Start your free 14-day ContainIQ trial

Resources

How to Deploy Postgres on Kubernetes | Tutorial

In this step by step tutorial, readers learn how to deploy PostgreSQL on Kubernetes. Readers will also learn some best practices to implement.

July 13, 2022

Hrittik Roy
Software Engineer

Kubernetes is an open source system for managing containerized applications. The orchestrator pro platform for automating deployment, scaling, and operating applications to facilitate easy manageme reliable operation of services. Currently hosted by the CNCF, it's one of the most popular orchestrati systems available, and is used across various organizations in different industries.

Meanwhile, PostgreSQL has become a popular choice for organizations looking for a database mana system that is both powerful and easy to use. Postgres is known for its reliability, flexibility, and perf making it an excellent choice for mission-critical applications.

Though Kubernetes was originally developed to primarily support stateless applications, it's grown t stateful applications, as well, and PostgreSQL has been widely adopted inside enterprises as a datal maintain the state. In this article, you'll walk through combining these popular technologies, Kuberne Postgres, with a step-by-step guide to deploying PostgreSQL on Kubernetes. You'll also look at whe may want to deploy Postgres on Kubernetes, as well as some best practices for doing so.

Why PostgreSQL on Kubernetes?

The combination of PostgreSQL and Kubernetes provides a scal ab solution that's well suited for modern application development a rea solution is out of the scope of this article, you'll learn how to set reS which offers a number of benefits.

Get the blog delivered directly to your inbox 📬
Enter your email here...
Subscribe

Improved Performance

and Kubernetes can be used to manage the deployment and scaling of the application as a whole.

Easier Disaster Recovery

You don't want to lose your operational or user data in any environment, but user error or technical f may result in it anyhow. PostgreSQL's Write-Ahead Logs (WAL) allows for easier disaster recovery by that all data is stored in the logs before the write operation to the database is performed, easing dat recovery when required, and allowing even unwritten updates to be salvaged.

Better Utilization of Resources

Kubernetes is very efficient with scaling, and allows for use cases like scaling pods up during peak h down afterwards without service interruption. Scaling helps optimize resource utilization and save o you use only the resources necessary, not over provisioning to accommodate an infrequent or irregu in demand.

Deploying PostgreSQL on Kubernetes

To deploy PostgreSQL on Kubernetes, you need to have some tools set up.

Prerequisites

- A working Kubernetes cluster. For this tutorial, a DigitalOcean cluster is used, but the steps of t tutorial will be the same for any cluster. To work locally, you can use something like kind or min set your cluster.
- A basic understanding of psql.
- kubectl installed and authenticated on your environment. You'll also need some working knowle the tool.

Deploying PostgreSQL

Deploying Postgres via ConfigMap with a PersistentVolume is one of the popular options for deployr it's the approach you'll be taking in this tutorial.

Apply ConfigMap

ConfigMaps help you separate data from code, and prevent secrets from exposing themselves in yo application's source code. With ConfigMaps, you can more easily deploy and update applications.

Create a ConfigMap by pasting the following code into your terminal:

```
cat <<EOF > postgres-config.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-config
  labels:
    app: postgres
```

Get the blog delivered directly to your inbox 📬
Enter your email here...
Subscribe

```
  POSTGRES_PASSWORD: psltest

EOF
```

The fields `POSTGRES_DB`, `POSTGRES_USER`, and `POSTGRES_PASSWORD` are your secrets, and you can
the values according to your preference. You can edit these values using text editors like vim or nan

Apply Manifest

The command below creates a new ConfigMap for our PostgreSQL deployment with a custom confi
The configuration consists of the fields `POSTGRES_DB`, `POSTGRES_USER`, and `POSTGRES_PASSWORD`.

```
@hrittikhere ➜ $ kubectl apply -f postgres-config.yaml
configmap/postgres-config created
```

Check ConfigMap

Use the following command to verify that your configmap is present and ensure you can locate `pos`
`config` on the terminal.

```
@hrittihere -> ~ $ kubectl get configmap

  NAME              DATA   AGE

  kube-root-ca.crt   1      86m

  postgress-config   3      3m42s
```

Create and Apply Persistent Storage Volume and Persistent Volume

In order to ensure data persistence, you should use a persistent volume (PV) and persistent volume
(PVC). A persistent volume (PV) is a durable volume that will remain even if the pod is deleted and s
data.

A persistent volume claim (PVC) is how users request and consume PV resources. Think of it as requ
the PV with parameters such as size of your storage disk, access modes, and storage class.

