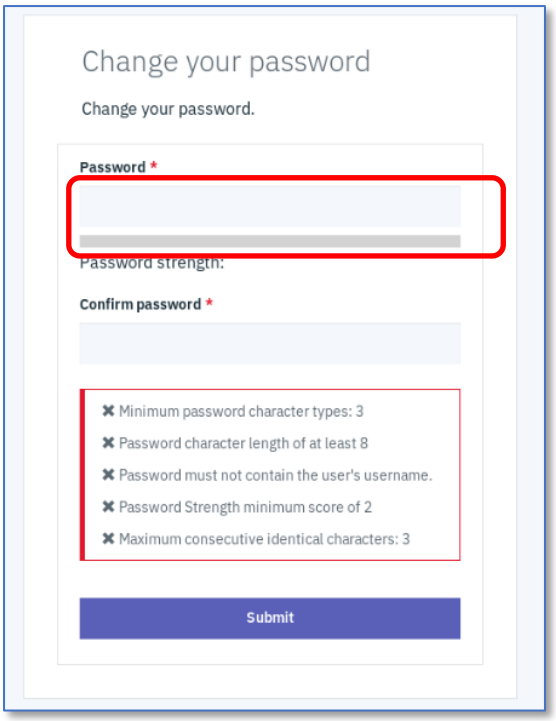


Copy/Paste the one-time link into a new browser tab.

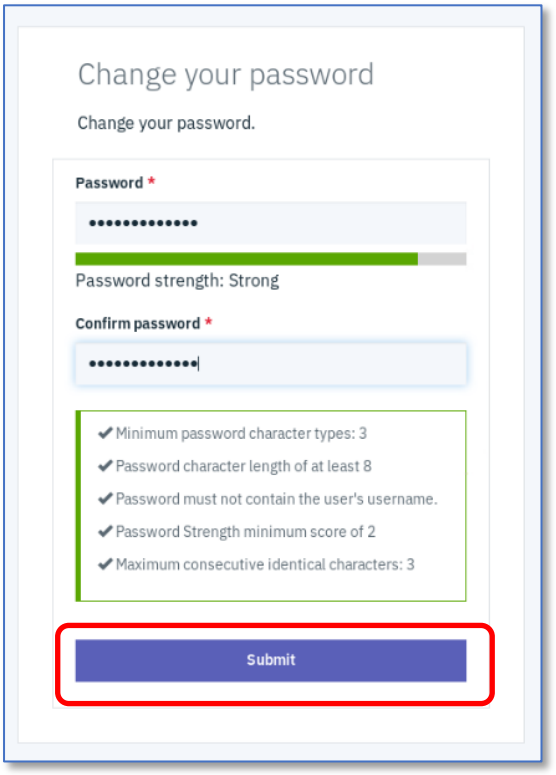
You’ll see the following: This will create an admin user in the portal registry



Click ‘sign in’



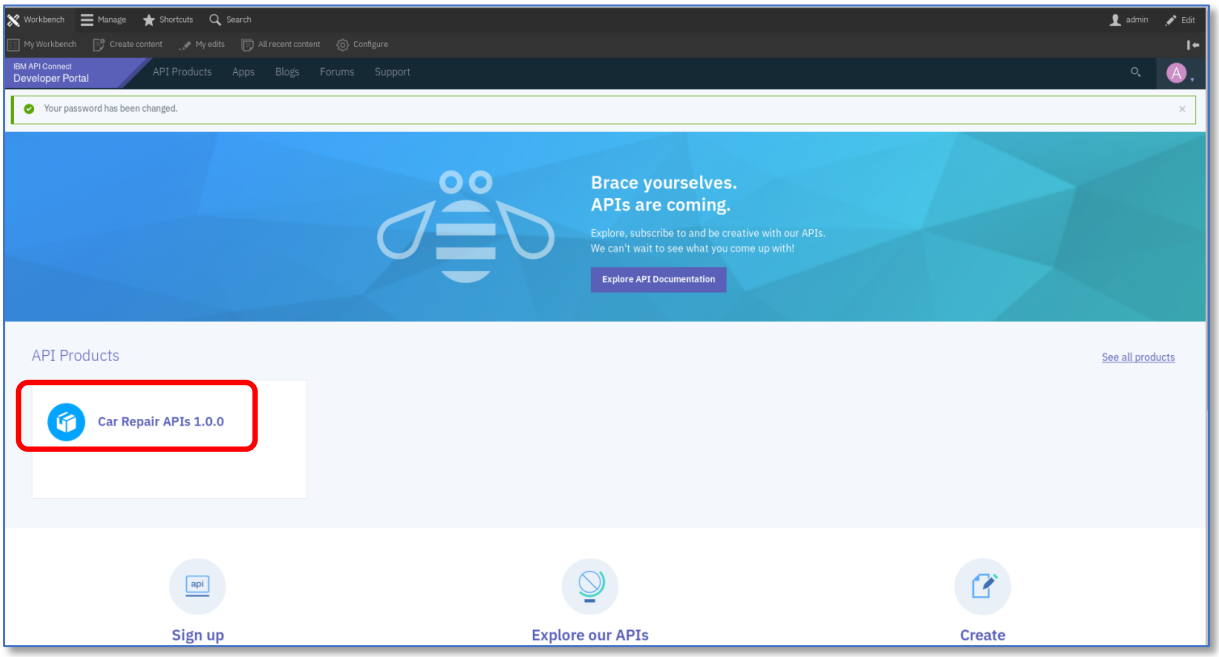
To save you thinking of a password to match the rules, we suggest using "`engageibmAPI1`"



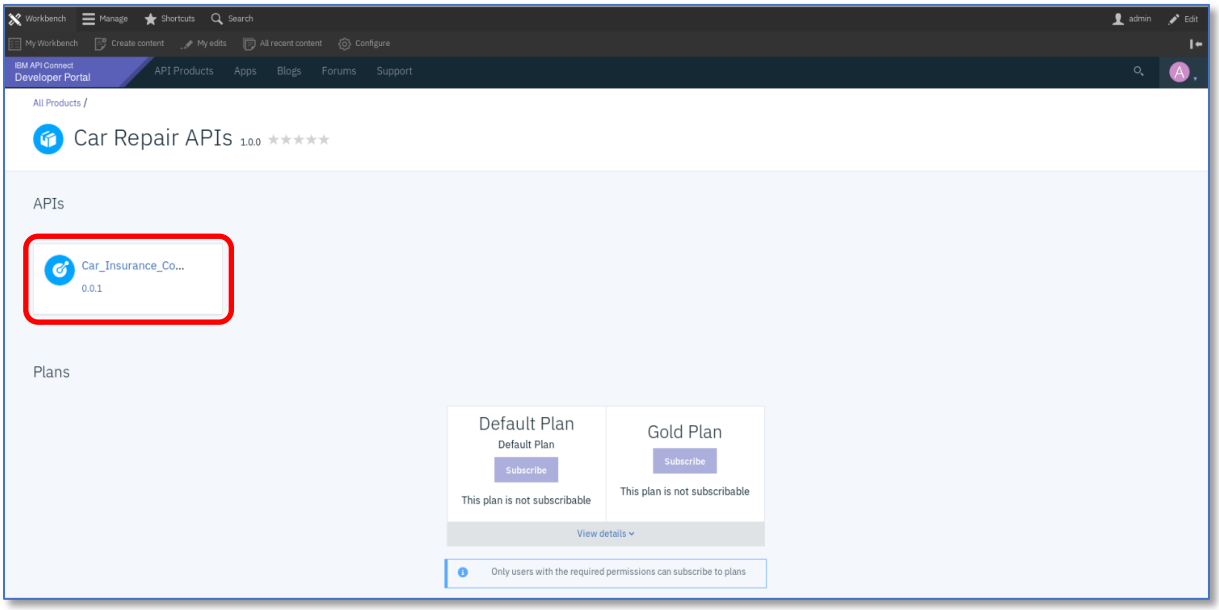
Change your password and click 'Submit'

You should see the portal – with our API already there!

As you're the owner you can also see all of the portal tooling – you can customise the portal look-and-feel if you wish – although we don't have time for that now..



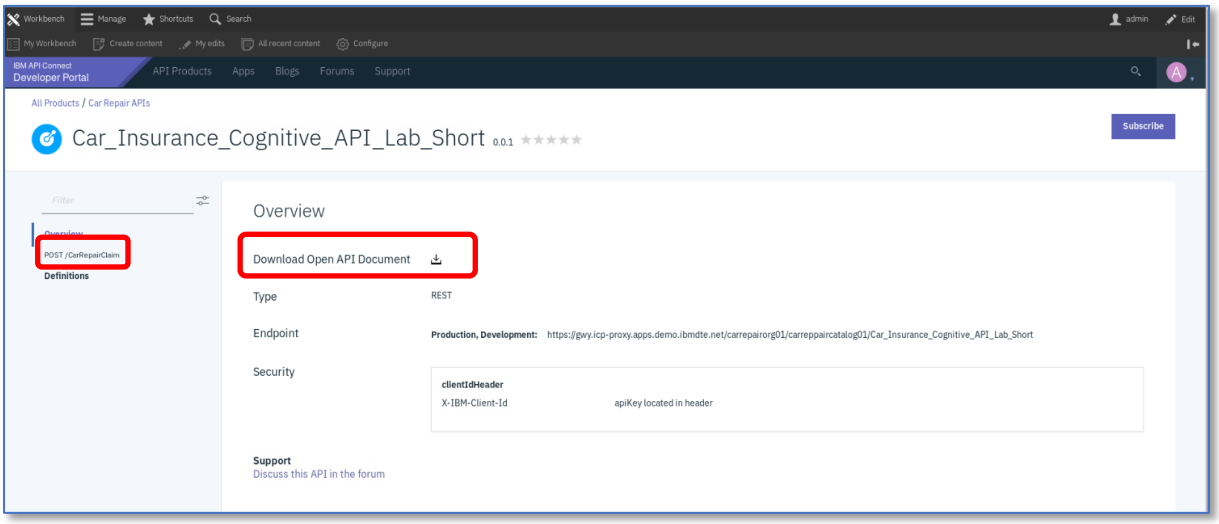
Let's see our API as others will see it: Click on the 'Car Repair APIs 1.0.0' product



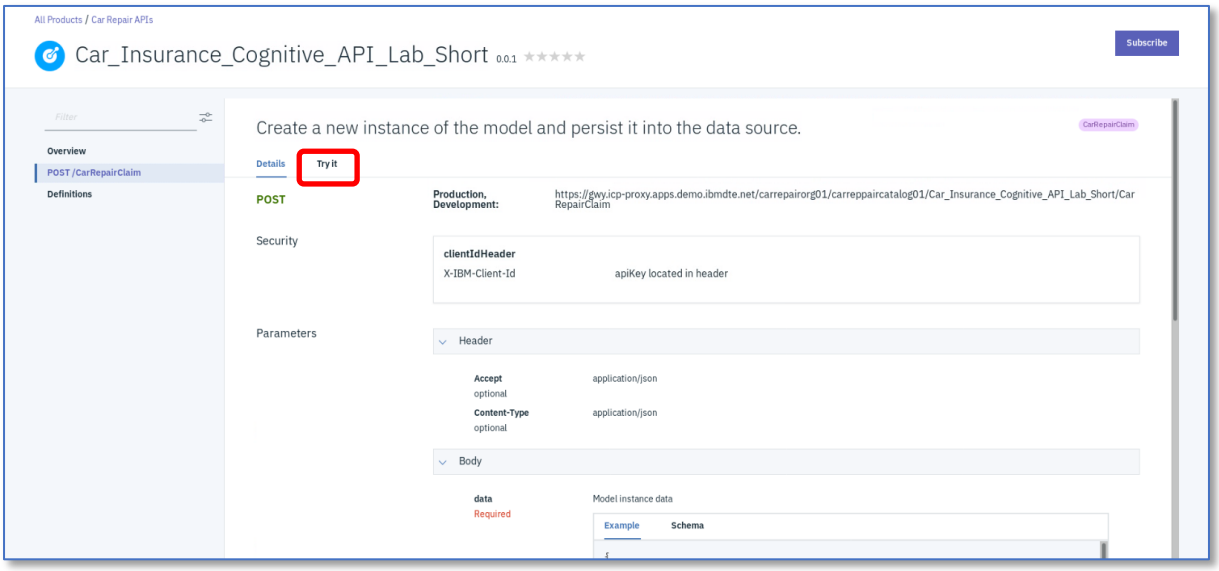
Don't worry that the plan isn't subscribable...

Click on the 'Car_Insurance_Co..' API

The portal gives you the ability to download the Open API, shows you the endpoint and how to manage the security.

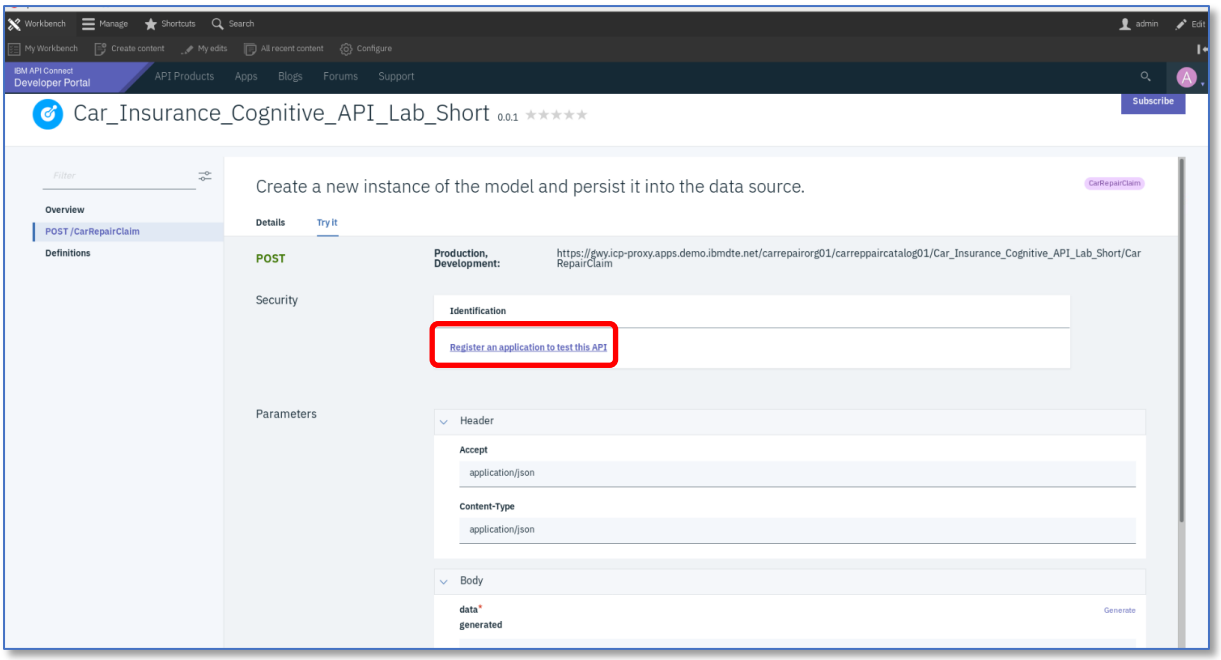


Click on ‘Post /CarRepairClaim’ (On the left)



This is our API – note that it still has the description from App Connect(!) – we’d usually change this in the API editor in API Connect.

