Deploy an API and a database in Kubernetes

This documentation will walk you through the process of deploying a simple Flask-based API alongside a recreation of a particular state of a database named ‘data_mart_airbnb’ within Kubernetes cluster.

Prerequisites:

1. Kubernetes up and running
2. kubectl command-line tool installed
3. Basic understanding of Kubernetes concept.

Step 1: Create Kubernetes Deployment for the API

1. Create API Docker image: Navigate to the directory api and build a Docker image of your API by using:

```
docker build -t giprocida/myflask:1.0 .
```

This command will create a Docker image on your local machine that will be stored in your local Docker image registry.

2. API deployment YAML: Create a deployment yaml file for the API deployment. You can use imperative commands to create it :

```
kubectl create deploy api-deploy --image=giprocida/myflask:1.0 --port=5000
--namespace=app-space --dry-run=client -o yaml > api-deploy.yaml
```

Here is a breakdown of what each part of the command does:

- `kubectl create deploy api-deploy --image=giprocida/myflask:1.0 --port=5000`: This part of the command creates a deployment named 'api-deploy' with the specified Docker image 'giprocida/myflask:1.0'.
- `--port=5000`: It instructs Kubernetes to expose port 5000 within the container.
- `--namespace=app-space`: This flag specifies the namespace in which the deployment should be created (app-space).
- `-dry-run=client -o yaml`: This flag specifies that the command should perform a dry run (simulating the execution of the command without actually applying changes to the cluster). The '-o yaml' indicates that the output should be in yaml format.
- `> api-deploy.yaml`: This part of the command redirects the output of the dry run to a file named "api-deploy.yaml".

To enable querying the database from the Flask-based API, it's advisable to store configuration details within a separate Kubernetes resource named ConfigMap. You can use imperative commands to create it:

```
kubectl create configmap my-sql-config --from-env-file=myvar --namespace=app-space --dry-run=client -o yaml > my-sql-
config.yaml
```

- `kubectl create configmap my-sql-config`: This part of the command initiates the creation of a ConfigMap named "my-sql-config.
- `--from-env-file=myvar`: This flag specifies that the data for the ConfigMap should be loaded from an environment file named "myvar". The values from this file will be used to create key-value pairs in the ConfigMap.

myvar
24 Aug 2023, 09:33 am

Each line of the file “myvar” corresponds to a specific environment variable related to a MySQL database configuration.

- mysql-deploy is the value of the `MYSQL_HOST` environment variable, which represents the hostname of the MySQL database server (which will be created later on).
- root is the value of the `MYSQL_USER` environment variable, which represents the username used to authenticate with the MySQL database.
- mysecret is the value of the `MYSQL_ROOT_PASSWORD` environment variable, which contains the password for the MySQL database's root user.
- data_mart_airbnb is the value of the `MYSQL_DATABASE` environment variable, which represents the name of the MySQL database that the application needs to interact with.

It is necessary to make adjustments to the `containers` segment, which serves the purpose of defining the configuration of the container integrated within a pod.

```
containers:
     - image: giprocida/myflask:1.0
       imagePullPolicy: Never
       name: myflask
       ports:
       - containerPort: 5000
       envFrom:
       - configMapRef:
           name: my-sql-config
```

Let's take a closer look at what we've added to the `containers` section in the given YAML

Here's an explanation of each part of this section:

- `imagePullPolicy: Never`: This specifies that Kubernetes should never attempt to pull the image from the container registry. It assumes that the image already exists on the node where the pod will be scheduled.
- `name: myflask`: This assigns the name "myflask" to the container.
- `envFrom:`: This section is used to specify that environment variables will be sourced from a different resource, in this case, a ConfigMap.
- `- configMapRef:`: This indicates that the environment variables will come from a ConfigMap.
- `name: my-sql-config`: This is the name of the ConfigMap from which the environment variables will be sourced.

Step 2: Expose the API

1. Service API YAML: Create a Kubernetes Service yaml file (api-svc.yaml) to expose the API. You can use imperative commands to create it :

`kubectl expose deploy api-deploy --port=5000 --target-port=5000 --type=NodePort --namespace=app-space --dry-run=client -o yaml > api-svc.yaml`

Here is a breakdown of what each part of the command does:

- `kubectl expose deploy api-deploy`: This part of the command initiates the exposure of a service using the Deployment named "api-deploy."
- `--port=5000`: This flag specifies that the service should expose port 5000. Other pods within the cluster can communicate with this server on the specified port.
- `--target-port=5000`: This flag specifies that the target port for the service should be set to port 5000. It’s the port on which the service will send requests to, that your pods (in this case, the API pods) will be listening on.