To deploy stateful applications such as a PostgreSQL database, for example, you'll need to create a
the database data. You can create a pod that mounts the PVC and runs the MySQL database.

For this tutorial, you will move forward with a local volume, using `/mnt/data` as the path to volume:

```
cat <<EOF > postgres-pvc-pv.yaml

kind: PersistentVolume
apiVersion: v1
metadata:
  name: postgres-pv-volume  # Sets PV's name
  labels:
    type: local  # Sets PV's type to local
    app: postgres
spec:
  storageClassName: manual
```

Get the blog delivered directly to your inbox 📬
Enter your email here...
Subscribe

```
    - ReadWriteMany
  hostPath:
    path: "/mnt/data"
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: postgres-pv-claim  # Sets name of PVC
  labels:
    app: postgres
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteMany  # Sets read and write access
  resources:
    requests:
      storage: 5Gi  # Sets volume size
EOF
```

Apply Manifest

Run the following command to create a new PVC and PV for your PostgreSQL deployment:

```
@hrittikhere ➜ ~ $ kubectl apply -f postgres-pvc-pv.yaml
persistentvolume/postgres-pv-volume created
persistentvolumeclaim/postgres-pv-claim created
```

Check PVC

Use the command below to check if PVC is bound to PV:

```
@hrittihere -> ~ $ kubectl get pvc

NAME                STATUS   VOLUME              CAPACITY   ACCESS MODES   STORAGE CLASS

postgress-pv-       Bound    postgress-pv-       5Gi        RWX            manual
claim                        volume
```

If the `STATUS` is “Bound”, you can use it for your deployments.

Create and Apply PostgreSQL Deployment

Deployments are a way to manage rolling out and updating applications in a Kubernetes cluster. The
a declarative way to define how an application should be deployed and updated, and can be used tc
to previous versions if needed.

After creating PVCs, PVs, and ConfigMaps, you can create a stateful application by creating a statef
follows:

Get the blog delivered directly to your inbox 📬
Enter your email here...
Subscribe

```
cat <<EOF > postgres-deployment.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres:10.1 # Sets Image
          imagePullPolicy: "IfNotPresent"
          ports:
            - containerPort: 5432  # Exposes container port
          envFrom:
            - configMapRef:
                name: postgres-config
          volumeMounts:
            - mountPath: /var/lib/postgresql/data
              name: postgredb
      volumes:
        - name: postgredb
          persistentVolumeClaim:
            claimName: postgres-pv-claim

EOF
```

Apply Manifest

The following command will create a new PostgreSQL deployment:

```
@hrittikhere ➜ ~ $ kubectl apply -f postgres-deployment.yaml
deployment.apps/postgres created
```

Successful Creation

Use the following command to check if your deployments and the children objects, such as pods, ar
successfully.

```
@hrittihere -> ~ $ kubectl get deployments

  NAME       READY   UP-TO-DATE   AVAILABLE   AGE

  postgress  0/1     1            0           21s

@hrittihere -> ~ $ kubectl get pods

  NAME                        READY   STATUS    RESTARTS   AGE

  postgress-7b9fb8d6c5-tnf68  1/1     Running   0          21s
```

Create and Apply PostgreSQL Service

Kubernetes services help you expose ports in various ways, inclu Por
a service on every node in a cluster, meaning that the service is er.
be useful for services that need to be accessible from outside th for
tutorial, you'll expose the database using NodePort with the help

Get the blog delivered directly to your inbox 📬
Enter your email here...
Subscribe

```
cat <<EOF > postgres-service.yaml

apiVersion: v1
kind: Service
metadata:
  name: postgres # Sets service name
  labels:
    app: postgres # Labels and Selectors
spec:
  type: NodePort # Sets service type
  ports:
    - port: 5432 # Sets port to run the postgres application
  selector:
    app: postgres

EOF
```

Apply Manifest

The command below will create a new PostgreSQL service which helps you to connect to `psql`:

```
@hrittikhere ➜ ~ $ kubectl apply -f postgres-service.yaml
service/postgres created
```

List All Objects

Listing all the objects can be done using the following command:

```
@hrittihere -> ~ $ kubectl get all

NAME                          READY   STATUS    RESTARTS   AGE
pod/postgress-7b9fb8d6c5-tnf68   1/1     Running   0          70s

NAME                 TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)         AGE
service/kubernetes   ClusterIP   10.245.0.1    <NONE>        443/tcp         90m
service/postgress    NodePort    10.245.155.3  <NONE>        5432:31710/TCP  20s

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/postgress   1/1     1            1           74s

NAME                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/postgress-7b9fb8d6c5   1         1         1       75s
```

Connect to PostgreSQL

The Kubernetes command line client ships with a feature that let rom host command line. The kubectl exec command accepts a pod n be executed, and an interactive flag that lets you launch a shell. You t to PostgreSQL pod:

Get the blog delivered directly to your inbox 📬
Enter your email here...
Subscribe

Use the password from the ConfigMap you created earlier, and the options `-it`.