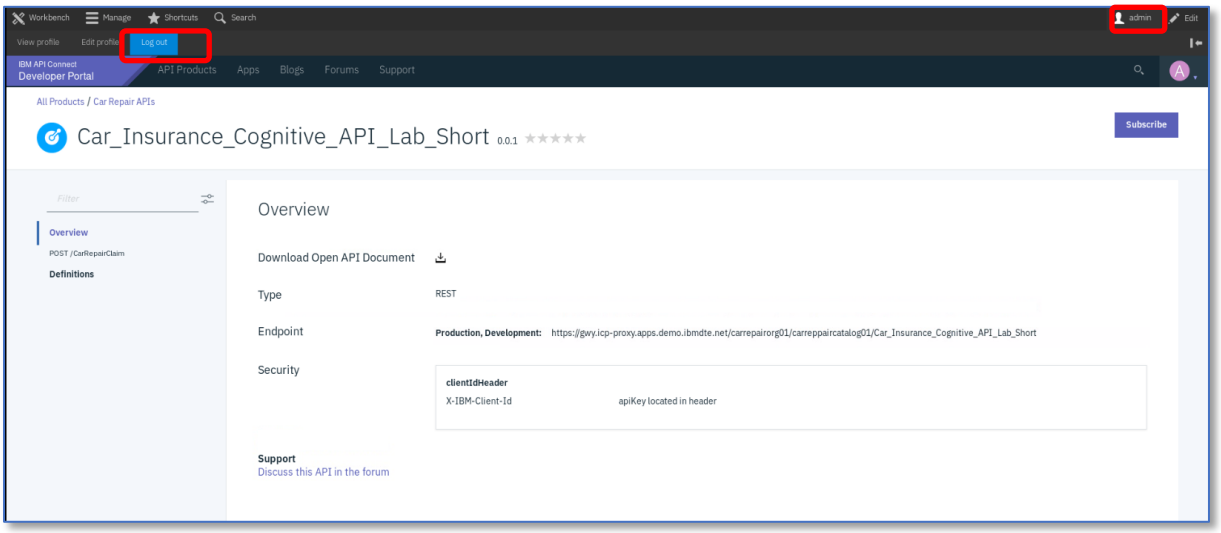
There is also a built-in tester, click ‘Try it’



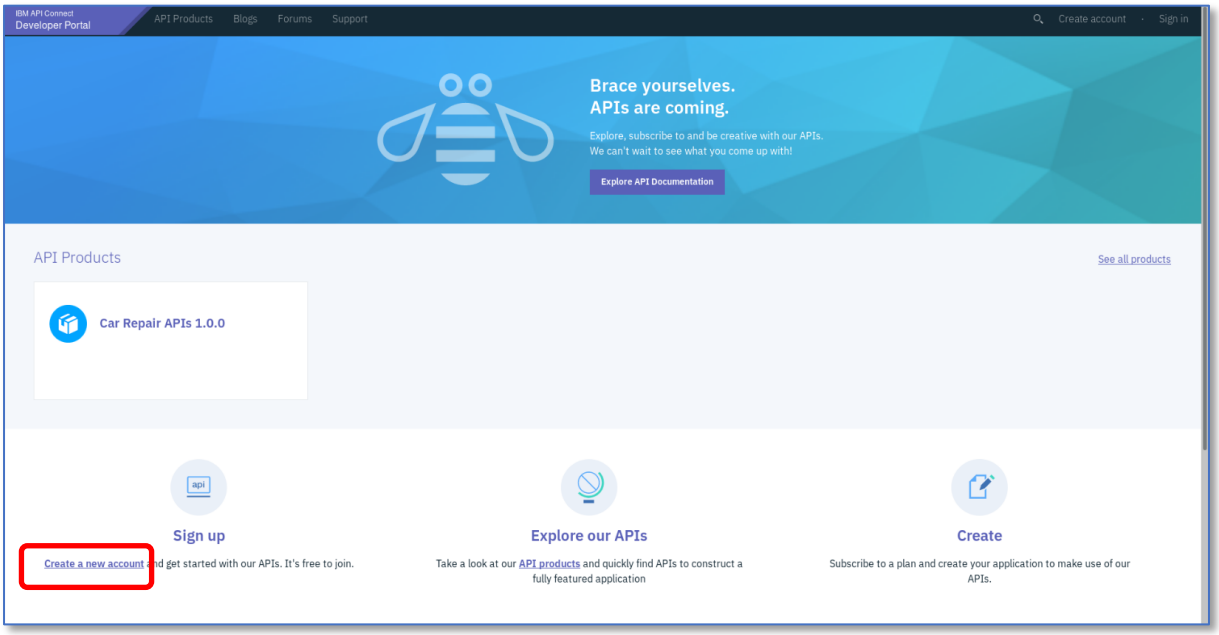
You'll notice that as we own the portal, we can't subscribe to the API and get an API key– we need to create a consumer to do that.

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

Click 'admin' (top right) and then 'logout' (top left)

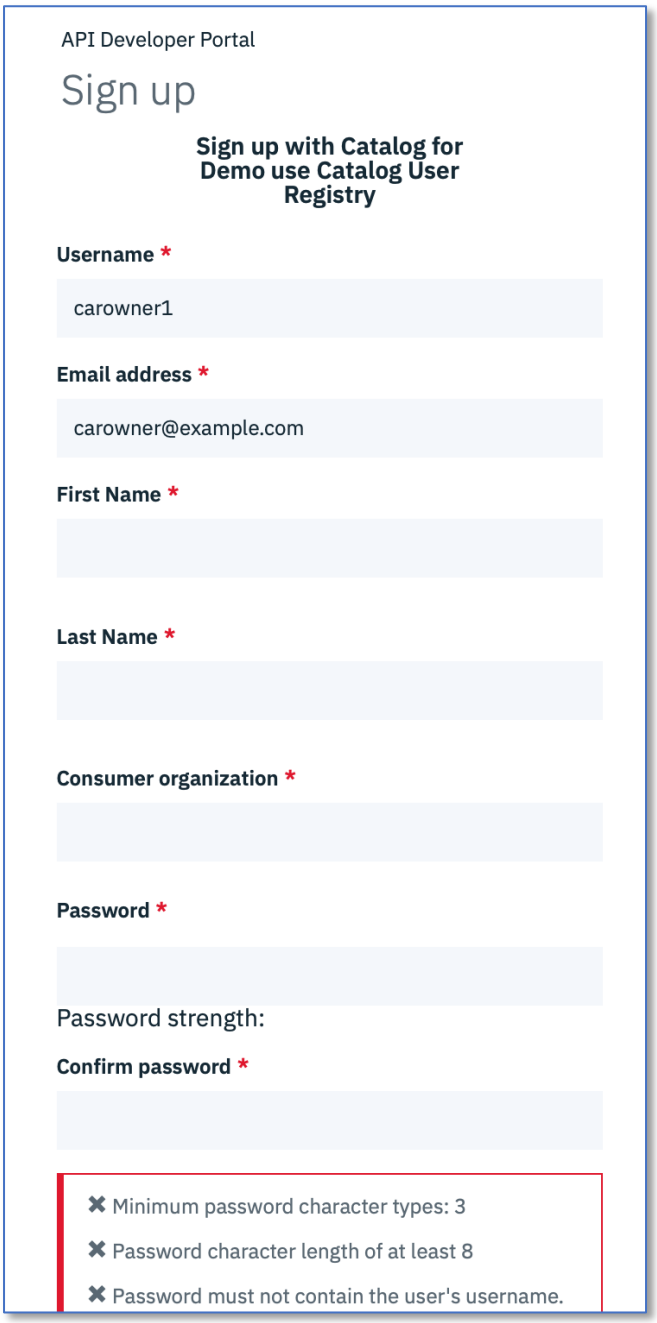


Now we’re back to how a new user would see the portal: So let’s sign up as one!



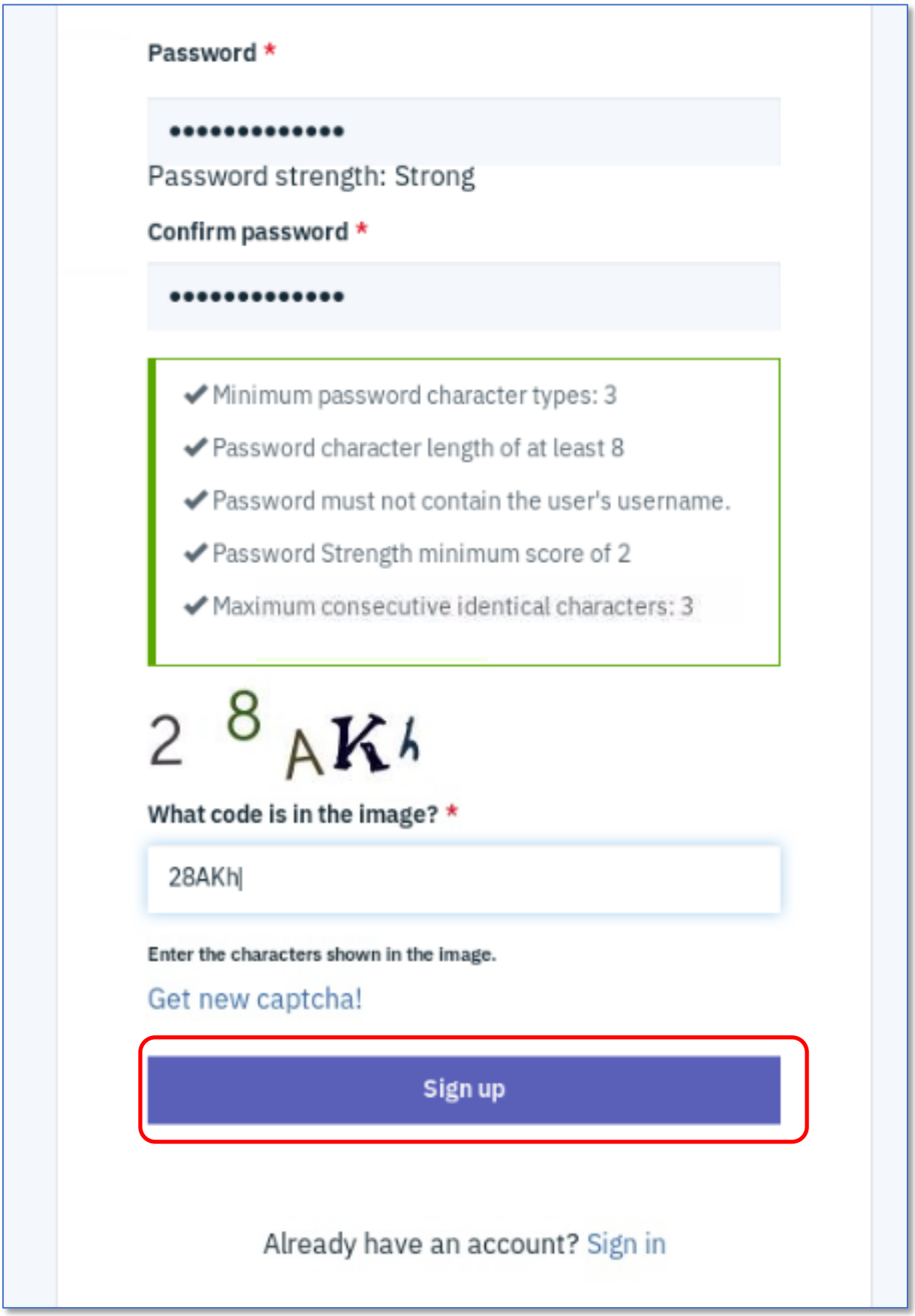
17 Signing up as a Consumer of our API using Portal Self-Service

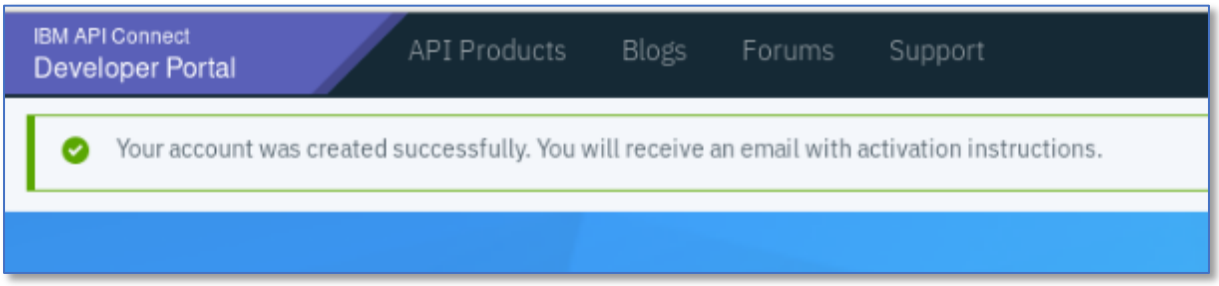
Click ‘Sign up – Create a new Account’



We’re going to create a consumer:
Username: carowner1
emailAddress: carowner@example.com
FirstName: Car
LastName: Owner
Consumer Organisation: Car Owners Inc
Password: engageibmAPI1

You'll need to solve the captcha! Then click 'sign up'

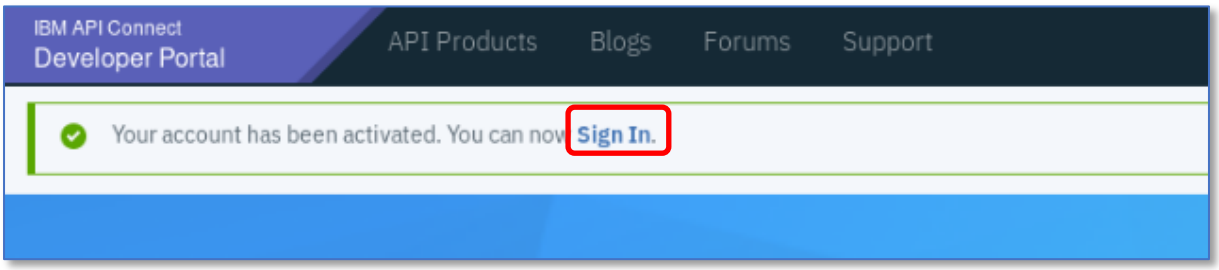




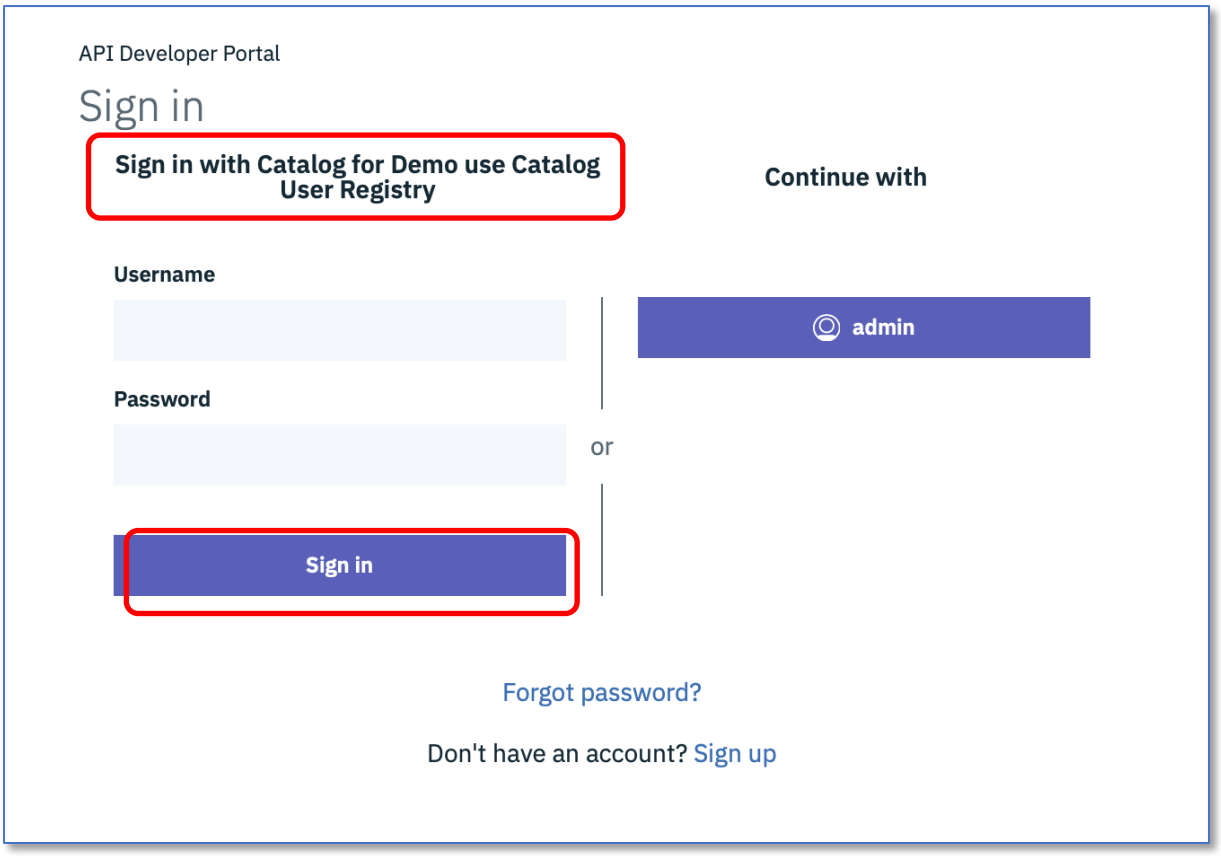
We'll need to get the email: You'll find it in your email page in your mailtrap.io. account.

