Pivotal Cloud Foundry

**Description:** **Cloud Foundry** is an open source [cloud computing](https://en.wikipedia.org/wiki/Cloud_computing) [platform as a service](https://en.wikipedia.org/wiki/Platform_as_a_service) (PaaS) originally developed by [VMware](https://en.wikipedia.org/wiki/VMware) and now owned by [Pivotal Software](https://en.wikipedia.org/wiki/Pivotal_Software) - a joint venture by [EMC](https://en.wikipedia.org/wiki/EMC_Corporation), [VMware](https://en.wikipedia.org/wiki/VMware) and [General Electric](https://en.wikipedia.org/wiki/General_Electric). This document helps developer to set up pivotal cloud foundry in eclipse and develop/deploy applications using spring cloud API to Pivotal Cloud Foundry(PCF).

**Contents:**

* **How to integrate Cloud Foundry plugin into Eclipse IDE**
* **How to Push/Deploy application to Pivotal Cloud Foundry**
* **How to Stop the Cloud Application**
* **How to Integrate services to cloud application**
* **How to create user defined service and bind service to application**
* **Deploying application with manifest.yml and its properties**
* **How to deploy two applications using manifest.yml**
* **Multiple manifests through inheritance**
* **Service Broker API**

**How to integrate Cloud Foundry plugin into Eclipse IDE**

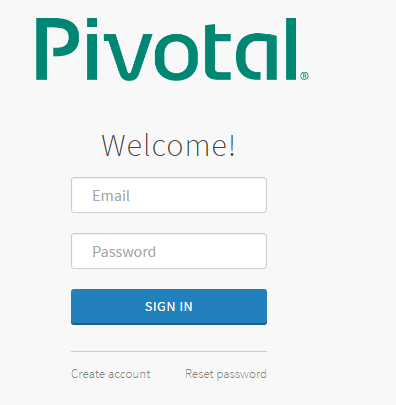
**Prerequisite:**

* Eclipse IDE (>= Kepler) and Spring Tool Suite
* Pivotal Web Services account: Go through step1 on how to set up free pivotal web services account.

**Steps to be followed:**

Step1: To set up pivotal web services account follow the below steps:

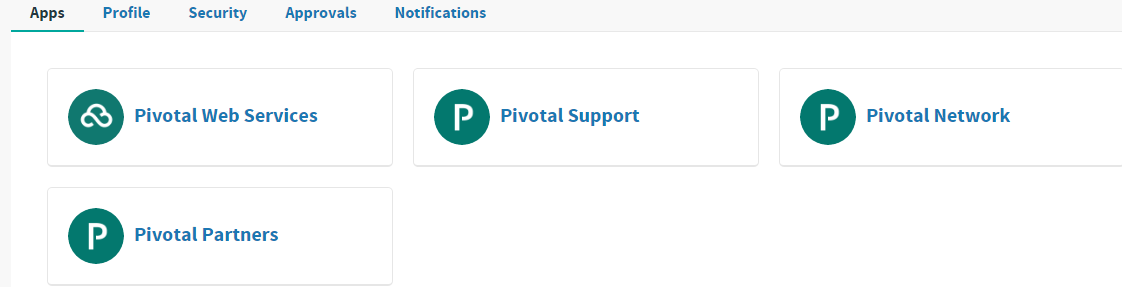
Click on : <https://login.run.pivotal.io/login>



Click on create account and provide the necessary details and click on sign up.