- `--type=NodePort`: This flag indicates that the service type should be a NodePort service. This means that the service will be accessible on a high-numbered port on all nodes in the cluster. Requests to this port will be forwarded to the pods.

Step 3: Deploy MySQL Database

1. Database Deployment YAML: Create a Deployment yaml file ('mysql-deploy') for the MySQL. You can use imperative commands to create it

`kubectl create deploy mysql-deploy --image=mysql --port=3306 --namespace=app-space --dry-run=client -o yaml > mysql-deploy.yaml`

- `kubectl create deploy mysql-deploy`: This part of the command instructs Kubernetes to create a new Deployment resource with the name "mysql-deploy."
- `--image=mysql`: This flag specifies the Docker image to use for the container in the deployment. In this case, the official MySQL image is being used.
- `--port=3306`: This flag specifies that the container in the deployment should listen on port 3306. This is the default port for MySQL database connections.

In order to initiate the database's state, it's essential to store the SQL script (data_ mart_airbnb.sql) within a ConfigMap. The SQL script comprises fabricated data relevant to Airbnb.

`kubectl create configmap my-sql-dump-config --from-file=data_mart_airbnb.sql --namespace=app-space -o yaml > my-sql-dump.yaml`

Here's what each part of this command does:

- `kubectl create configmap my-sql-dump-config`: This part of the command tells Kubernetes to create a new ConfigMap named "my-sql-dump-config."
- `--from-file=data_mart_airbnb.sql`: This flag specifies that the contents of the file named "data_mart_airbnb.sql" should be used to populate the ConfigMap. This SQL file will be stored as data in the ConfigMap.

It is necessary to make adjustments to the `spec` segment:

```
spec:
      volumes:
       - name: my-sql-dump
         configMap:
           name: my-sql-dump-config
      containers:
      - image: mysql
        name: mysql
        ports:
        - containerPort: 3306
        env:
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              configMapKeyRef:
                name: my-sql-config
                key: MYSQL_ROOT_PASSWORD
          - name: MYSQL_DATABASE
            valueFrom:
```

```
              configMapKeyRef:
                name: my-sql-config
                key: MYSQL_DATABASE
        volumeMounts:
          - name: my-sql-dump
            mountPath: /docker-entrypoint-initdb.d
```

- `volumes:`: This section defines the volumes that can be used by the containers in your pod.
- `- name: my-sql-dump`: This defines the name of the volume that you're creating, which is "my-sql-dump."
- `configMap:`: This specifies that the volume will be populated from a ConfigMap.
- `env:`: This is where you define environment variables for the container.
- `- name: MYSQL_ROOT_PASSWORD`: This defines an environment variable name `MYSQL_ROOT_PASSWORD`.
- `valueFrom:`: This indicates that the value of the environment variable will come from a different source.
- `configMapKeyRef:`: This specifies that the value will be fetched from a ConfigMap.
- `name: my-sql-config`: This is the name of the ConfigMap from which the value of the environment variable will be sourced.
- `key: MYSQL_ROOT_PASSWORD`: This indicates the specific key within the ConfigMap that holds the value for the environment variable.
- `- name: MYSQL_DATABASE`: This defines another environment variable named `MYSQL_DATABASE`.
- Similar to the `MYSQL_ROOT_PASSWORD` entry, the value for `MYSQL_DATABASE` is also sourced from the ConfigMap.
- `volumeMounts:`: This section is where you define how volumes should be mounted in the container.
- `- name: my-sql-dump`: This corresponds to the name of the volume defined earlier.
- `mountPath: /docker-entrypoint-initdb.d`: This specifies the directory path within the container where the volume should be mounted. The SQL script contained within the my-sql-dump volume will be mounted at the path /docker-entrypoint-initdb.d and subsequently executed. This action will efficiently initiate your MySQL database by incorporating the data from the SQL file.