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:



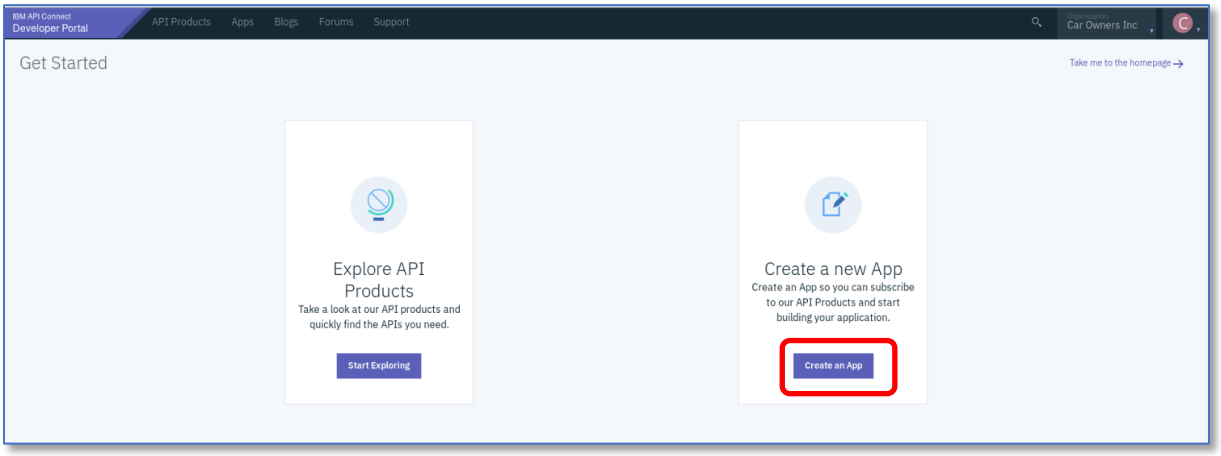
Click on Sign In;



Make sure you're using registry 'carrepaircatalog01'

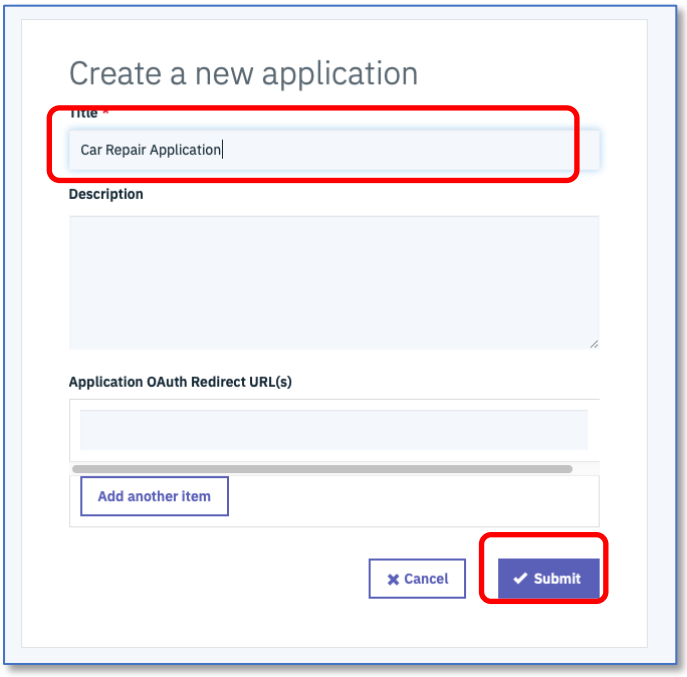
Sign in with your credentials you just created on the left side login: `carowner1/engageibmAPI1`

You'll get the following screen: Click on 'Create an App'

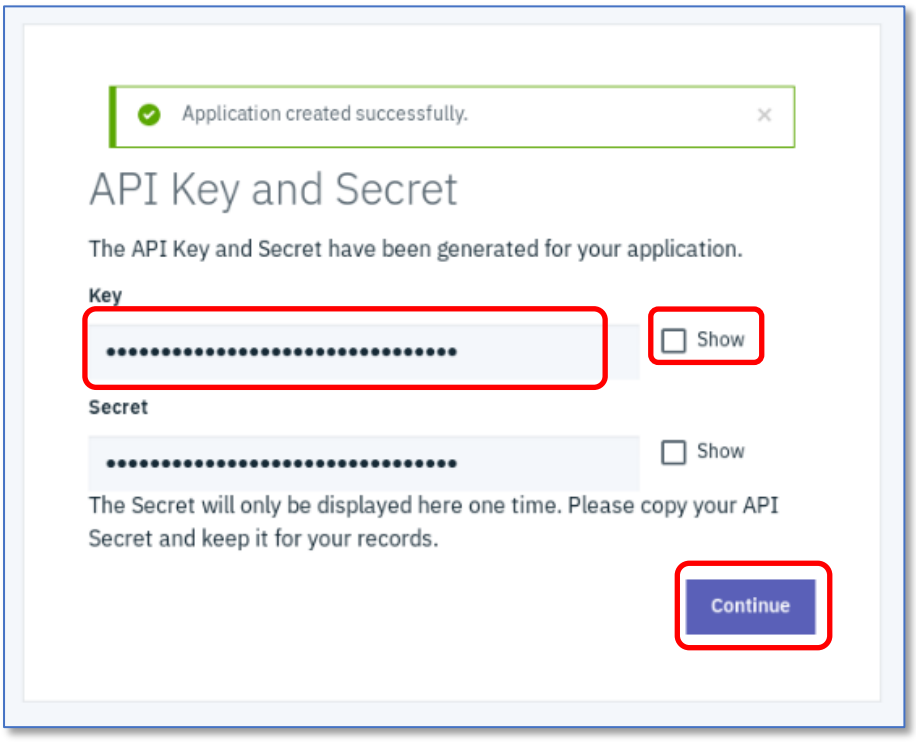


We're going to create a new application: This will give us an API key so we can call our APIs.

Give our App a title e.g Car Repair Application and click ‘Submit’:



This gives us an API key and secret. Click ‘Show’ to see them:

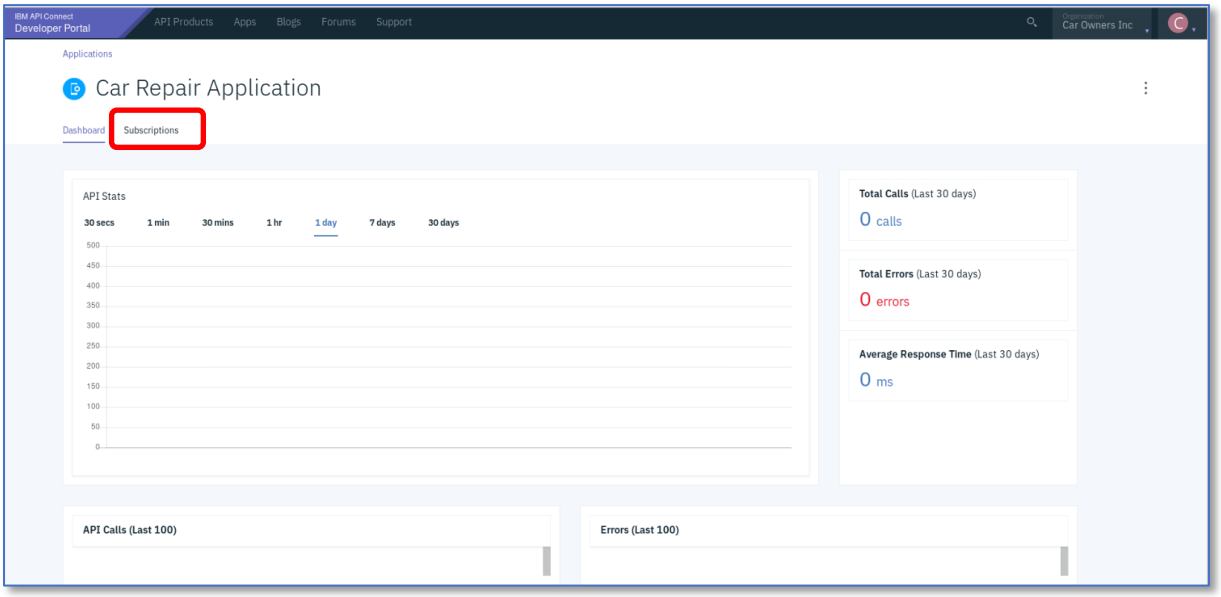


You’ll only ever be able to see the secret once here. For this lab, we haven’t asked for secrets so you won’t need to remember it.

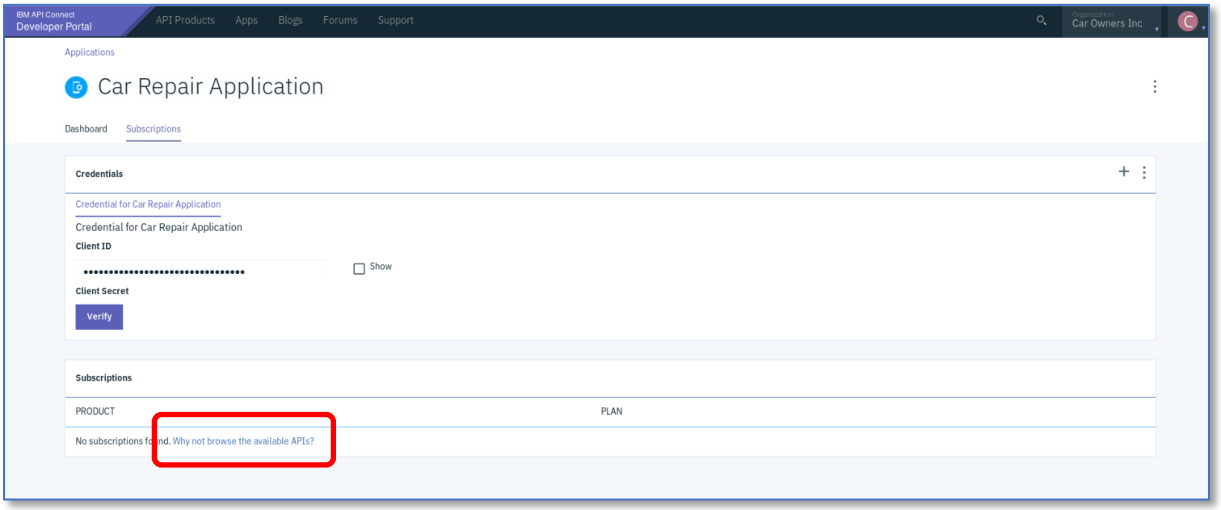
Click ‘Show’ to get your API key.

Copy it somewhere safe then click ‘continue’

You’ll now see the stats for your application. API Connect keeps the stats for both consumers and providers! At the moment, we don’t have an API calls, so no stats…

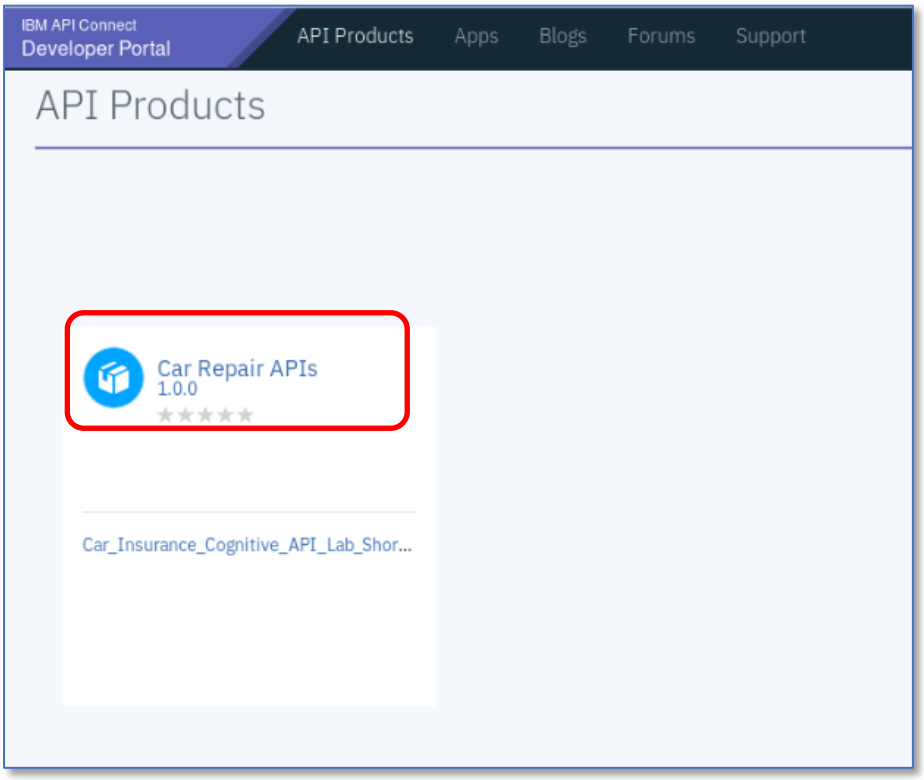


Click on the ‘subscriptions’ tab.

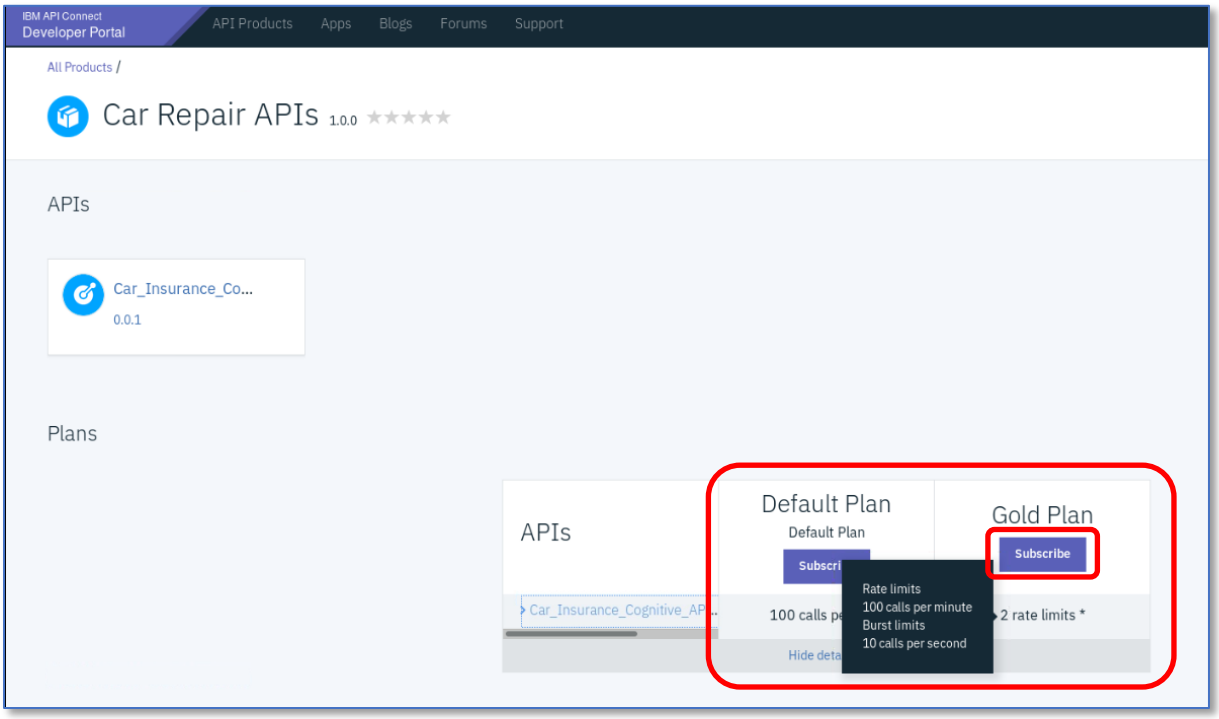


We’ve not subscribed to any APIs – click on ‘Why not browse the available APIs?’

There's only one API Product to subscribe to – normally there would be many..