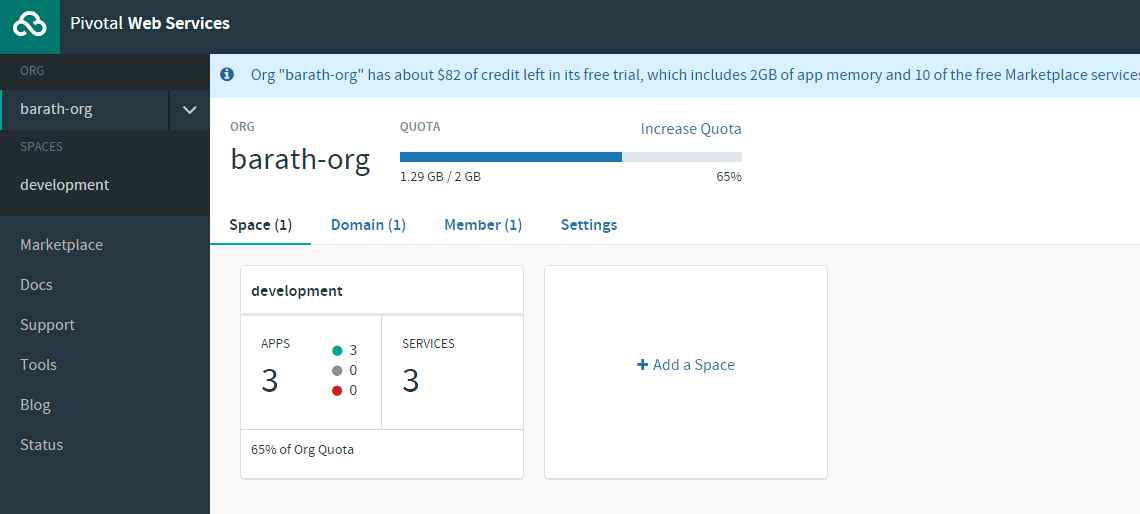
Once account is setup, login to your portal by providing your credentials

Once logged in,



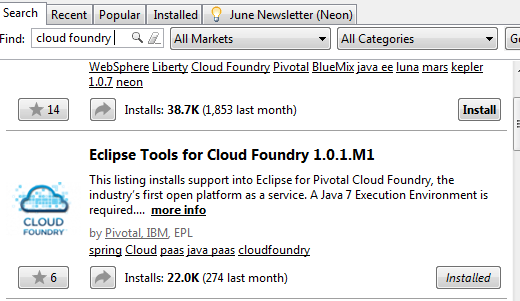
**Please note this is a free account created with free space of 2 GB and limited to free services provided by pivotal.**

Click on Pivotal web services to enter into your workspace like below:



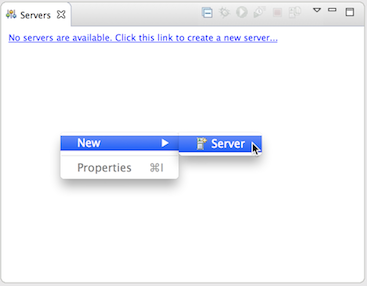
**Step 2:** Open Eclipse IDE, Go to Help🡪 Eclipse Market Place.

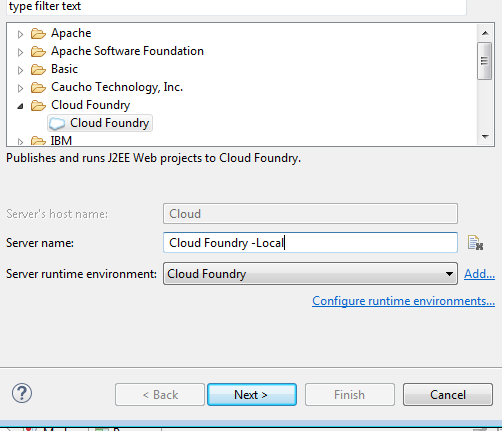
Search for Cloud foundry and click on install.



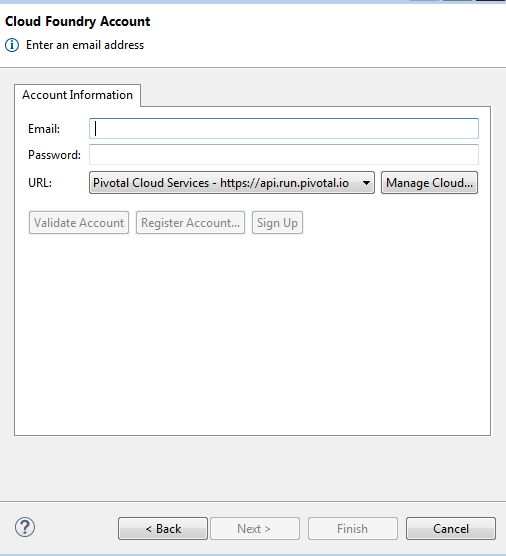
**Step 3:** Once installed, Go to Servers View🡪 New Server and then click on

Cloud Foundry like below:

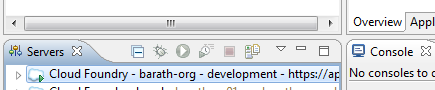




**Step 4:** Go to the next page and enter your username and password for your pivotal web services account. Note that you must have an account registered as a perquisite or else go to step 1 to get it registered.