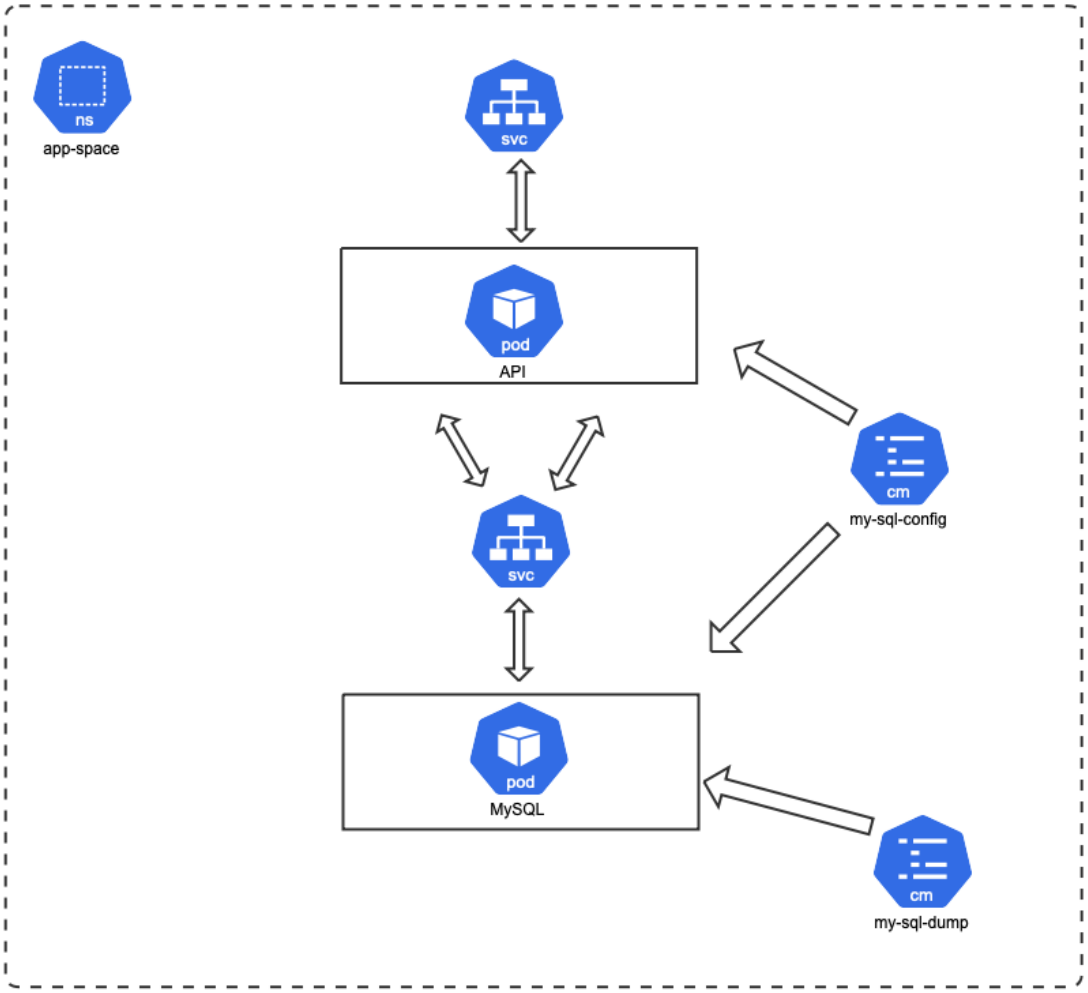
Finally, let's have a look at the diagram for the application architecture presented below.. Don't forget that the service API will be exposed on a port on each node in the cluster so that it will be reachable from external traffic. To gain deeper insights into NodePort, you can explore these resources: Kubernetes NodePort vs LoadBalancer vs Ingress? When should I use what?.

To test the application, you can use the curl command:

Here's what each part of this command does:

`curl -d "rating=5" -X POST http://localhost:31553/filter/rating`

- `curl`: This is the command-line tool used to make HTTP requests.
- `-d "rating=5"`: This flag specifies the data to be sent in the HTTP request body. In this case, it's sending the data "rating=5" to the server.
- `-X POST`: This flag specifies the HTTP request method to be used, which is "POST" in this case. It indicates that you want to send data to the server.
- `http://localhost:31553/filter/rating`: This is the URL to which the HTTP request will be sent. It's in the format of http://hostname:port/path


ns
app-space
svc
pod
API
cm
my-sql-config
svc
pod
MySQL
cm
my-sql-dump