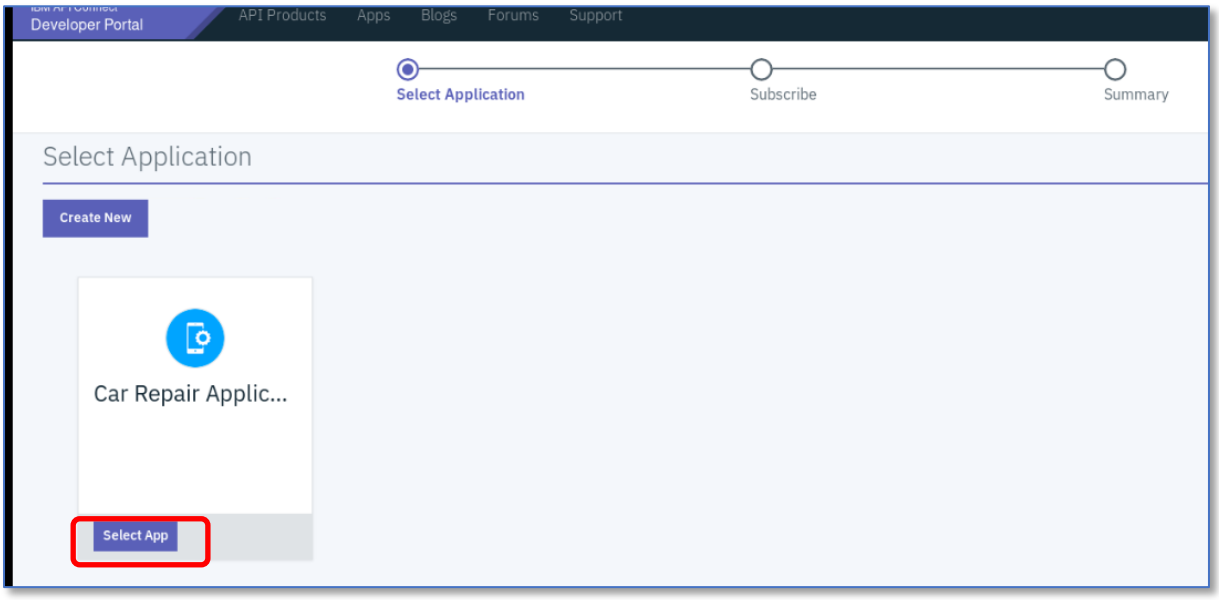


Click on the 'Car Repair APIs' product – you can now see the plans:

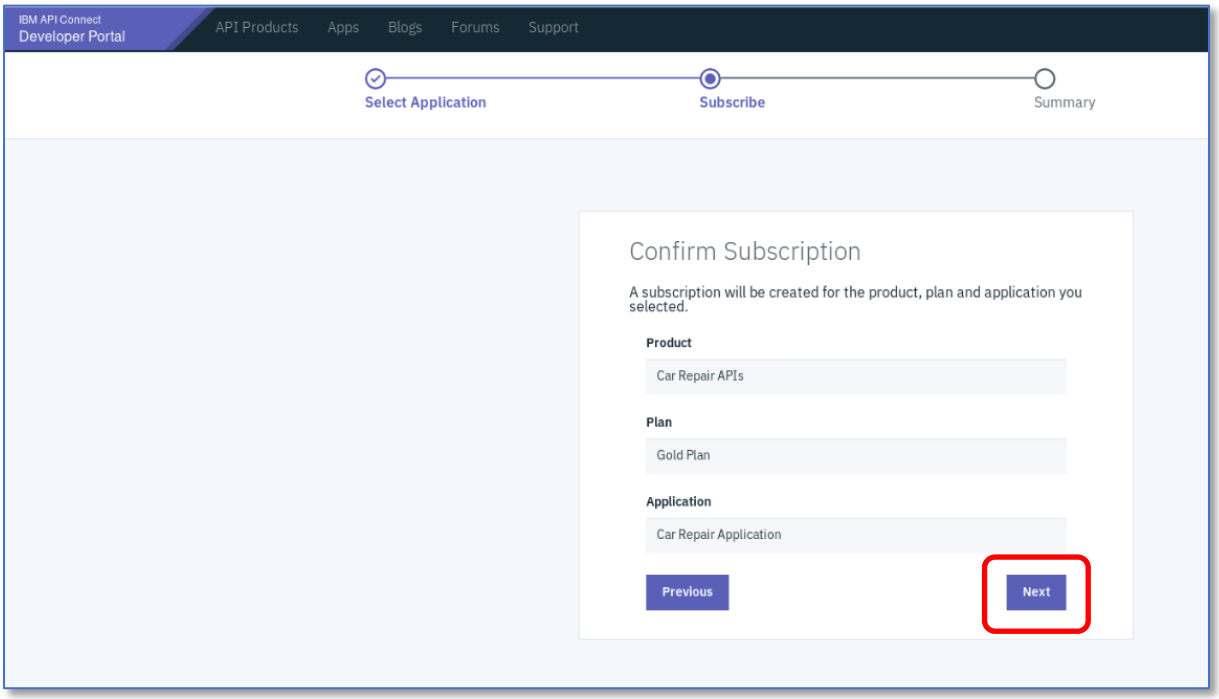


You'll need to hover over to get the limits – we want the gold plan. Click on 'subscribe'

We want to subscribe to the plan – but which application do we want to use to subscribe? We can have many applications but in this lab we've only created one:

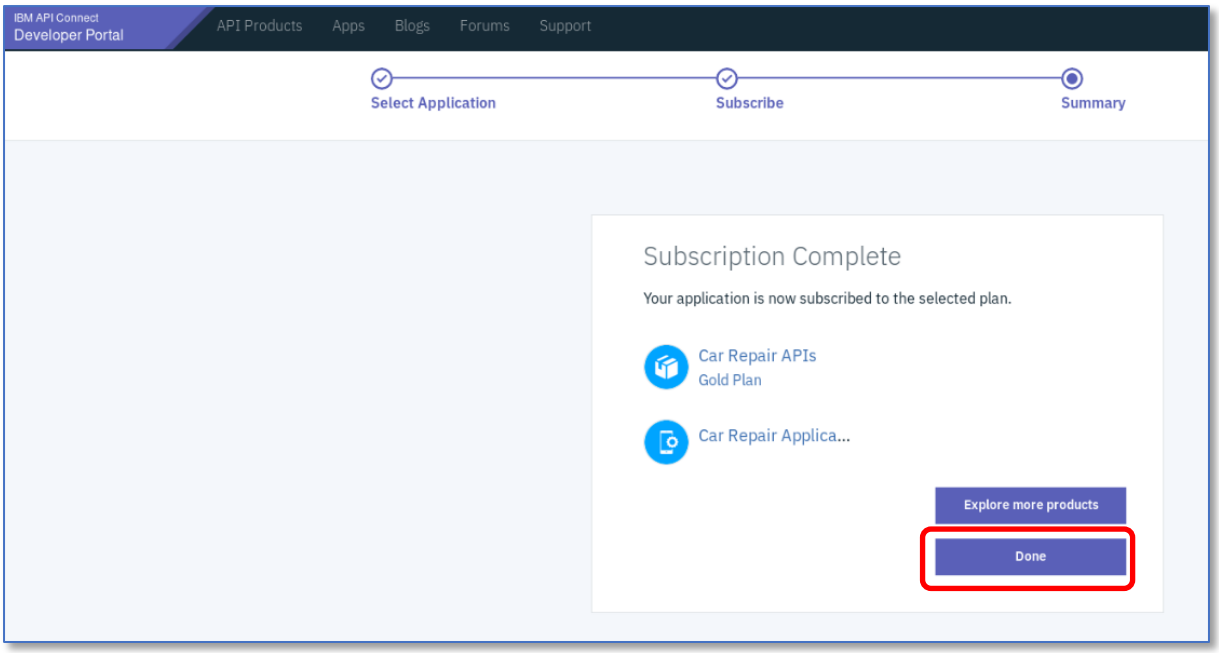


This is our application we created earlier – click 'Select App' for our Car Repair Application

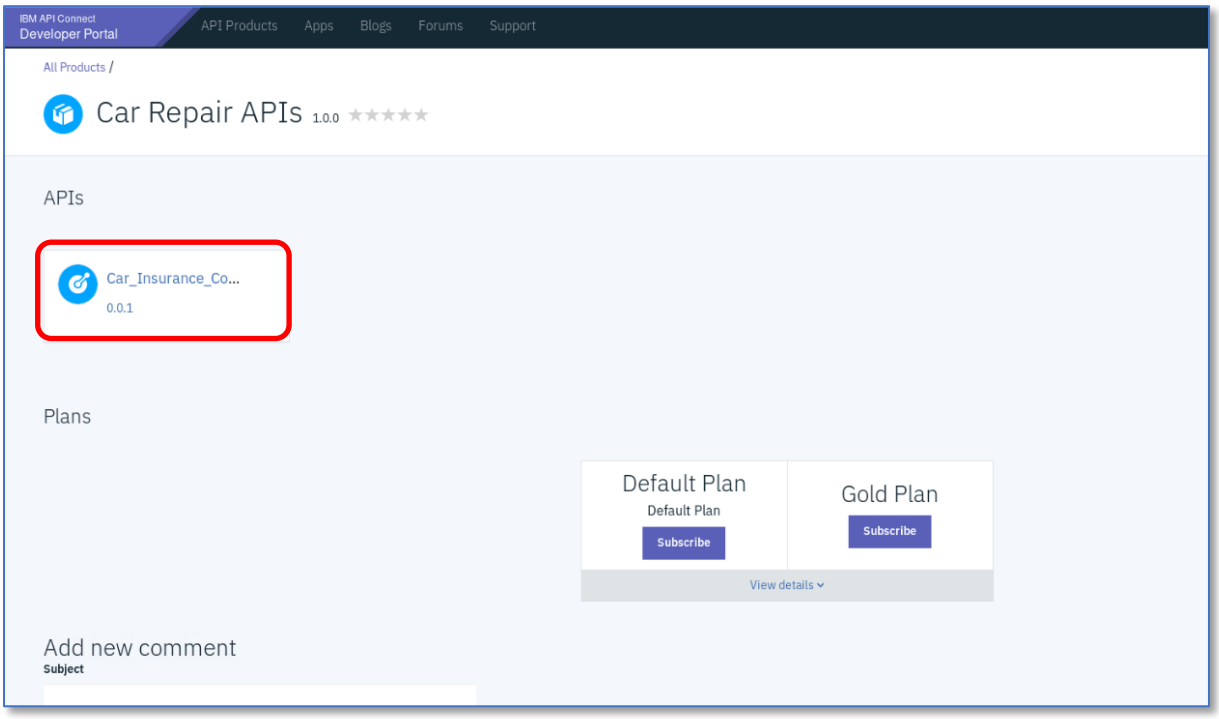


We now need to confirm our subscription – click Next.

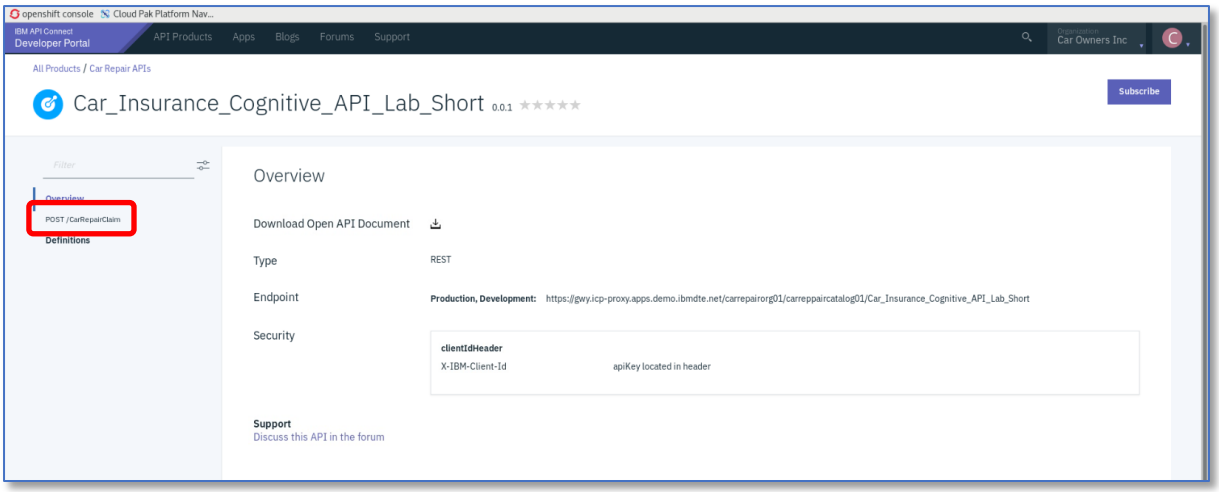
Click Next



Now click done – we are subscribed to our API!

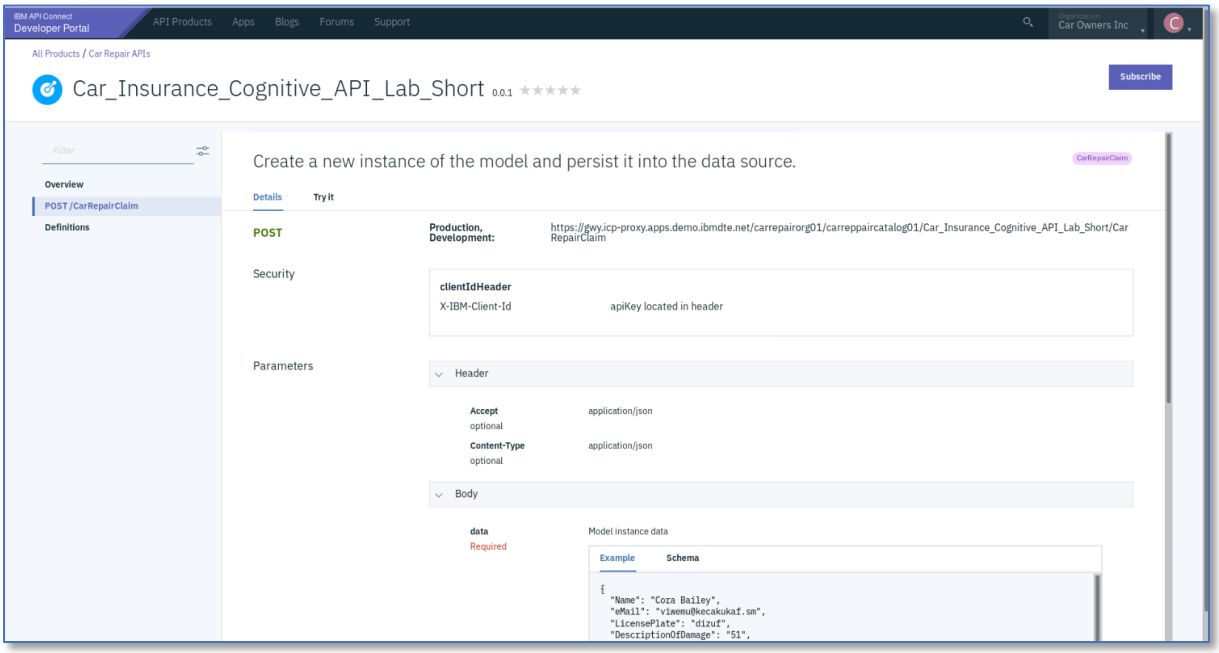


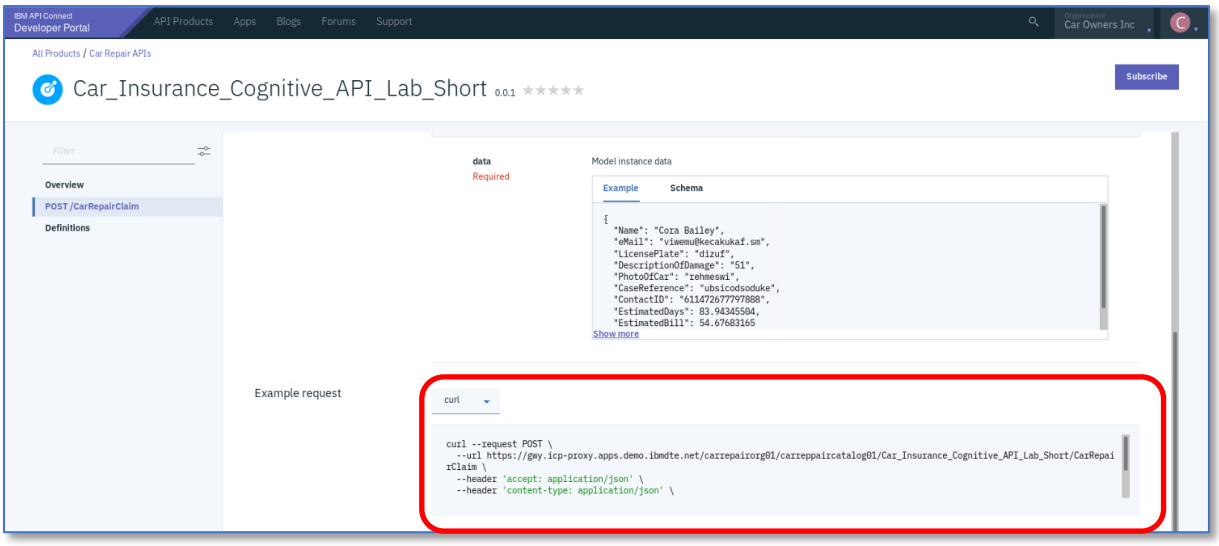
We're now back at the product screen – click on the API itself, not the plan.



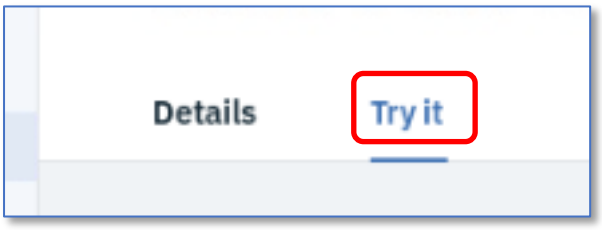
Click on POST – note the portal has everything you need to call your API – if you scroll down, it's even generated clients in various languages for you (that's how we created our test clients in curl in our scripts for this lab).