**Step 5 :** Once set up is done, Cloud foundry server is up and running like below :



**How to Push/Deploy application to Pivotal Cloud Foundry**

**Step 1:** Create a simple spring boot application.

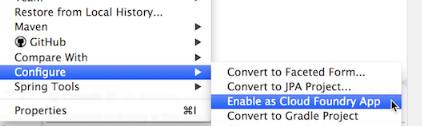
Please go through attached class file for basic Spring MVC application.

Attached class file :

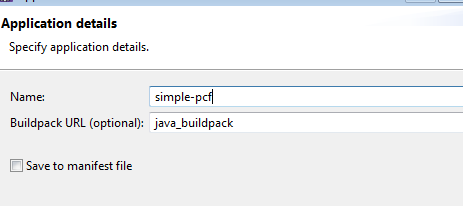
**Step 2:**

Two ways of deploying the application:

* Drag the project and drop into cloud foundry server
* Right click on the project and select configure🡪 Enable as Cloud foundry app



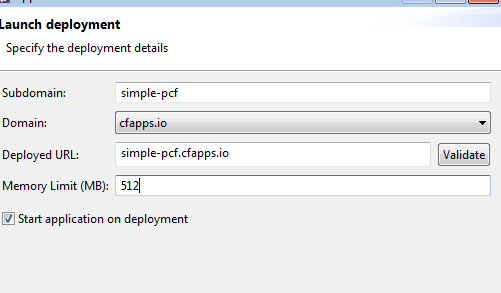
**Step 3** : Once dropped: enter the necessary information to get it deployed:



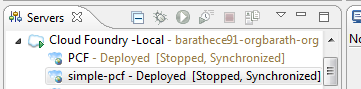
**Step 4**: Go to the next page and enter the necessary details and allocate the appropriate memory required.

Please note this is a free service of only 2 GB memory allocated so assign memory accordingly

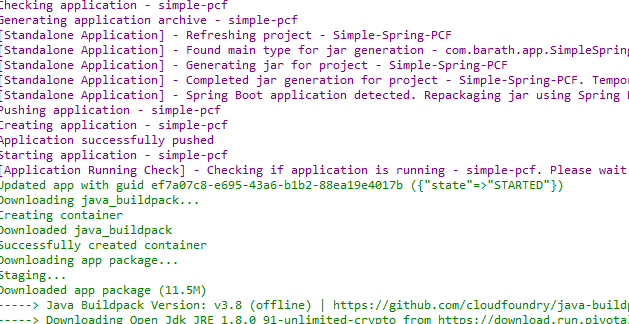
There is an option whether to deploy the application on startup or not.



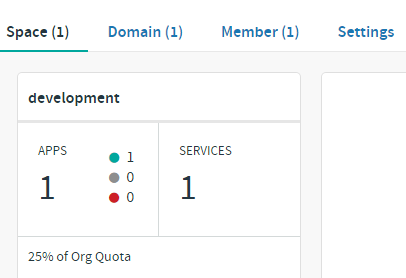
Finally deployed project will be displayed as below:



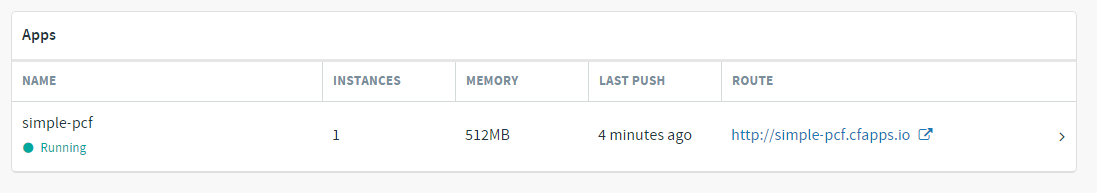
**Step 5:** Finally click on finish , Application will get pushed/deployed into cloud foundry and you can see the logs in console view as below :



**Step 6:** Now login to pivotal web services account to view the application.

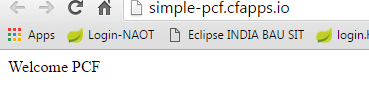


Click on the Apps to view the application:



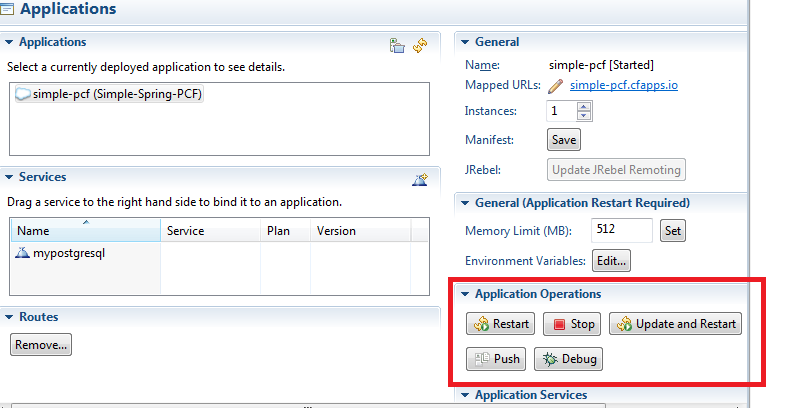
you can see the name of the application as well the route to access the application:

Finally to verify the application, click on the route to view the welcome page



How to stop the cloud application

**Step 1**: Double click on the project deployed and application page will open as below:



**Step 2**: There are various options to choose from:

**Stop**: To stop the application

**Push**: To push the changes into application

**Update and Restart :** To push and restart the application

**Restart** : To restart the application

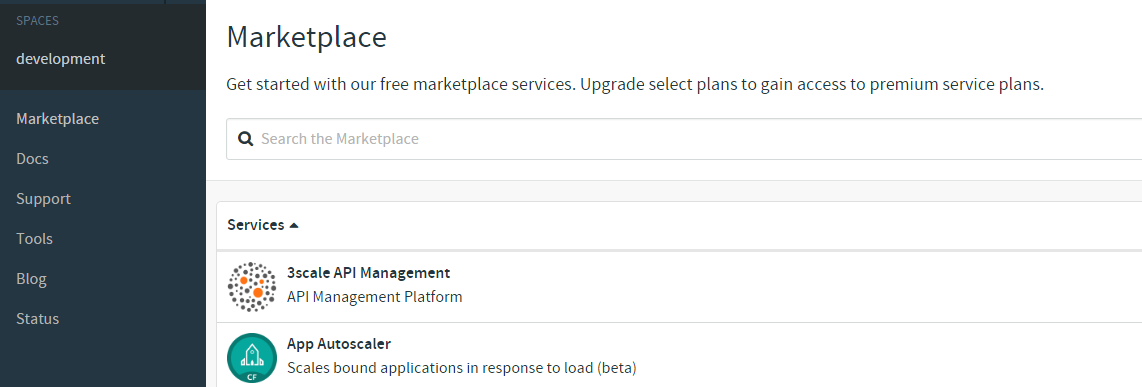
**How to Integrate services to cloud application**

**Description:** Pivotal provides 10 free services to bind it to the application.

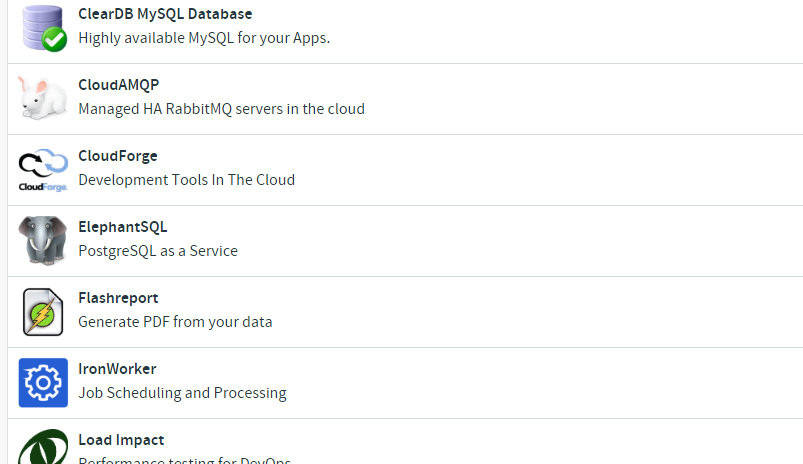
In this example, we will see how to bind Elephant-SQL(Postgre as a service) to the application