- **-i:** Stands for interactive.
- **-t:** Attaches a tty (terminal) to the running command.

```
@hrittikhere ➜ ~ $ kubectl exec -it postgres-7b9fb8d6c5-tnf68 --  psql -h localhost
Password for user admin:
psql (10.1)
Type "help" for help.

postgresdb=# \l
                                  List of databases
    Name    |  Owner   | Encoding |  Collate   |   Ctype    |   Access privileges
------------+----------+----------+------------+------------+-----------------------
 postgres   | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 postgresdb | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 template0  | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
            |          |          |            |            | postgres=CTc/postgres
 template1  | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
            |          |          |            |            | postgres=CTc/postgres
(4 rows)

postgresdb=#
```

With PostgreSQL running, you're now able to connect to the database and start writing some data to tables.

Best Practices Deploying PostgreSQL on Kubernete

When deploying PostgreSQL on Kubernetes, there are some best practices that you should follow to the security and stability of your application.

Run the Container as Unprivileged User

You should always run the database container as an unprivileged user. This helps secure your data a unauthorized access to your database. The most essential things to ensure that you run the contain unprivileged user are:

*Make sure your container image launches as a user other than root (e.g. ensure USER is not 0 or ro

- Make sure your Pod Security Context is set to non-root by setting runAsNonRoot to true.

ncrypt Your Data

ou should always encrypt your data to avoid data loss or theft. ake
our data is encrypted in transit as well as in rest to prevent vari es.
ore about how to encrypt your data, check out this CNCF web

Create a Separate Namespace for Your Data

Overview

Kubernetes Monitoring Made Simple.
Start Free Trial
No card required

Get the blog delivered directly to your inbox 📬
Enter your email here...
Subscribe

unauthorized access.

A new database namespace also helps monitor resources, and you can apply limits if you need to ba resources.

You can create a namespace as follows:

```
kubectl create namespace [namespace-name]
```

Final Thoughts

In this tutorial, you've deployed a PostgreSQL database running on Kubernetes. This setup is great f use cases, but it's important to remember you have configured it to store data in node-local memory official documentation provides more details about support for cloud volumes, NFS, cephs, and mor

Finally, keep an eye on your resource usage, and scale your deployment accordingly to avoid perfor issues. You can do it with the help of ContainIQ, an out-of-the-box solution that allows you to monit health of your cluster and your objects. It monitors events, logs, and traces making troubleshooting and offers user-friendly, pre-built dashboards to allow you to get a view of your cluster health at a g

Start your free 14-day ContainIQ trial

Start Free Tr

No card require

Hrittik Roy

Software Engineer

Hrittik is a writer and a software engineer specializing in cloud native ecosystems. He has worked on many large-scale projects and has experience in both the technical and the business aspects of cloud computing. He is a frequent speaker at conferences and has written numerous articles on software development and distributed systems. In his free time, he likes to go for long walks.

READ MORE

Get the blog delivered directly to your inbox

Enter your email here...

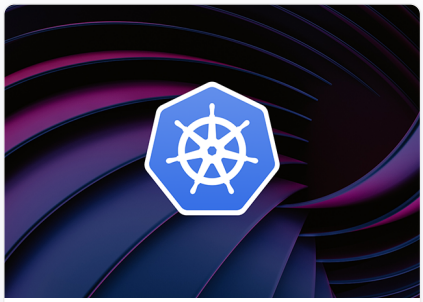
Subscribe

Looking to learn more? The below posts may be helpful for you to learn more about Kubernetes and our company.

Resources

Collecting Logs from AWS Fargate | Tutorial

Oghenevwe... November 27, 2022



Resources

Kubernetes in Production | 8 Tips & Best Practices in 2022

Sudip Sen... October 23, 2022

Resources

Automated Distributed Tracing Using eBPF (Part 1)

Matt Lenh... October 20, 2022

Get the blog delivered directly to your inbox 📬

Enter your email here...

Subscribe