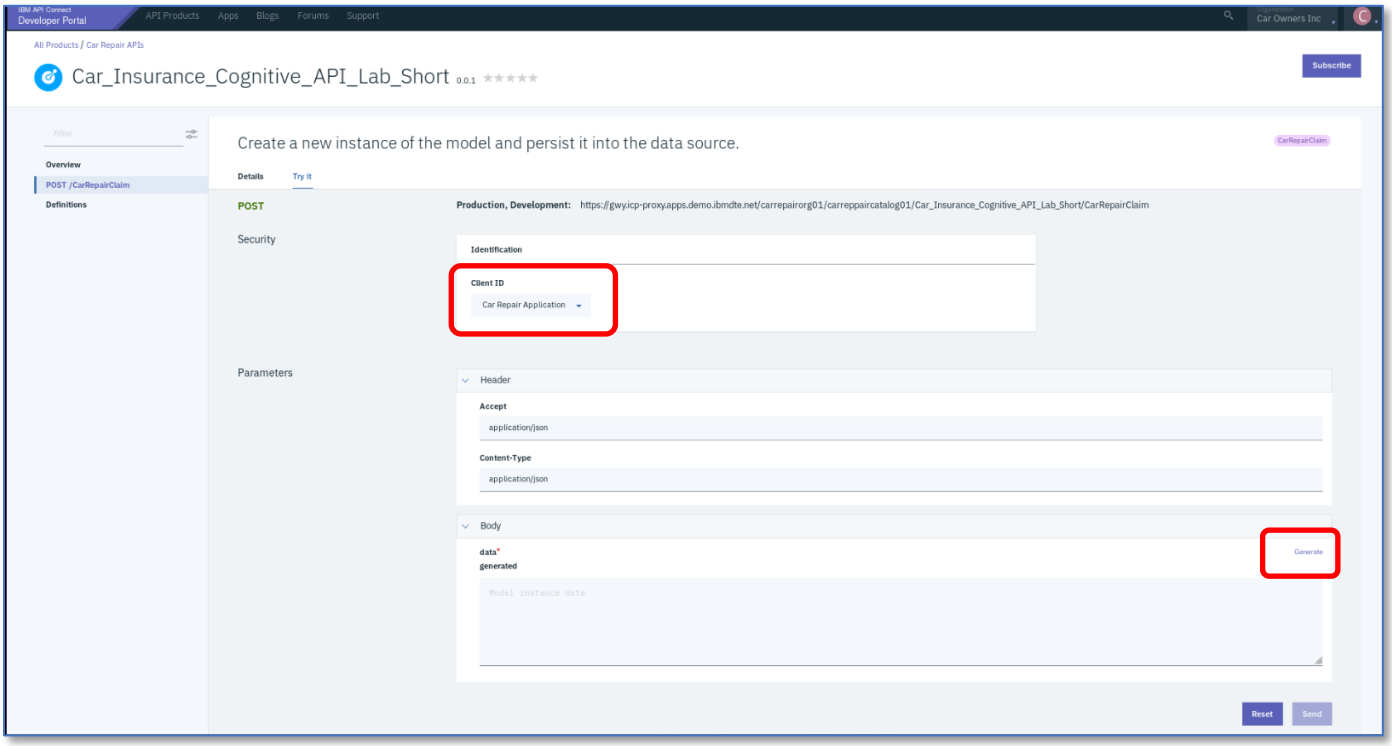
You can use the pull-down to generate client code for a number of languages.



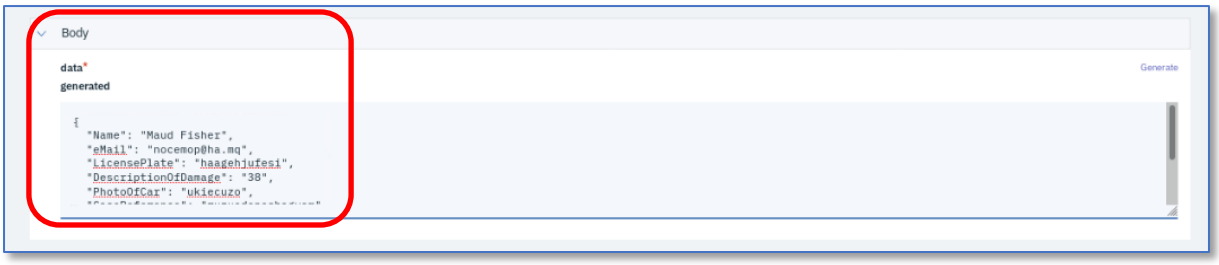


Scroll up to the top and click ‘Try it’





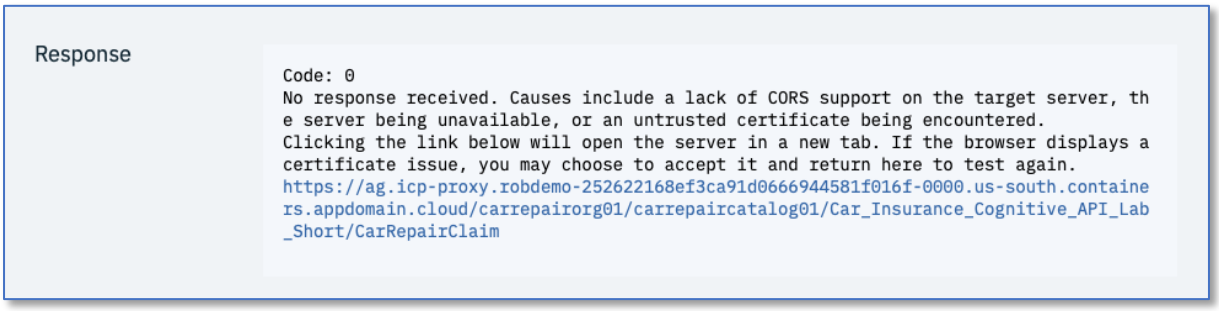
Note that your car repair application ClientID is setup for you.
Click ‘Generate’ – the portal will generate a request with random sample data for you:



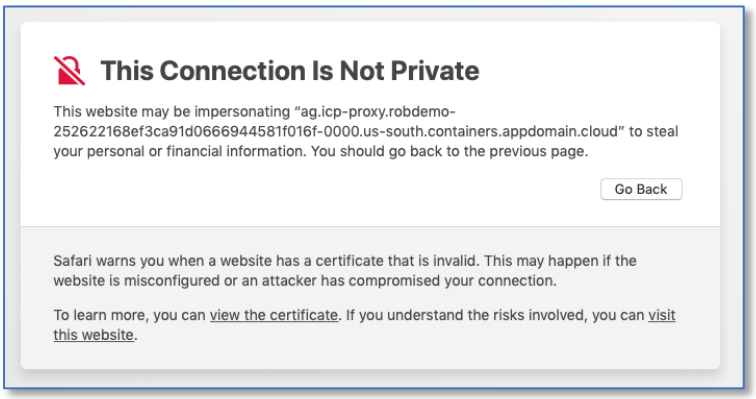
Now click ‘Send’

We got a response – our API is running and we’ve gone through the gateway.

IMPORTANT: You may get this instead. If you do, follow the instructions and send your API request again.



If you see something like the below, it’s the cause. Click ‘visit this website’ or accept the certificate or whatever your browser needs..

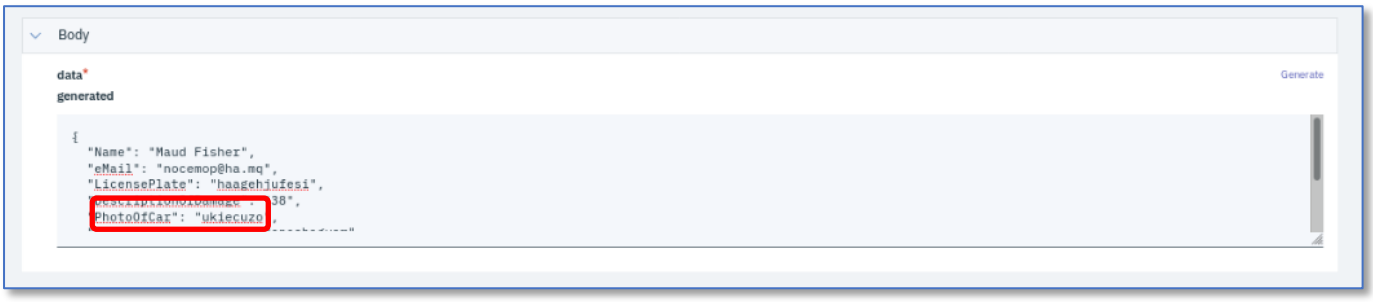


You may then see ‘405 Method not allowed!’ – this is fine – your browser is sending a ‘GET’ request and our API takes a ‘POST’. This is all fine. Go back to your original tab and try the request again.

When we sent our request we got a 400 error from Watson saying that the image cannot be processed. Like below.

This is good news! It means that API Connect has accepted your API call, called the API integration flow on App Connect, called the Watson Image Recognition Service which is now complaining there isn’t an image! All is actually working as designed!

And the reason is this:



The PhotoOfCar should be a base64 image – and the generated data is not… and hence Watson says ‘There is no image’

If you wish, you can get a base64 version of the test images from the github repository here https://github.com/IBM/cp4i-demos/tree/master/cognitive-car-insurance-claims
both `carbase64.txt` and `chickenbase64.txt` are available for you to view and copy the contents of: Put them into the “PhotoOfCar” field. Don’t forget to make

sure ‘LicensePlate’ is a short length – or you might get Salesforce complaining it’s too long like this:

```
Code: 400 Bad Request
Headers:
content-type: application/json
x-ratelimit-limit: name=Default rate-limit,100;
x-ratelimit-remaining: name=Default rate-limit,99;
{
    "error": {
        "statusCode": 400,
        "message": "The value in one of your Salesforce input fields is too long.",
        "errorDetail": "{\"name\":\"STRING_TOO_LONG\",\"errorCode\":\"STRING_TOO_LONG\",\"fields\":[\
"EngineeringReqNumber__c\"]}"
    }
}
```

You will see success output like this:

Response

```
Code: 201 Created
Headers:
content-type: application/json; charset=utf-8
x-ratelimit-limit: name=Default rate-limit,100;
x-ratelimit-remaining: name=Default rate-limit,99;
{
    "CaseReference": "5003z000027uWXVAA2",
    "EstimatedBill": 300,
    "EstimatedDays": 3,
    "LicensePlate": "abc123",
    "Name": "MIGUEL CLARK",
    "eMail": "wadilon@fukof.cg"
}
```

Unless you send the chicken picture in base64 in which case you’ll get:

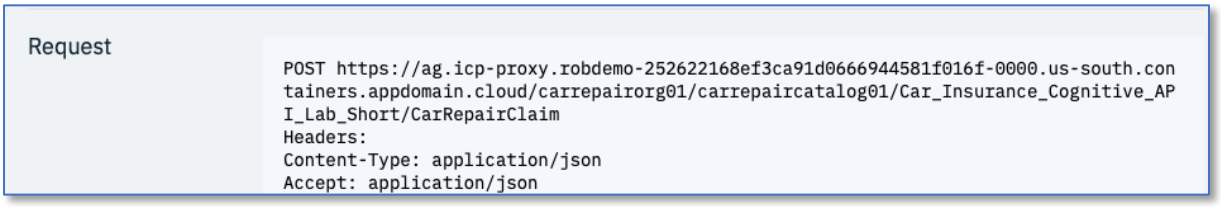
Response

```
Code: 400 Bad Request
Headers:
content-type: application/json
x-ratelimit-limit: name=Default rate-limit,100;
x-ratelimit-remaining: name=Default rate-limit,99;
{
    "error": {
        "statusCode": 400,
        "message": "There is no car in this image, please resubmit"
    }
}
```

We do, however also have test scripts based on the curl, with pictures ready to go!

We'll need to set up the environment variables to point to the right place, just as we did to test before.

The easiest way to find the info you need is by looking in the 'Request' box of the 'try it' tester:



It also shows your API key (We've cut the screenshot) as the 'X-IBM-Client-Id'.

In the terminal window, do:

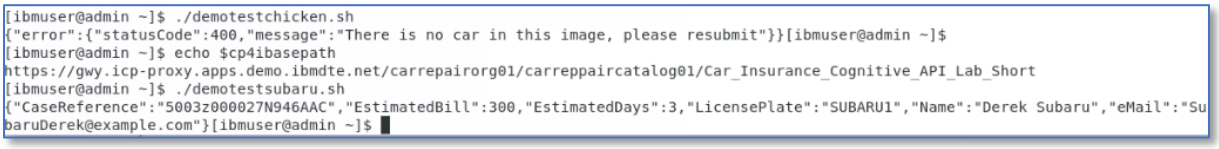
```
export cp4ibasepath=https://<<yourserver, begins with
ag>>/carrepairorg01/carreppaircatalog01/Car_Insurance_Cognitiv
e_API_Lab_Short
```
(note – you don't need the '/CarRepairClaim' – the test script knows about that)

```
export cp4iclientid=<<yourapikey>>
```

Then you can do:
`./demotestchicken.sh`

And `./demotestcar.sh`

You'll get similar to this:



You can then go into Salesforce and check that your Subaru case has been placed into Salesforce – just as it was before.

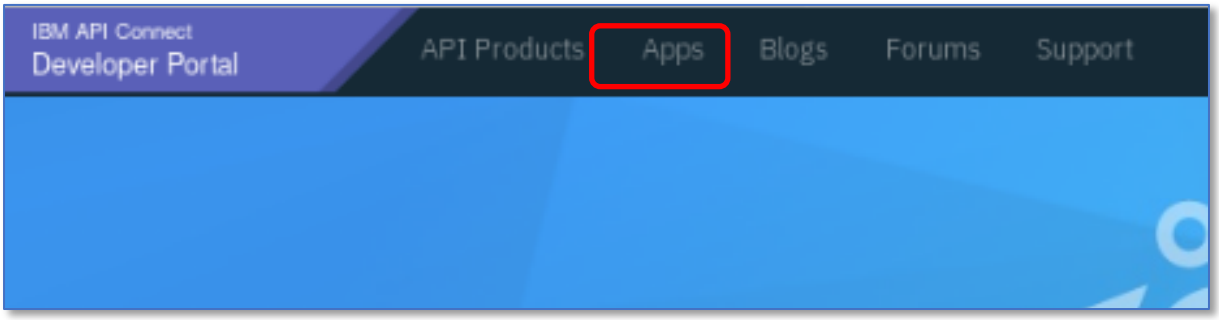
We are calling the same API with the same connectors. What's different is that we are going through a secure gateway with rate limits and APIKey security. Once the gateway validates the request, it passes it down to App Connect to do the integration logic.