**Step 1:** Login to Pivotal web services account :

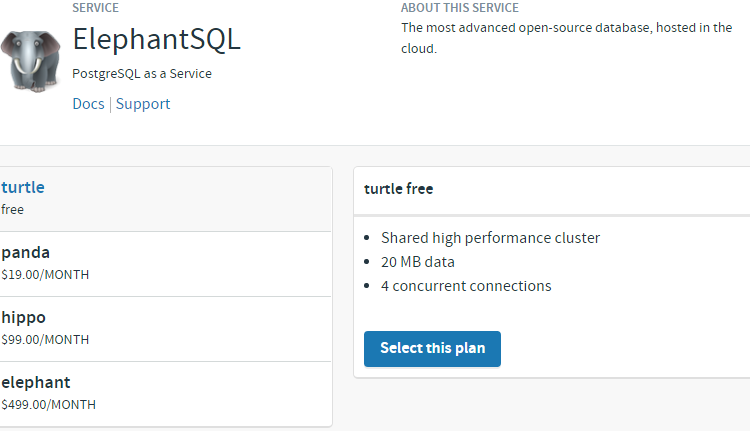
**Step 2**: Click on Market place :



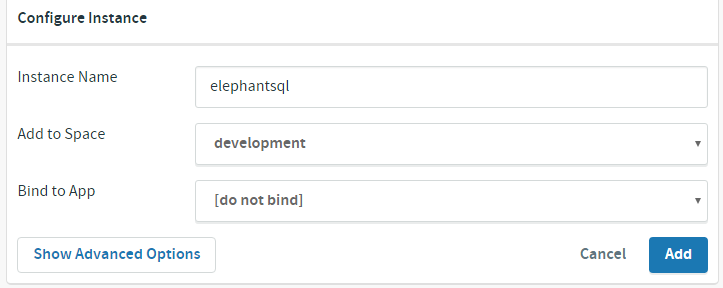
**Step 3:** Click on Elepant SQL :



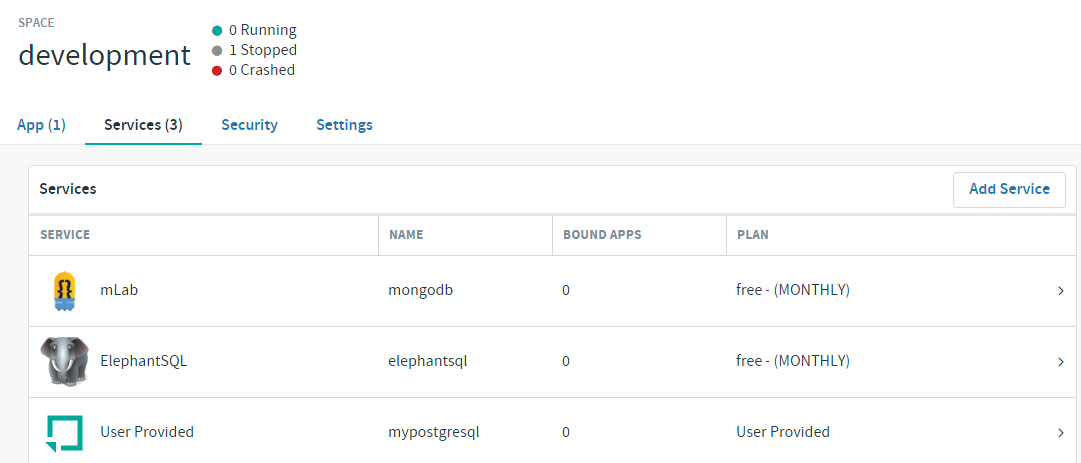
Step 4: Click on Select this plan (FREE PLAN)



