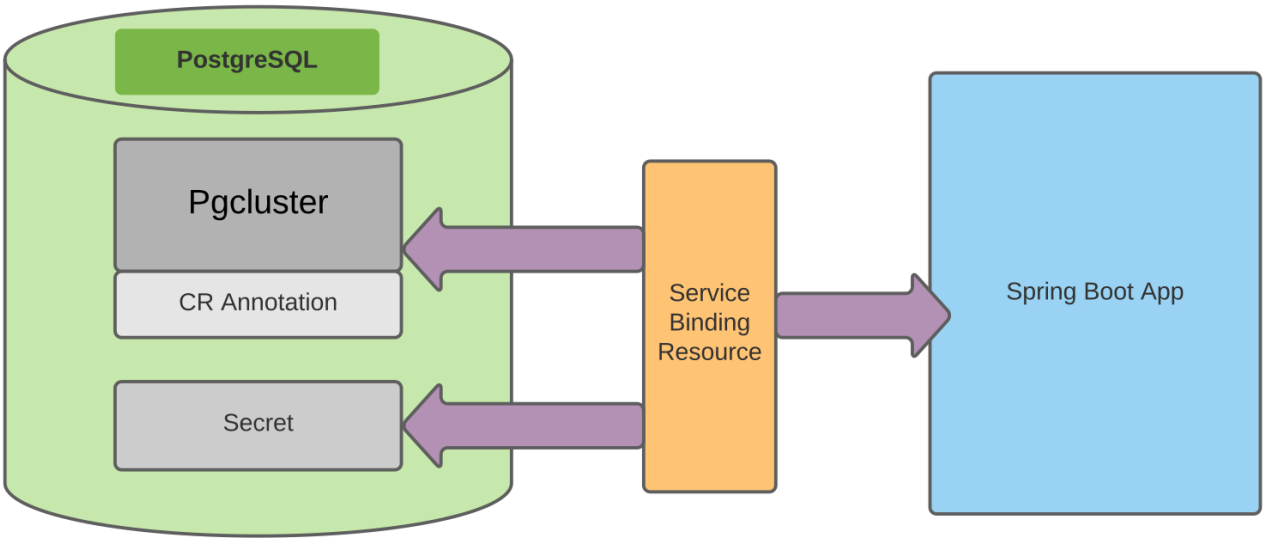
Quick start

In this quick start, you will see a sample application that you can deploy and use to play around. As part of this exercise, a backing service and an application is required. The backing service is a PostgreSQL database and application is a Spring Boot REST API server.

Here is a schematic diagram of this setup:



The service binding operator collects backing service configuration required for the connectivity and expose it to the sample application.

This topic provides the following information on how to deploy application and connect it to a a backing service:

1. Prerequisites
2. Creating a database instance
3. Deploying an application
4. Connecting the application to a backing service

Prerequisites

The following components must be installed and configured:

- Kubernetes cluster (**Note:** You can use minikube or kind, locally)
- kubectl
- Service Binding Operator.

Creating a database instance

The application is going to connect to a PostgreSQL database backend. The PostgreSQL can be setup using the Crunchy PostgreSQL operator from OperatorHub.io.

The installation of the operator doesn't create a database instance for connection.

1. To create a database instance, you need to create custom resource (`Pgcluster`) and that will trigger the operator reconciliation. For convenience, run this command to create `Pgcluster` custom resource:

```
kubectl apply -f
https://gist.githubusercontent.com/baijum/b99cd8e542868a00b2b5efc2e1b7dc10
```

2. Ensure all the pods in `my-postgresql` is running (it will take few minutes):

```
kubectl get pod -n my-postgresql
```

You should see output something like this:

```
NAME                                      READY   STATUS    RESTARTS
AGE
backrest-backup-hippo-9gtqf               1/1     Running   0
13s
hippo-597dd64d66-4ztww                    1/1     Running   0
3m33s
hippo-backrest-shared-repo-66ddc6cf77-sjgqp   1/1     Running   0
4m27s
```

The database will be empty by default, it requires the schema and sample data to work with the application.

3. You can initialize the database with the schema and sample data using this command:

```
bash <(curl -s
https://gist.githubusercontent.com/baijum/b99cd8e542868a00b2b5efc2e1b7dc10
database.sh)>
```

Now the database is ready to connect from application. The next section explains how to configure application:

Deploying an application

1. Deploy the `spring-petclinic-rest` app with this `Deployment` configuration:

```
kubectl apply -f
https://gist.githubusercontent.com/baijum/b99cd8e542868a00b2b5efc2e1b7dc10
deployment.yaml
```

2. Port forward the application port and try to access it from your local system

```
kubectl port-forward --address 0.0.0.0 svc/spring-petclinic-rest 9966:80
-n my-postgresql
```

3. open http://localhost:9966/petclinic

You should see a Swagger UI where you can play with the API.

Since the binding is not present in the application, you cannot see be any values in results.

If you try to access list of all pets, you can see an error like this:

```
curl -X GET "http://localhost:9966/petclinic/api/pets" -H "accept: applica

{"className":"org.springframework.transaction.CannotCreateTransactionExcep
```

```
not open JPA EntityManager for transaction; nested exception is
org.hibernate.exception.JDBCConnectionException: Unable to acquire JDBC
Connection"}
```

In the next section, you will see how to fix it.

Connecting the application to a backing service

The application was not working as the bindings were not present in the app.

1. Create the ServiceBinding custom resource to inject the bindings:

```
apiVersion: binding.operators.coreos.com/v1alpha1
kind: ServiceBinding
metadata:
    name: spring-petclinic-rest
    namespace: my-postgresql
spec:
    services:
    - group: "crunchydata.com"
      version: v1
      kind: Pgcluster
      name: hippo
    - group: ""
      version: v1
      kind: Secret
      name: hippo-hippo-secret
    application:
      name: spring-petclinic-rest
      group: apps
      version: v1
      resource: deployments
    mappings:
    - name: type
      value: "postgresql"
```

For the convenience, the above resource can be installed like this:

```
kubectl apply -f
https://gist.githubusercontent.com/baijum/b99cd8e542868a00b2b5efc2e1b7dc10
binding.yaml
```

The next section explains the ServiceBinding configuration.

2. Port forward the application port and access it from your local system

```
kubectl port-forward --address 0.0.0.0 svc/spring-petclinic-rest 9966:80
-n my-postgresql
```

3. Open http://localhost:9966/petclinic

You should see a Swagger UI where you can play with the API.

If you try to access list of all pets, you can see the result like this:

```
$ curl -X GET "http://localhost:9966/petclinic/api/pets" -H "accept:
application/json"
[{"id":1,"name":"Leo","birthDate":"2000/09/07","type":
{"id":1,"name":"cat"},
"owner":
{"id":1,"firstName":"George","lastName":"Franklin","address":"110...
```

Summary

The Service Binding operator collects the backing service configuration required for connectivity and exposes it to the applications. Suppose the Service Binding operator is not present. In that case, the application's admin needs to extract all the configuration details and create a Secret resource and expose it to the application through volume mount in Kubernetes.