17.1 Viewing the API Statistics in the Portal

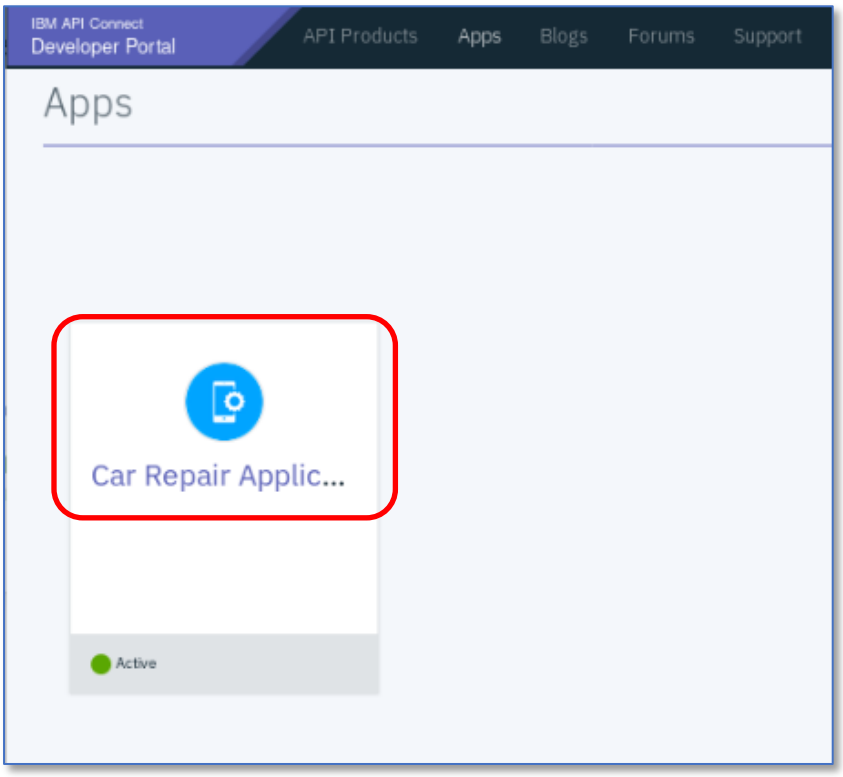
If you go back to the portal, you can check on your stats for your API:

Go to the carrepair portal site

and sign in as carowner1/engageibmAPI1
Click on 'Apps' in the top menu:

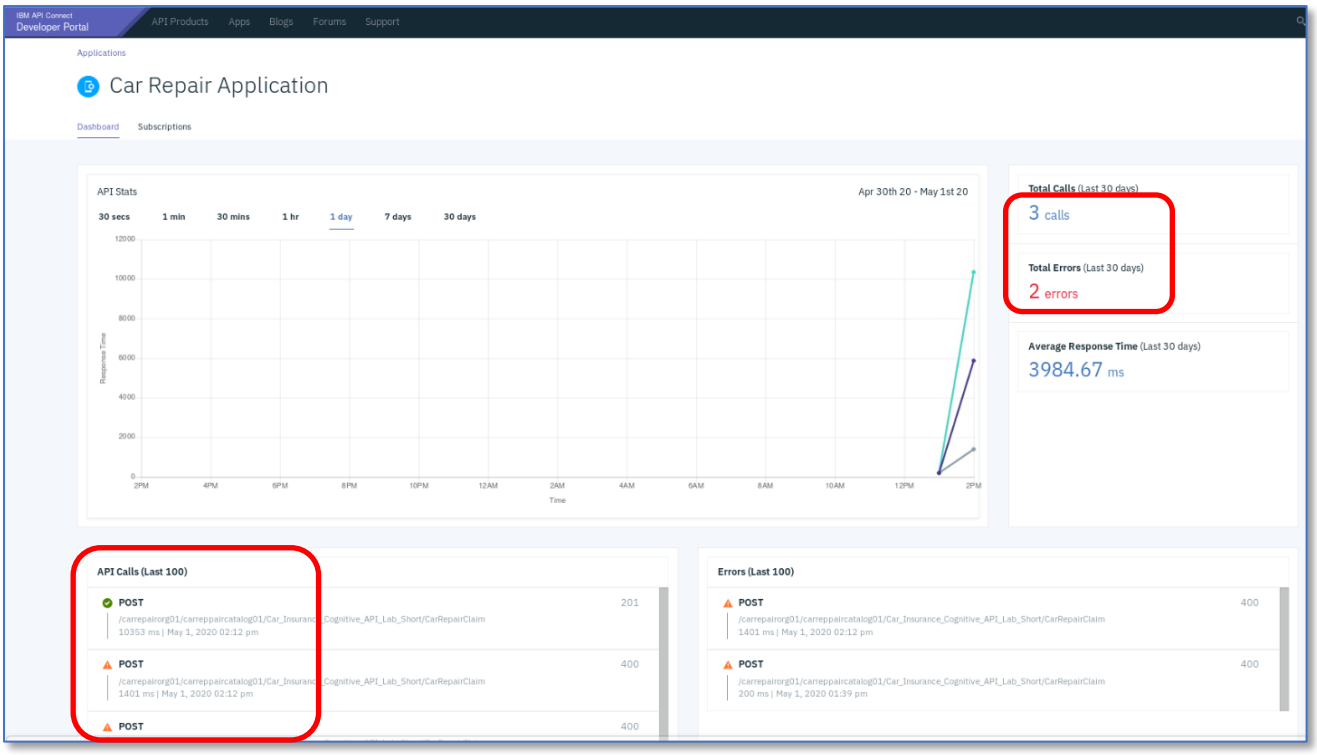


Now click on the car repair application:



You can see your API stats – for example, here we can see three calls with 2 errors:

We had 1 call with no picture, one call with no car (the chicken) and one call which created a Salesforce case (the call with the Subaru car image). If you make more calls, you'll see more stats.


IBM API Connect
Developer Portal
API Products
Apps
Blogs
Forums
Support
Applications
Car Repair Application
Dashboard
Subscriptions
API Stats
Apr 30th 20 - May 1st 20
30 secs
1 min
30 mins
1 hr
1 day
7 days
30 days
Response Time
Time
Total Calls (Last 30 days)
3 calls
Total Errors (Last 30 days)
2 errors
Average Response Time (Last 30 days)
3984.67 ms
API Calls (Last 100)
POST
201
/carrepairorg01/carreppaircatalog01/Car_Insurance_Cognitive_API_Lab_Short/CarRepairClaim
10353 ms | May 1, 2020 02:12 pm
POST
400
/carrepairorg01/carreppaircatalog01/Car_Insurance_Cognitive_API_Lab_Short/CarRepairClaim
1401 ms | May 1, 2020 02:12 pm
POST
400
Errors (Last 100)
POST
400
/carrepairorg01/carreppaircatalog01/Car_Insurance_Cognitive_API_Lab_Short/CarRepairClaim
1401 ms | May 1, 2020 02:12 pm
POST
400
/carrepairorg01/carreppaircatalog01/Car_Insurance_Cognitive_API_Lab_Short/CarRepairClaim
200 ms | May 1, 2020 01:39 pm

18 Finished! And Summary

Well done, you've completed the lab!

Some more water, a beverage or even a snack may be in order – unless you are fasting of course.

It was a lot of pages, but well done for getting here. What did you do again?

- Created a series of SaaS endpoints to do the lab with. You can use these again after this lab – They don't expire.
- Created secure managed connections to each of these endpoints using the CP4I connectors
- Created an API and API integration flow to process car repair claims – with minimal (or no) code. Apart from the odd spreadsheet style formula and testing scripts – did you see any code?
- Tested the connections from within the tooling, building your integration interactively.
- Used test scripts based on curl commands generated from the API portal to test your integration
- Deployed your API as a highly available, scalable resilient Kubernetes deployment of containers and pods onto CP4I runtime on OpenShift
- Created secure Kubernetes 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 – this topology can be customized to match your business.
- Pushed the API definition from App Connect to API connect to manage it.
- Added an APIKey security policy to keep your API secure
- 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-serice features including the interactive tester.
- Re-used the automated test-script based on the generated curl scripts to test your API
- Viewed your API statistics in the portal analytics.

Not bad for a few hours' work!

If you want to try the 'Extension Scenario' with ServiceNow – the information you need is below:

19 Extension Scenario:

If you'd like to try the extended scenario, there aren't step by step instructions – but you should be able to build it yourself – it's not different to what we've already been doing.

The steps are the same – just go back to the appropriate part of the lab.

Remember that many of the lab steps are '1 time only' – you'll find building a second API a lot quicker!

19.1 You will need

19.1.1 A Watson Language Translation Service

We already build this in the main lab. You have the connector configured as well

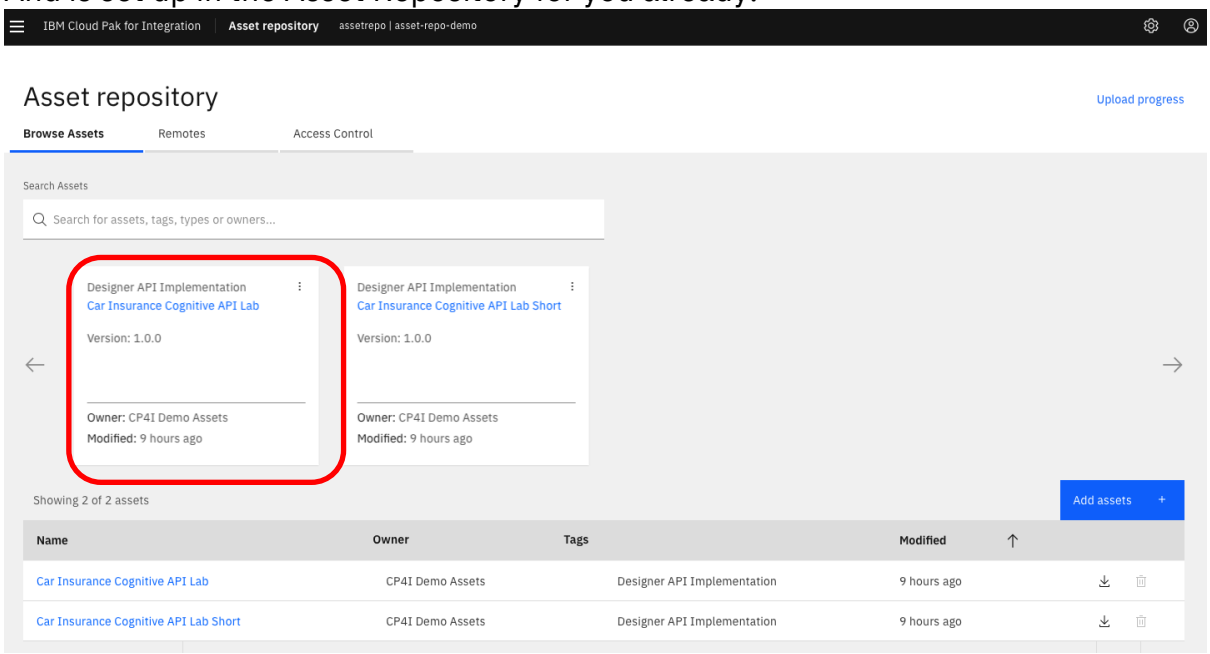
19.1.2 A ServiceNow instance

You can create a free developer ServiceNow account at
https://developer.servicenow.com
Instructions for this are at the end of this lab guide

19.1.3 The 'Extended' Integration flow to Import into Designer

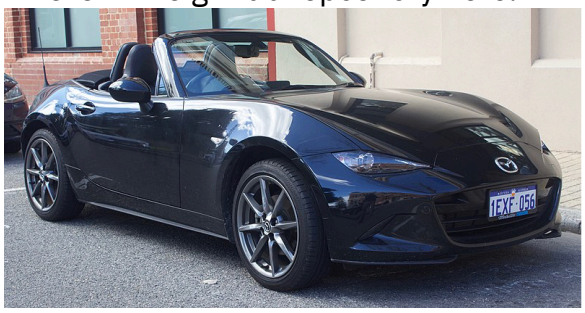
This is hosted on the git repository here: https://github.com/IBM/cp4i-demos/blob/master/cognitive-car-insurance-claims/Car%20Insurance%20Cognitive%20API%20Lab.yaml

And is set up in the Asset Repository for you already:



19.1.4 A picture of a convertible/roadster car and test scripts with it in.

This is in the github repository here:



https://github.com/IBM/cp4i-demos/blob/master/cognitive-car-insurance-claims/roadster.jpg

Script:
https://github.com/IBM/cp4i-demos/blob/master/cognitive-car-insurance-claims/demotestroadster.sh

The URL to download the script is https://raw.githubusercontent.com/IBM/cp4i-demos/master/cognitive-car-insurance-claims/demotestroadster.sh

So use `curl https://raw.githubusercontent.com/IBM/cp4i-demos/master/cognitive-car-insurance-claims/demotestroadster.sh -o demotestroadster.sh`

And
`curl https://raw.githubusercontent.com/IBM/cp4i-demos/master/cognitive-car-insurance-claims/roadster.jpg -o roadster.jpg`

Don't forget to do `chmod +x demotestroadster.sh` to make it executable.

19.2 List of Tasks

- Create a free ServiceNow developer instance (instructions at end)
- Obtain the credentials for that instance (instructions at end)
- Set up the ServiceNow connector in designer – similar to how we did it for Salesforce and Watson
  DON'T FORGET TO RENAME THE ACCOUNT TO "App Connect Trial"
- Import the flow to Designer use the asset without 'short' in its name.

- Test the new flow, using the `demotestroadster.sh` script for a Roadster. Pictures of convertibles/roadsters will mean an incident will be created in ServiceNow as well as a case in SalesForce.
  Remember that the base path will be different (Will not have 'Short' in it)
  Don't forget to check the Spanish description!

- Update the `credentials.yaml` file with the ServiceNow credentials.

- Update the Kubernetes secret to add the ServiceNow credentials. The easiest way is to do '`oc delete secret carrepaircreds01`' and then '`./generateSecrets.sh carrepaircreds01`' to recreate it. Alternatively, you can create another secret if you wish.

- Export the .bar file from Designer

- Deploy the .bar file to CP4I using the Dashboard – you can use the same secret and run them both at the same time.

- Test the flow on CP4I

- Push the API to API connect using the GUI.
  You can use the same product or a different one. Use the same Provider Organization, same Catalog, same ClientID and ClientSecret in the 'Share REST APIs'.
  You don't need to do the registration again – that's a one-time. Similar for Catalog, Provider Org, Portal, Gateway etc etc

- Add security to your API. Add a plan to your product.

- Publish your product and API

- Make sure your application is subscribed to the new API – you don't need a new consumer! You should be able to use the same API Key for the same Application.

- Test your new API – You're done! Good Luck!

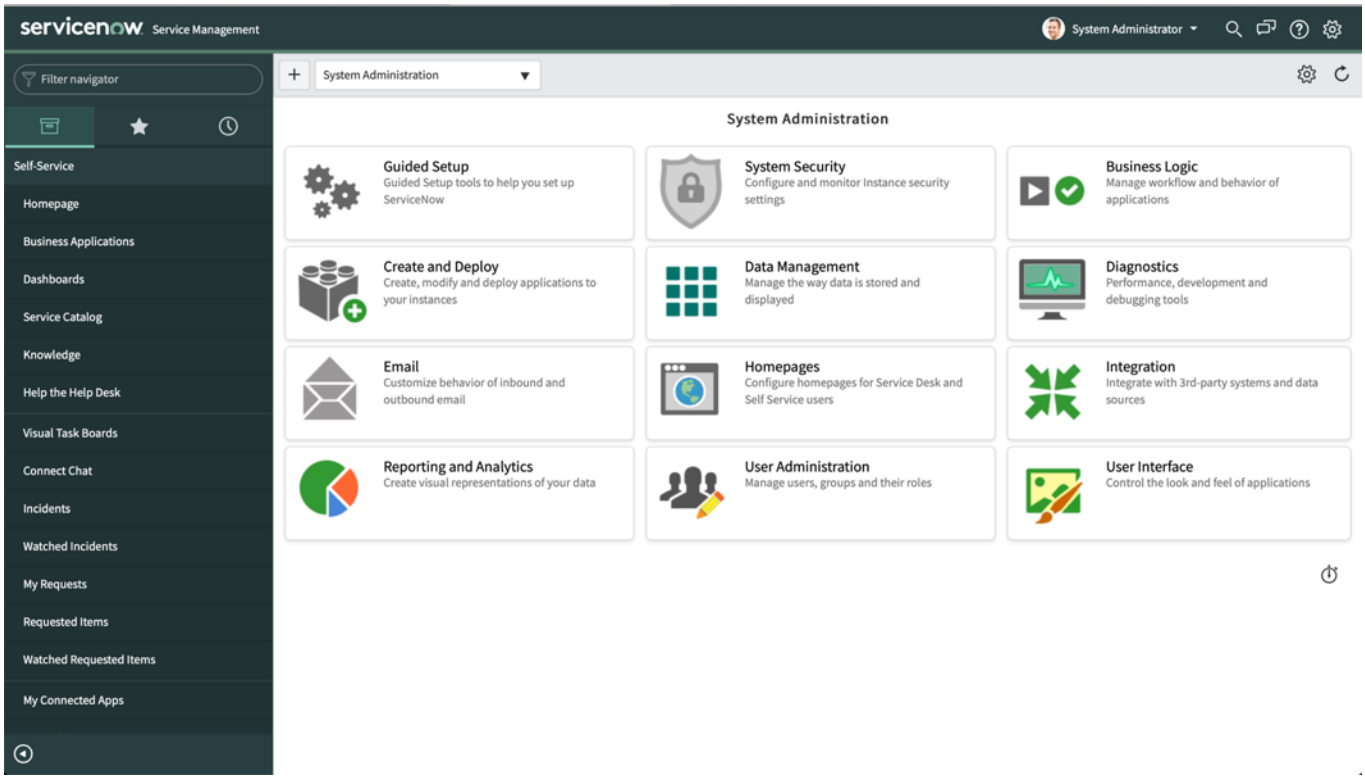
19.3 Setting up ServiceNow and the connector

(These instructions are not as step-by-step as in the lab – and apologies for the inconsistent formatting, but should give you the information that you need)