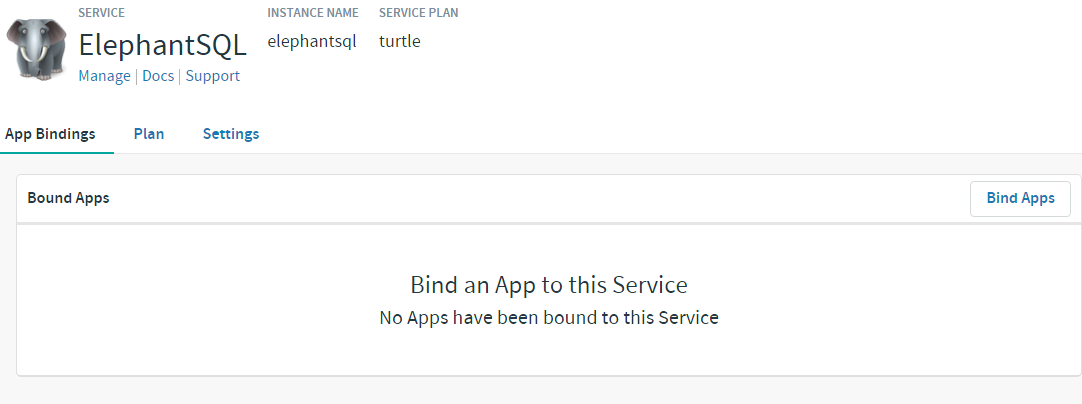
**Step 5:** provide the instance name as elephantsql and then click on add:



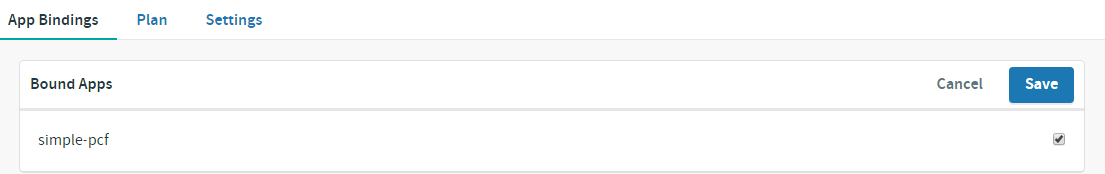
**Step 6:** Once added, Go to organization space and click on services to view the services registered:



**Step 7:** Click on ElephantSQL service:



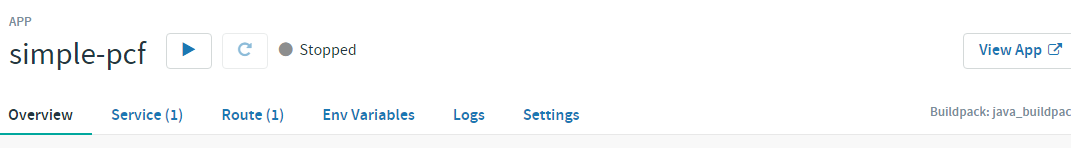
**Step 8:** Click Bind Apps tab and select the application to bind the service.



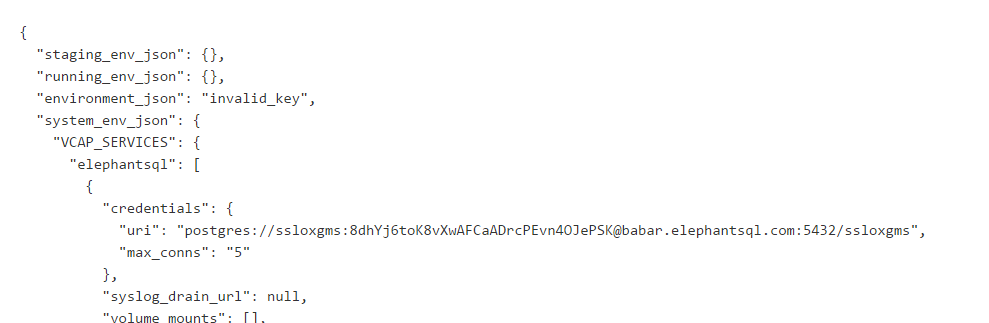
**Step 9**: Click on Save to bind the selected service.

**Step 10:** Now click on the application :





**Step 11:** Go to ENV Variables panel:



Now you can see elephantsql is binded to the application with credentials uri and max\_conns

**Step 12:** Now let us create DataSource to make a connection to the DB in application.

Note: There are various ways of creating datasource in spring application.

