



## Postgres deployment

This time, we will not use the command line but the OpenShift web console to deploy an instance of Postgres. Start by opening the Web interface:

Red Hat OpenShift Container Platform

Project: All Projects

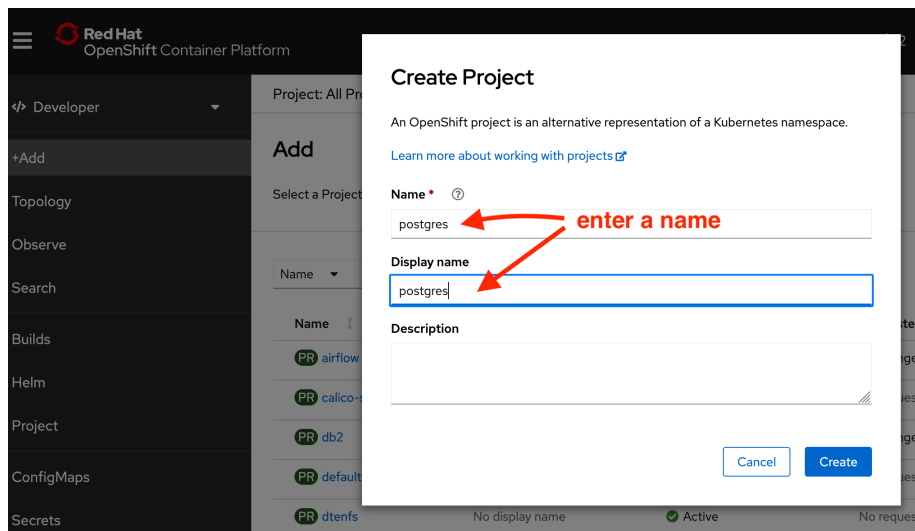
**Add**

Select a Project to start adding to it or [create a Project](#).

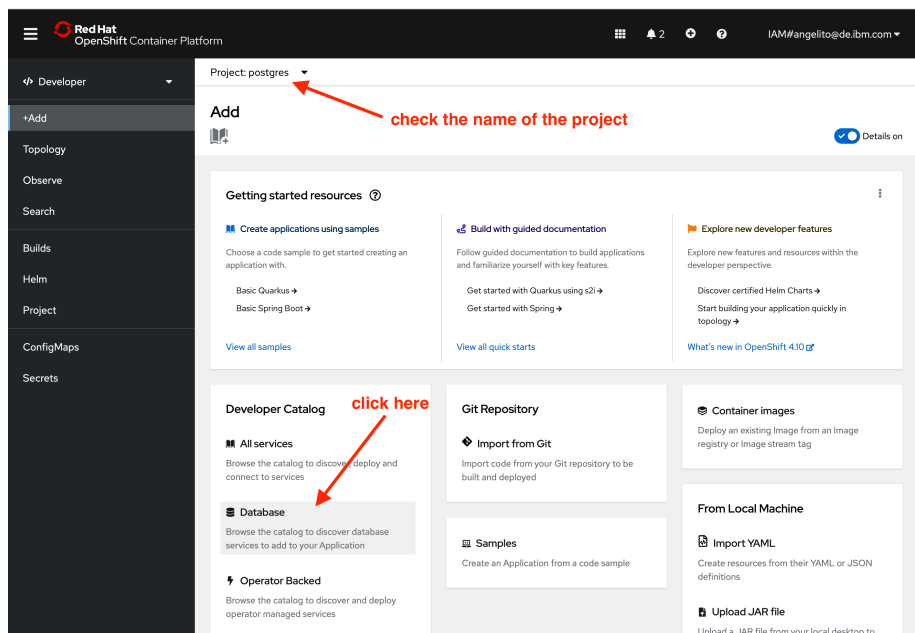
**click here**

Name	Display name	Status
PR airflow	No display name	Active
PR calico-system	No display name	Active

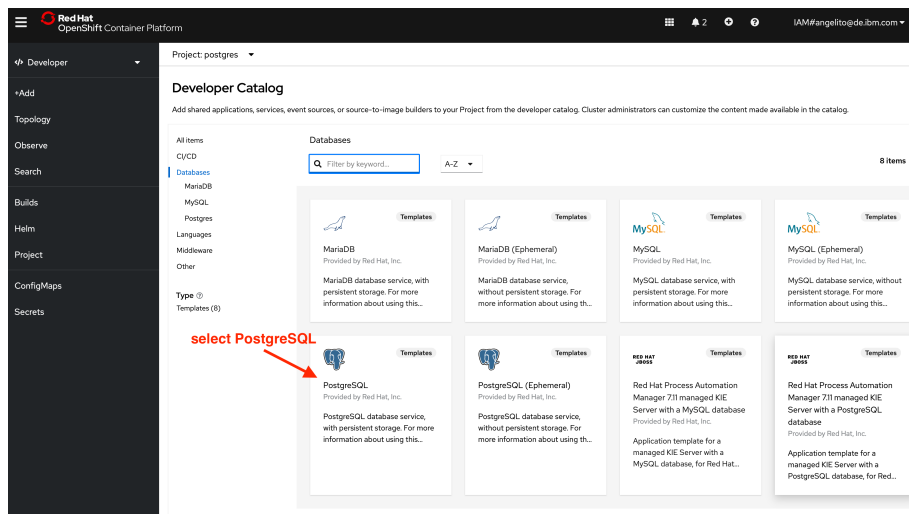
Now, create a new project:



For the next step, ensure that you are in the newly created project and add a database deployment template:



Select the PostgreSQL template:



**PostgreSQL**  
Provided by Red Hat, Inc.



[Instantiate Template](#)

← **click here**

#### Provider

Red Hat, Inc.

#### Created at

🕒 Mar 7, 2023, 12:00 PM

#### Support

[Get support](#)

#### Documentation

[Refer documentation](#)

#### Description

PostgreSQL database service, with persistent storage. For more information about using this template, including OpenShift considerations, see <https://github.com/sclorg/postgresql-container/>.

NOTE: Scaling to more than one replica is not supported. You must have persistent volumes available in your cluster to use this template.

In the next panel, you just need to enter a user and the password. Leave the rest fields as default:

**Red Hat OpenShift Container Platform**

**Developer**

- +Add
- Topology
- Observe
- Search
- Builds
- Helm
- Project
- ConfigMaps
- Secrets

### Instantiate Template

**Namespace \***  
 postgres

**Memory Limit \***  
 512Mi  
 Maximum amount of memory the container can use.

**Database Service Name \***  
 postgresql  
 The name of the OpenShift Service exposed for the database.

**PostgreSQL Connection Username \***  
 postgres  
 Username for PostgreSQL user that will be used for accessing the database.

**PostgreSQL Connection Password \***  
 postgres  
 Password for the PostgreSQL connection user.

**PostgreSQL Database Name \***  
 sampledb  
 Name of the PostgreSQL database accessed.

**Volume Capacity \***  
 1Gi  
 Volume space available for data, e.g. 512Mi, 2Gi.

**Version of PostgreSQL Image \***  
 10-e18  
 Version of PostgreSQL image to be used (10-e17, 10-e18, or latest).

**Create** **Cancel**

**PostgreSQL**  
 DATABASE: POSTGRESQL  
 • [View documentation](#)  
 • [Get support](#)  
 PostgreSQL database service, with per considerations, see <https://github.com>  
 NOTE: Scaling to more than one replica template.  
 The following resources will be created:  
 • DeploymentConfig  
 • PersistentVolumeClaim  
 • Secret  
 • Service

After a few minutes, you can check if the deployment succeeded by inspecting the pods

```
# Replace the command with your own one inside the single quotes and run the cell
# Example OC_LOGIN_COMMAND='oc login --token=sha256~3bR5KXgwiUoaQiph2_kIXCDQnVfm_HQy3YwU2m-t
OC_LOGIN_COMMAND='_replace_this_string_by_pasting_the_clipboard_'
$OC_LOGIN_COMMAND
```

```
# Check if there is one pod running and another one completed
oc project postgres
oc get pods
```

Check if there is one service defined for postgres

```
# Look for a service. The default name will be postgresql and the port 5432
oc get services
```

In case you want to access the Postgres database with a GUI like DBeaver, SqiereL SQL or an external python program, it is necessary to externalize the

service port:

```
servicename=$(oc get services | grep postgresql | awk '{print $1}')
oc expose svc $servicename --type=NodePort --generator="service/v2" --name=pg-nodeport
```

If everything went well, you will see two services. The port 5432 is the default postgres and it is accessible inside the cluster. The NodePort entry will map 5432 to another one (the number attached right to the colon) and can be used by applications hosted outside OpenShift.

*# Two services will be displayed:*

```
oc expose svc pg-nodeport
oc get services
```

And the address of the host can be retrieved by querying the routes:

*# This is the host name of Postgres to access it from outside the cluster*

```
oc get routes
```

Finally, you can verify if you can connect to the default database and insert data:

```
podname=$(oc get pods --field-selector=status.phase=Running -o custom-columns=POD:.metadata.name)
oc rsh $podname psql --version
oc rsh $podname psql -c '\l'
oc rsh $podname psql -c 'create table test(c1 integer, c2 char(1));'
oc rsh $podname psql -c "insert into test values(1,'a')"
oc rsh $podname psql -c 'select * from test'
```

---

Previous Section: DataStage deployment

Return to main