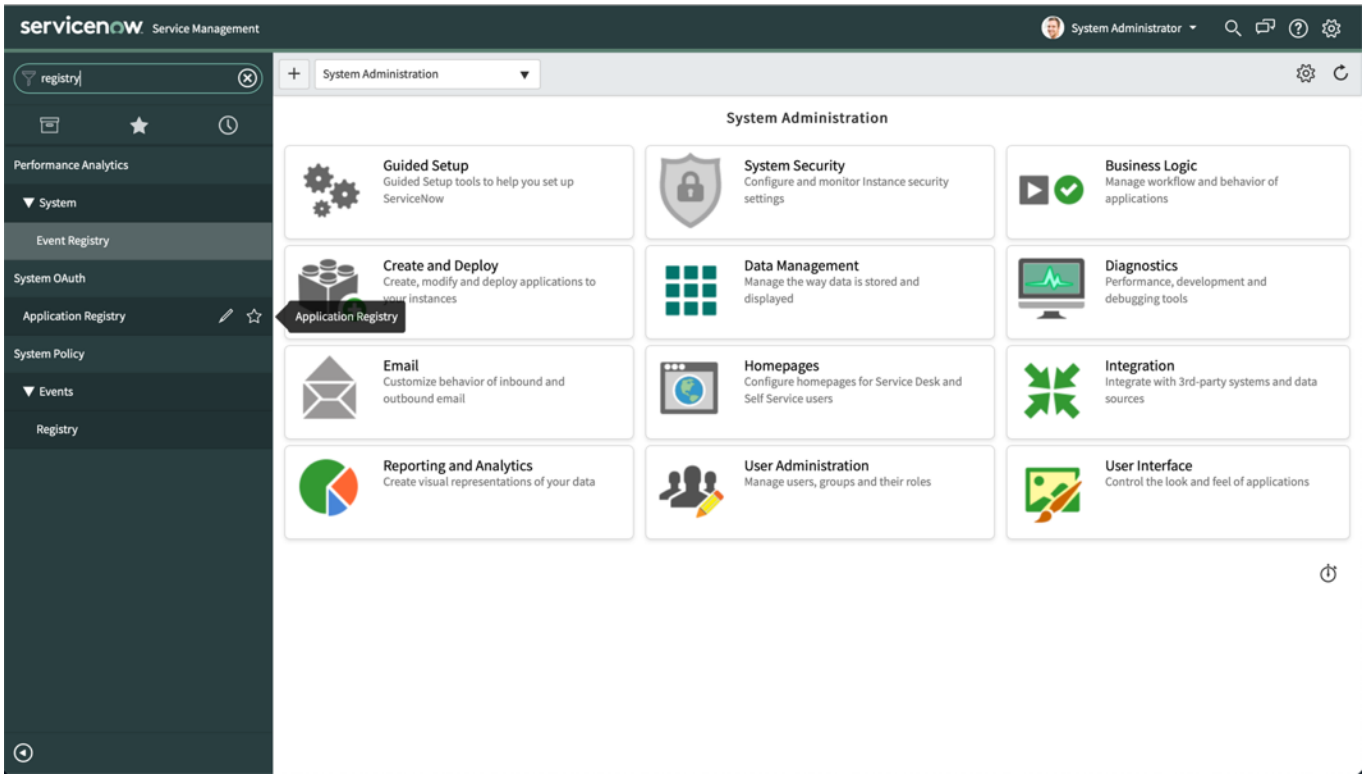
```
accounts:
  servicenow:
    - name: "<same name as cloud account>"
      credentials:
        authType: "oauth2web"
        accessToken: "<accessToken as shown generated below>"
        refreshToken: "<refreshToken as shown generated below>"
            clientID: "<ClientID as shown generated below>"
        clientSecret: "<ClientSecret as shown generated below>"
      endpoint:
        url: "Your ServiceNow instance URL"
```

Note: If you want to use OAuth token-based credentials then you will need to download and install Postman app. You do not need Postman for basic auth, which is illustrated using UI in following instructions

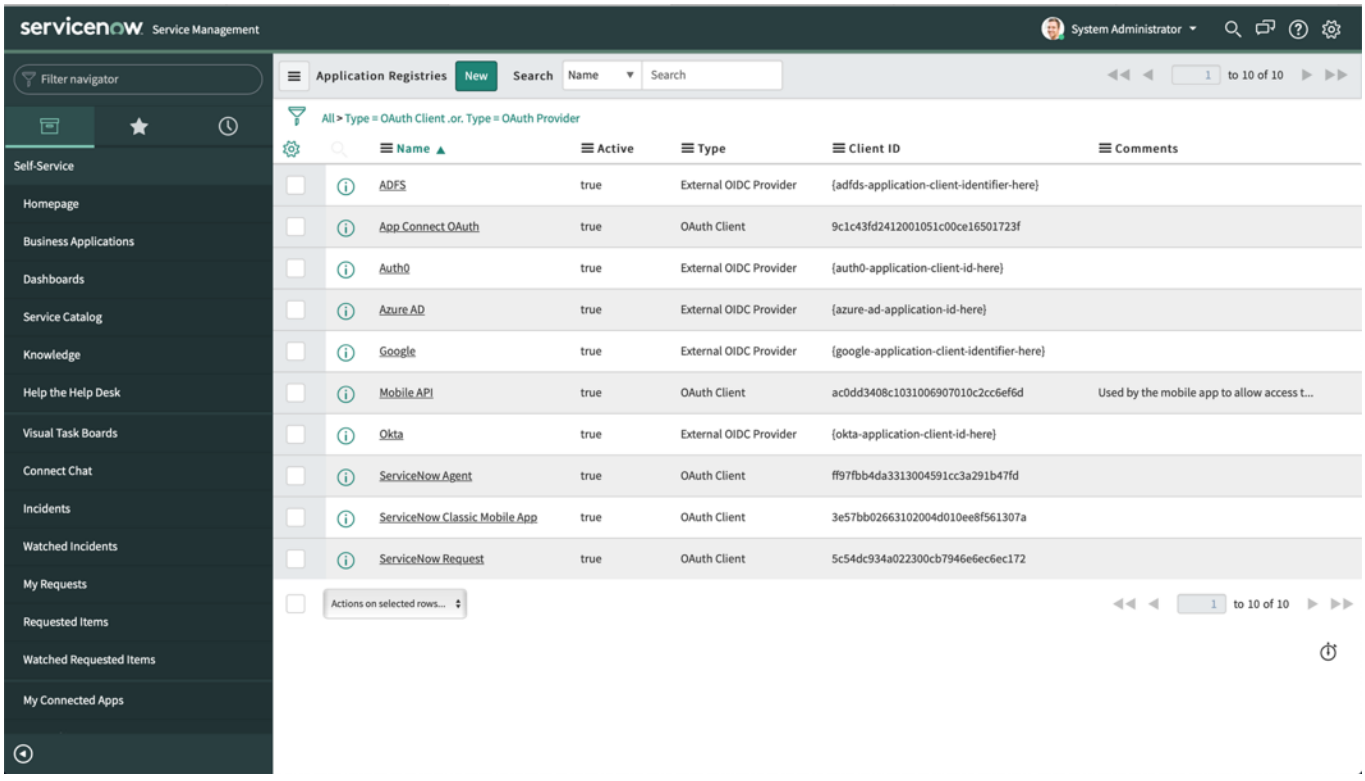
To get started you will require admin level access to your ServiceNow account. If you want to create a free ServiceNow account to test out App Connect, you'll have to register for a ServiceNow Account here. Once your account is activated, you can request a ServiceNow personal developer instance.

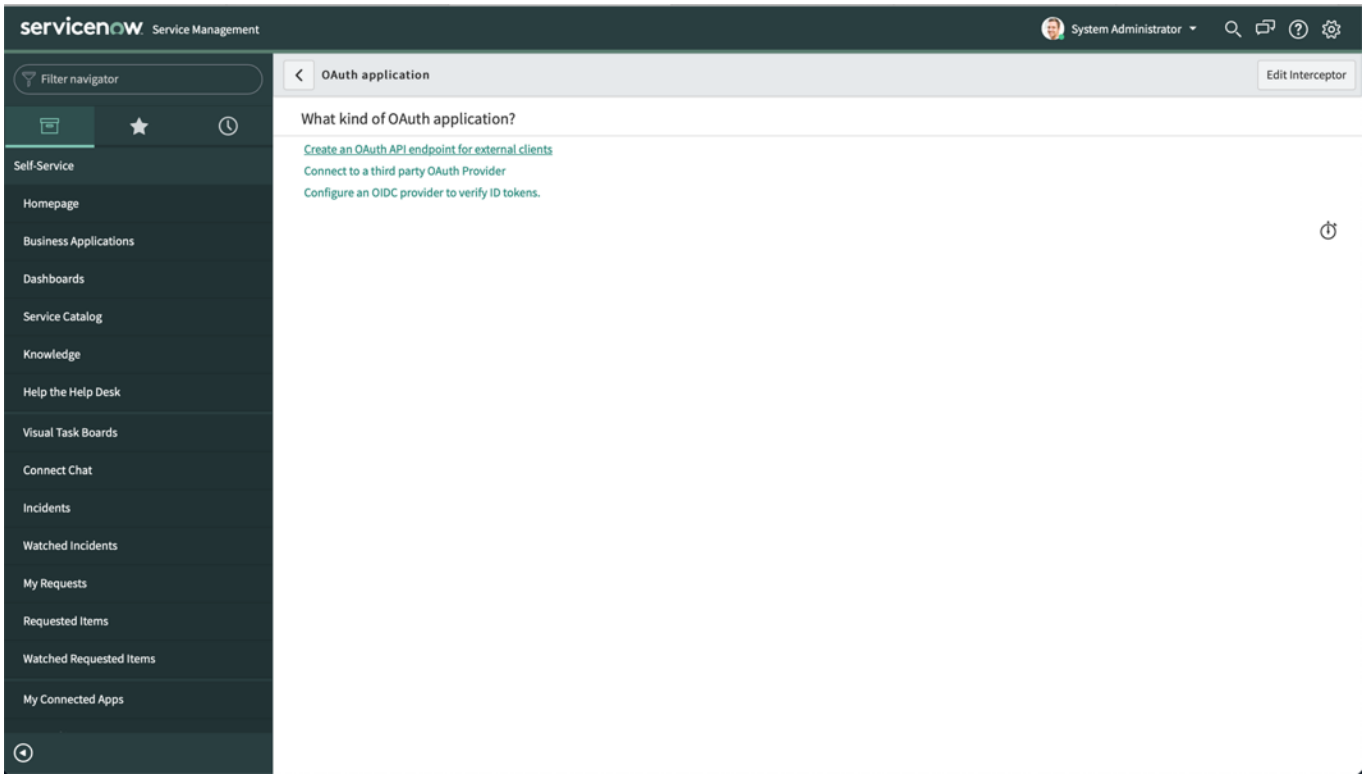


Search for *Registry* in Filter navigator search bar and then select *Application Registry*

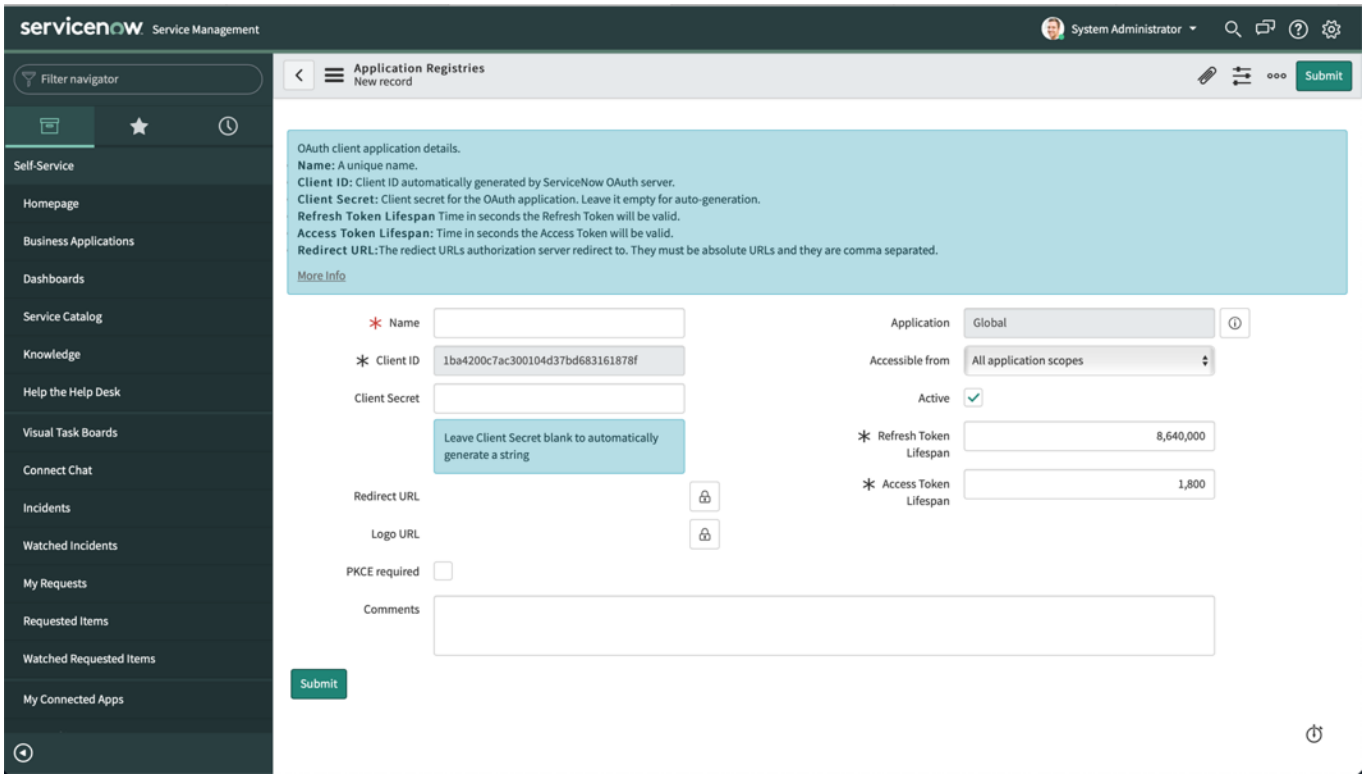


Create an OAuth API endpoint for external clients





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.

Connect to ServiceNow

*URL:

https://<your instance URL>/

ServiceNow instance URL e.g. https://<servicenow-id>.service-now.com/

*User Name:

<Username> used to log into the instance

ServiceNow user name

Note this can be different from one you use to log into Snow

*User Password:

<Password> used to log into the instance

ServiceNow user password

Note this can be different from one you use to log into Snow

*Client Id:

<Client Id> retrieved in above steps

ServiceNow client identifier

*Client Secret:

<Client Secret> retrieved in above steps

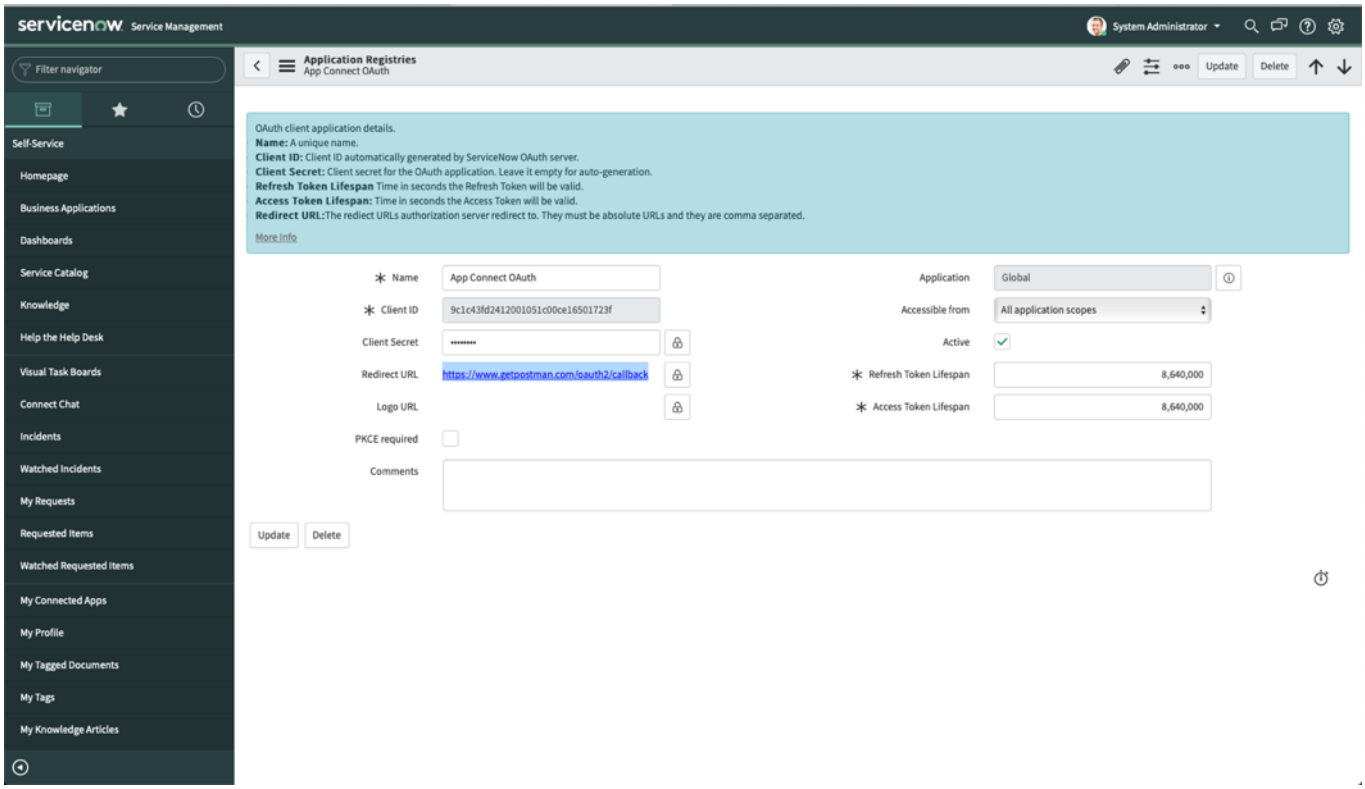
ServiceNow client secret

Connect

If you want use OAuth token-based authentication, then follow below steps to retrieve OAuth tokens.