**Case 1: Using application.properties:**

Spring boot provides following properties to create a datasource object

**spring.datasource.url=**

**spring.datasource.username=**

**spring.datasource.password=**

**spring.datasource.driver-class-name=**

Now we know the credentials Uri to connect to ElephantSql,

**Example:**

postgres://ssloxgms:8dhYj6toK8vXwAFCaADrcPEvn4OJePSK@babar.elephantsql.com:5432/ssloxgms

**Corresponding entries in application.properties:**

**spring.datasource.url=** postgres://babar.elephantsql.com:5432/ssloxgms

**spring.datasource.username=**ssloxgms

**spring.datasource.password=**8dhYj6toK8vXwAFCaADrcPEvn4OJePSK

**spring.datasource.driver-class-name=org.postgresql.Driver**

**Case 2 : Using cloud`s environment variables:**

**spring.datasource.url=** ${VCAP\_SERVICES.service-name.url}

**spring.datasource.username==** ${VCAP\_SERVICES.service-name.username}

**spring.datasource.password==** ${VCAP\_SERVICES.service-name.password}

**spring.datasource.driver-class-name=org.postgresql.Driver**

**Case 3: Using Service Connector API**

**Example: Let class extend AbstractCloudConfig**

Make use of connectionFactory().datasource(SERVICE\_INSTANCE\_ID/NAME) to create datasource

@Bean

**public** DataSource dataSource() {

DataSource dataSource=connectionFactory().dataSource("elephantsql");

**return** dataSource;

}

**How to create user defined service and bind service to application**

**Description:** User-provided service instances enable developers to use services that are not available in the marketplace with their applications running on Cloud Foundry. Please read below reference for more details.

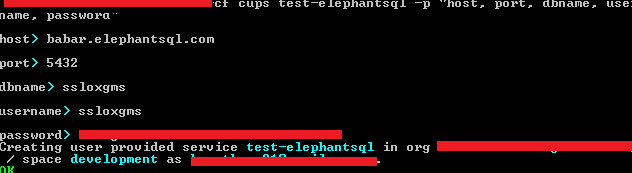
**Reference:** <https://docs.cloudfoundry.org/devguide/services/user-provided.html>

**Step1:** We will create a user defined service using PCF CLI application:

Login into Pivotal: cf login –a [API END POINT] –u [USER NAME] –p [password]

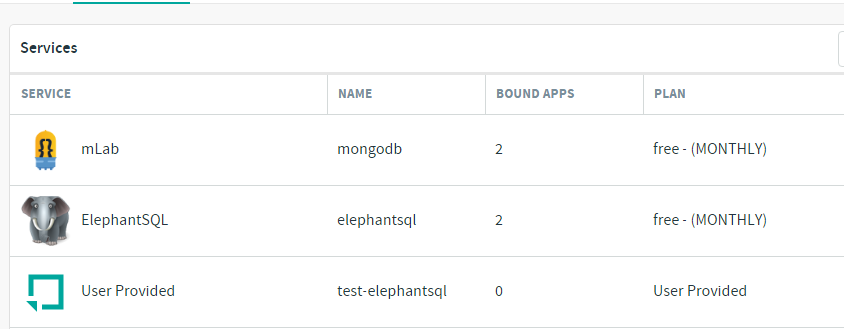
**Step 2:** Create user defined service :

cf cups SERVICE\_INSTANCE -p "host, port, dbname, username, password"



**Step 3:** Once successfully created, go to org space and select services :

You will see user-provided service is created like below:



**Step 4:** Bind the created service into one of the existing applications in PCF:

Once binded to the application, go to environment variables of the application:

You will see the details like below:



**Step 4:** Now make use of the created service in the application code using one of the cases discussed above

**Deploying application with manifest.yml and its properties**

Example Manifest

You can deploy applications without even using a manifest. The benefits manifests may provide include consistency and reproducibility. When you want applications to be portable between different clouds, manifests may prove especially useful.

Manifests are written in YAML. The manifest below illustrates some YAML conventions, as follows:

* The manifest may begin with three dashes.
* The applications block begins with a heading followed by a colon.
* The application name is preceded by a single dash and one space.
* Subsequent lines in the block are indented two spaces to align with name.

---

applications:

- name: simple-pcf

memory: 512M

host: nifty

A minimal manifest requires only an application name. To create a valid minimal manifest, remove the memory and host properties from this example.