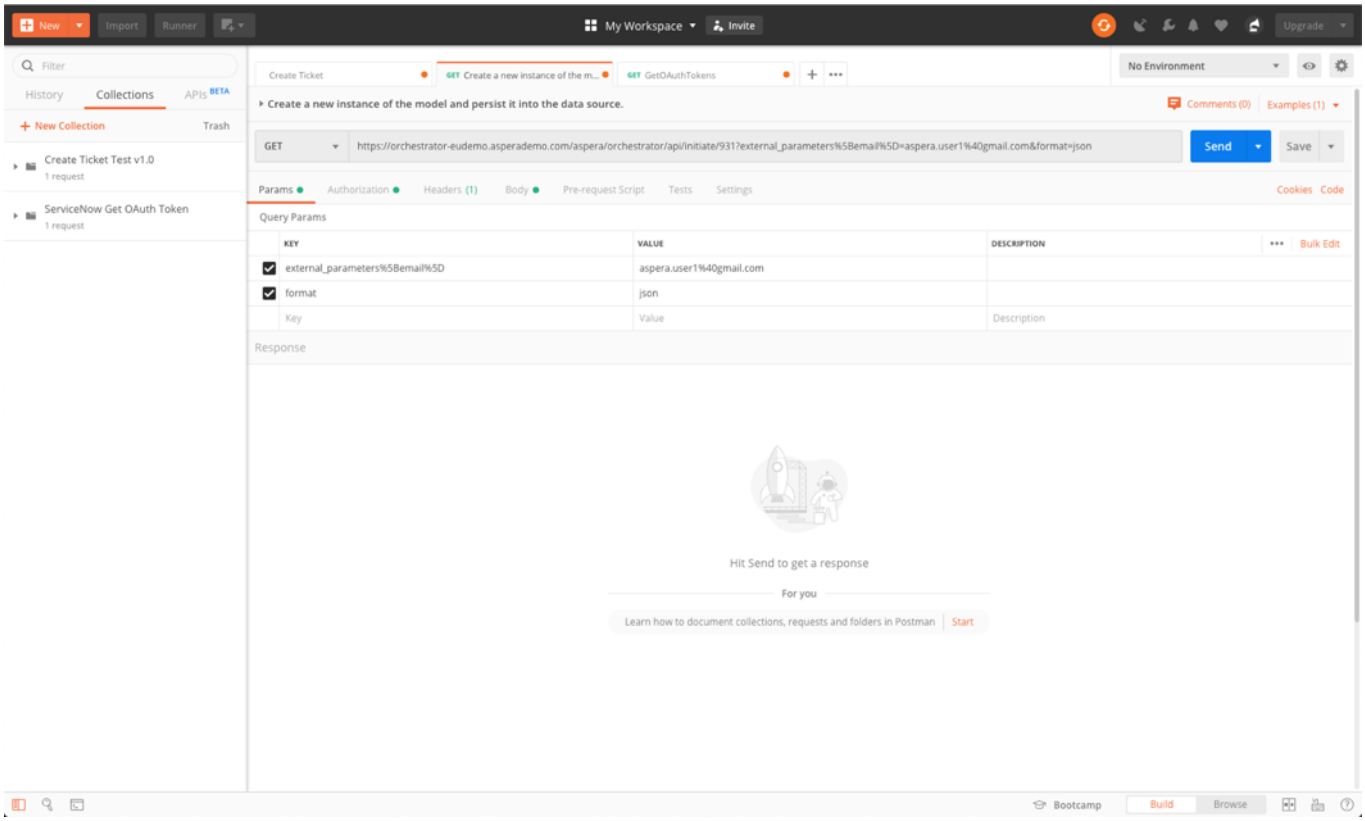
Inside your OAuth endpoint we now need to make few updates to make it work with Postman so that we can generate access tokens.

Insert Postman url: https://www.getpostman.com/oauth2/callback in Redirect URL field
Also, increase Access Token Lifespan to same as Refresh Token Lifespan

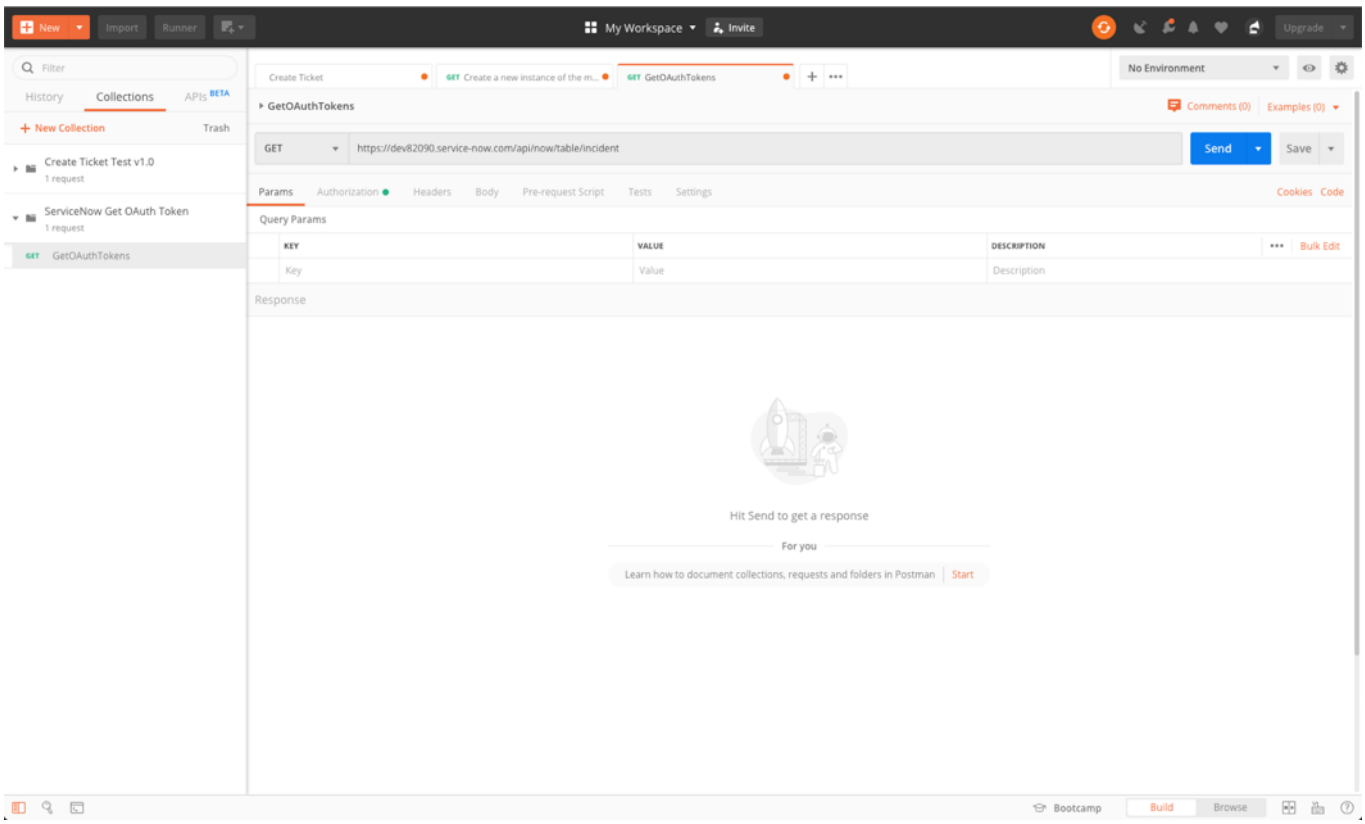


Click on Update button to save the changes.

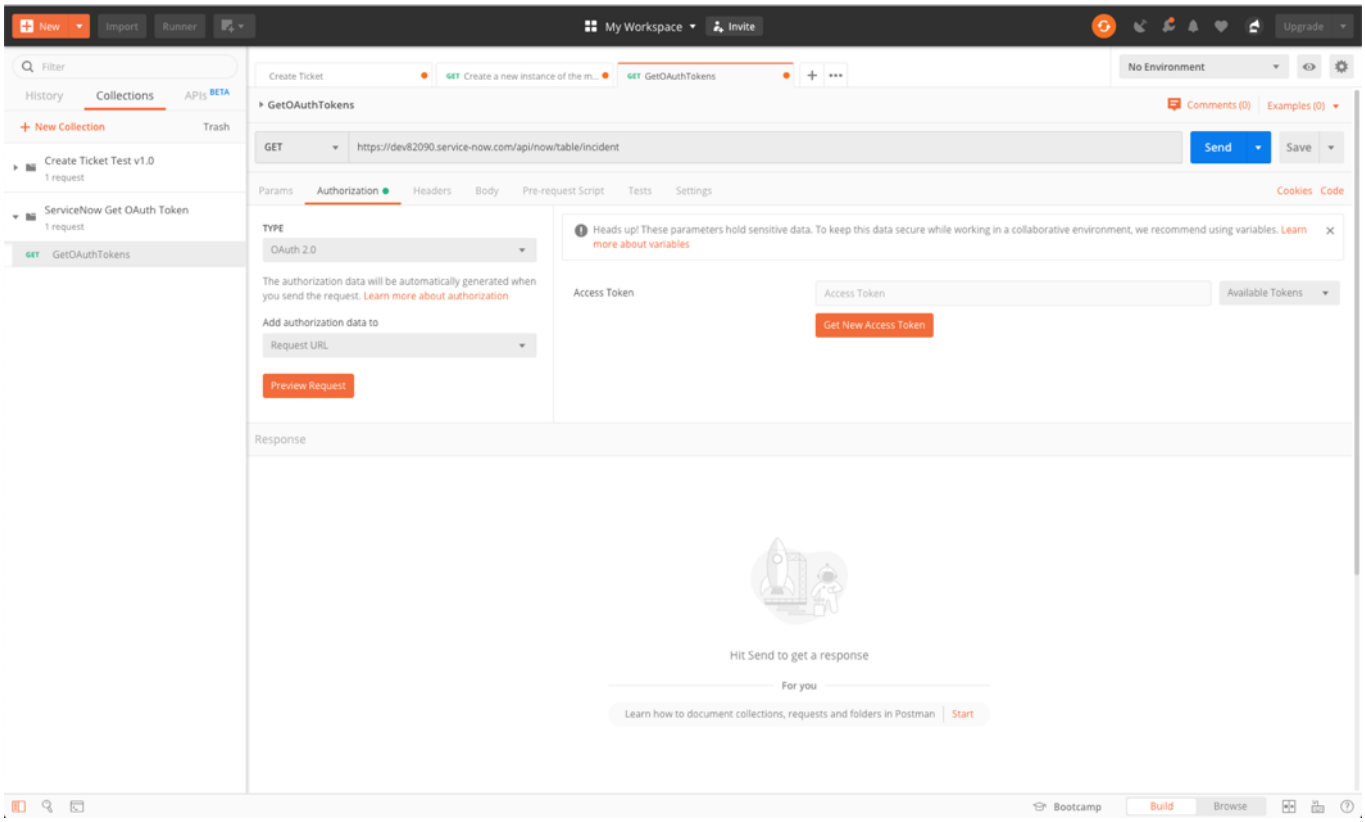
Now, open Postman app on your machine



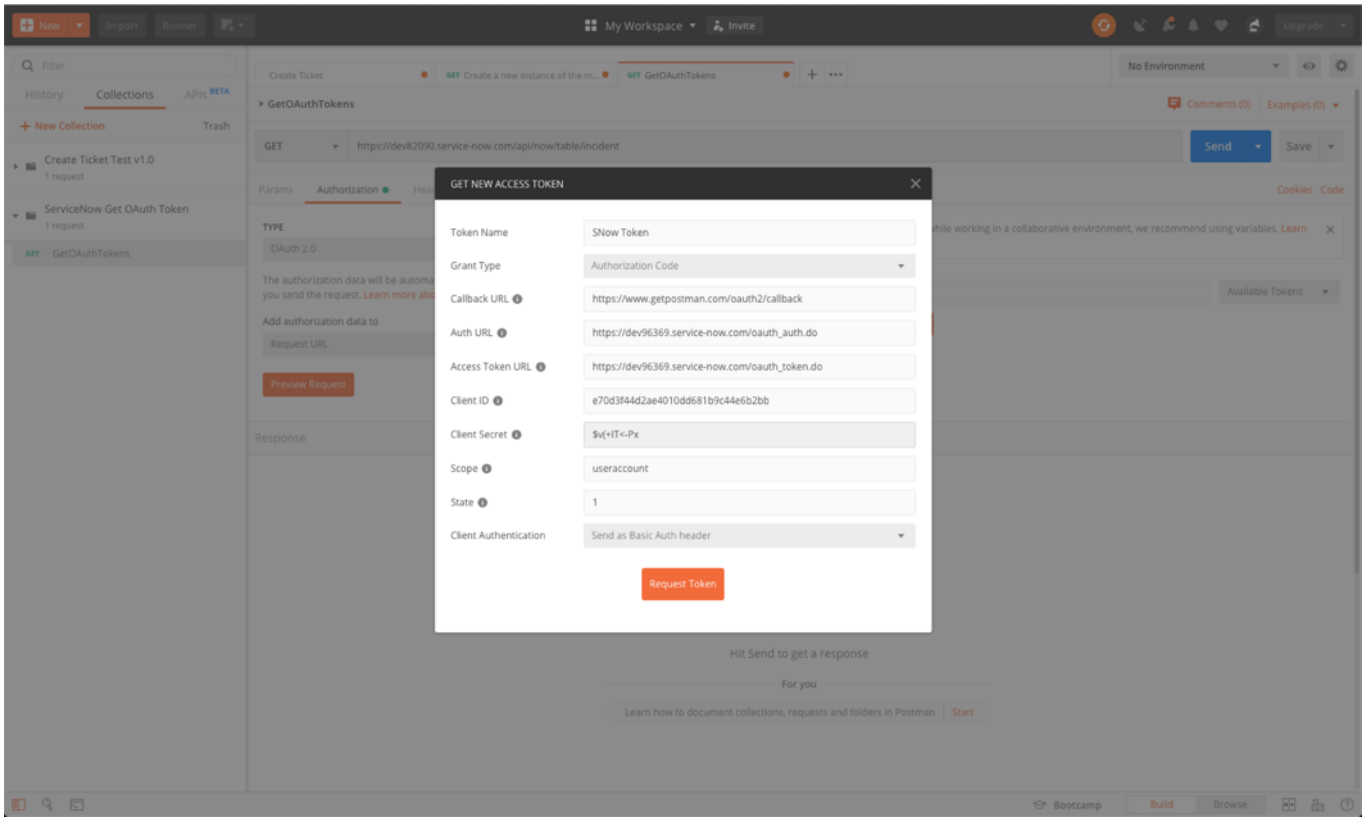
Create a new Collection with a new API request
Select HTTP request method as GET and use HTTP URL as below:
https://<your URL>/api/now/table/incident



Now *click on Authorization* to config auth parameters. Set *Auth type to OAuth 2.0* and click on *Get New Access Token button*.



Configure GET NEW ACCESS TOKEN panel with following details:
Token Name – any unique name
Grant Type – Authorization Code
Callabck URL – same as the one you inserted in OAuth endpoint in ServiceNow https://www.getpostman.com/oauth2/callback
Auth URL - https://<your ServiceNow URL>/oauth_auth.do
Access Token URL - https://<your ServiceNow URL>/ oauth_token.do
Client ID – As generated in above steps
Client Secret – As generated in above steps
Scope – useraccount
State – 1
Client Authentication – Send as Basic Auth header



Click on Request token and it will kick start OAuth web dance. You will require your admin username/pwd to authenticate yourself with web dance.

Once through click on Allow and it **will give you Access Token and Refresh Token** generated for you. Store them for future use