## Always Provide an Application Name to cf push

cf push requires an application name, which you provide either in a manifest or at the command line.

 cf push locates the manifest.yml in the current working directory by default, or in the path provided by the -foption.

If you do not use a manifest, the minimal push command looks like this:

$ cf push my-app

**Configurable Attributes/properties in maniyest.yml:**

**domain attribute:** Every Cloud Foundry instance may have a shared domain set by an admin. Unless you specify a domain, Cloud Foundry incorporates that shared domain in the route to your application.

**host attribute:** Use the host attribute to provide a hostname, or subdomain, in the form of a string. This segment of a route helps to ensure that the route is unique. If you do not provide a hostname, the URL for the app takes the form of APP-NAME.DOMAIN.

**hosts attribute:** Use the hosts attribute to provide multiple hostnames, or subdomains. Each hostname generates a unique route for the app. hosts can be used in conjunction with host. If you define both attributes, Cloud Foundry creates routes for hostnames defined in both hostand hosts.

**no-hostname attribute:** By default, if you do not provide a hostname, the URL for the app takes the form of APP-NAME.DOMAIN. If you want to override this and map the root domain to this app then you can set no-hostname as true.

**memory attribute:** Use the memory attribute to specify the memory limit for all instances of an app.

**instances attribute:** Use the instances attribute to specify the number of app instances that you want to start

**timeout attribute:** The timeout attribute defines the number of seconds that Cloud Foundry allocates for starting your application.

**Services:** Applications can bind to services such as databases, messaging, and key-value stores.

**Please go through below link for more details on attributes and its usage:**

[**https://docs.cloudfoundry.org/devguide/deploy-apps/manifest.html**](https://docs.cloudfoundry.org/devguide/deploy-apps/manifest.html)

**How to deploy two applications using manifest.yml**

Consider we have two application named sports and movies

Creata a simple manifest.yml like below: (Note only application name is enough to deploy the apps)

**---**

**applications:**

**- name: sports**

**Path: <PROVIDE PATH TO APPLICATION WAR FILE>/sports.war**

**- name: movies**

**Path: <PROVIDE PATH TO APPLICATION WAR FILE>/movies.war**

Please note that we can bind other attritubes to each application as required.

## Multiple Manifests with Inheritance

A single manifest can describe multiple applications. Another powerful technique is to create multiple manifests with inheritance. Here, manifests have parent-child relationships such that children inherit descriptions from a parent. Children can use inherited descriptions as-is, extend them, or override them.

Content in the child manifest overrides content in the parent manifest, if the two conflict.

This technique helps in these and other scenarios:

* An application has a set of different deployment modes, such as debug, local, and public. Each deployment mode is described in child manifests that extend the settings in a base parent manifest.
* An application is packaged with a basic configuration described by a parent manifest. Users can extend the basic configuration by creating child manifests that add new properties or override those in the parent manifest.

The benefits of multiple manifests with inheritance are similar to those of minimizing duplicated content within single manifests. With inheritance, though, we “promote” content by placing it in the parent manifest.

Every child manifest must contain an “inherit” line that points to the parent manifest. Place the inherit line immediately after the three dashes at the top of the child manifest. For example, every child of a parent manifest called base-manifest.yml begins like this:

---

...

inherit: base-manifest.yml

You do not need to add anything to the parent manifest.

In the simple example below, a parent manifest gives each application minimal resources, while a production child manifest scales them up.

**simple-base-manifest.yml**

---

path: .

domain: shared-domain.example.com

memory: 256M

instances: 1

services:

- singular-backend

# app-specific configuration

applications:

- name: springtock

host: 765shower

path: ./april/build/libs/april-weather.war

- name: wintertick

host: 321flurry

path: ./december/target/december-weather.war

**simple-prod-manifest.yml**

---

inherit: simple-base-manifest.yml

applications:

- name:springstorm

memory: 512M

instances: 1

host: 765deluge

path: ./april/build/libs/april-weather.war

- name: winterblast

memory: 1G

instances: 2

host: 321blizzard

path: ./december/target/december-weather.war

**Note**: Inheritance can add an additional level of complexity to manifest creation and maintenance. Comments that precisely explain how the child manifest extends or overrides the descriptions in the parent manifest can alleviate this complexity.

**Service Broker API**

**<<YET TO BE DOCUMENTED>>**