



# Postgres on Kubernetes Workshop Florence

Davide Tammaro, Principal Sales Engineer EMEA

Sergio Romera, Senior Sales Engineering EMEA South Team

10 June, 2025



EDB  
Postgres for the AI Generation



WEBINAR  
17TH JUNE AT 11:00 AM

# Meet the Future of EDB Postgres® AI

Each day, 13 more enterprises choose to build their sovereign data and AI platform on Postgres.

**Will June 17 be the day you do?**

---

## Enter our Prize Draw!

Webinar attendees can enter for a chance to win two tickets to the Goodwood Festival of Speed (UK, July 12–13)



# Agenda

Start	End	Session	
14:00	14:30	Registration & Welcome	
14:30	14:45	Introduction to Postgres and EDB (EDB - Davide Tammaro)	
14:45	15:00	EDB CloudNativePG Operator	
15:00	16:00	Interactive session It's time to go hands-on! (EDB - Sergio Romera)	1/2
16:00	16:15	Coffee time	
16:15	17:30	Interactive session It's time to go hands-on! (EDB - Sergio Romera)	2/2



# CREW for today's workshop

Red Hat team



Natale Vinto

EDB team



Davide Tammaro  
Principal Sales Engineer EMEA



Sergio Romera  
Senior Manager, Sales Engineering EMEA Team



# Introduction to Postgres and EDB



# 20+ years of Postgres innovation & adoption

- Number one contributor to Postgres, fastest-growing and most loved Database in the world
  - 2 Core Team members, 7 Committers, 9 Major Contributors, 10 Contributors, #1 site for desktop downloads
- Over 700 employees in more than 30 countries
- EDB Postgres AI
  - The industry's first platform that can be deployed as cloud, software or physical appliance
  - Secure, compliant and enterprise grade performance guaranteed



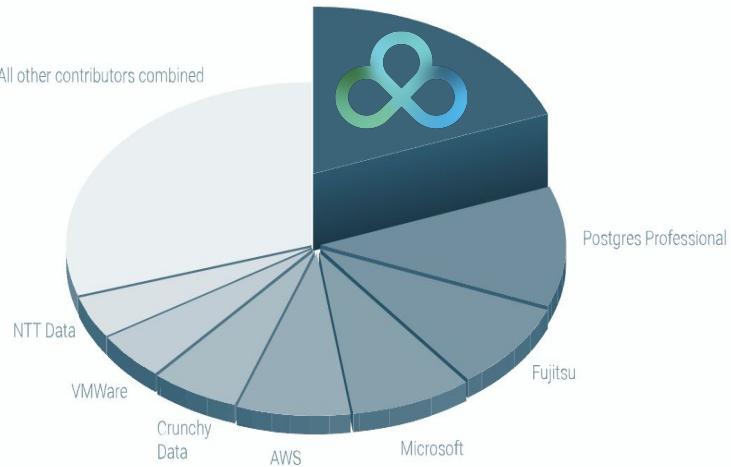
## 20+ years of innovation

- 760+ employees, 300 dedicated to Postgres
- 79 countries

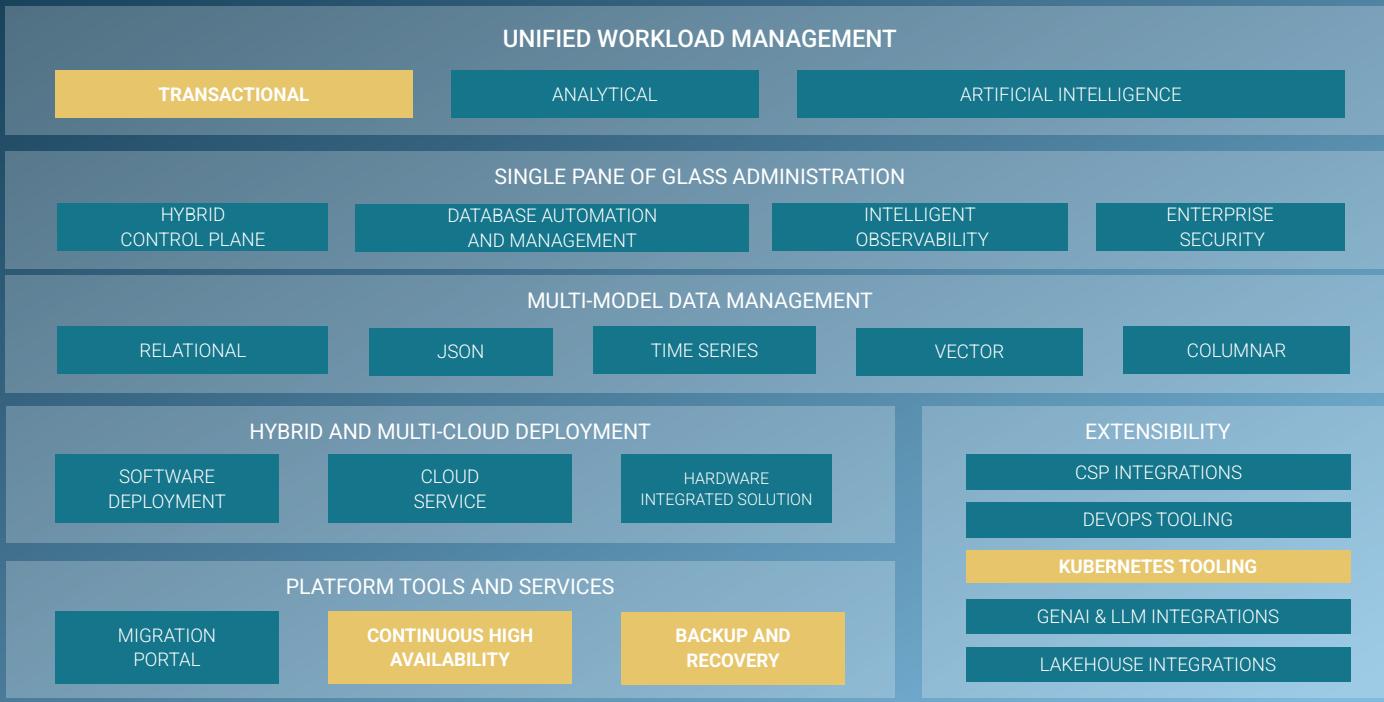
## #1 contributor to Postgres

- Present in the Core Team
- 7 Committers
- 9 Major Contributors

30% + of Postgres Code Contributed in 2024



# EDB POSTGRES AI PLATFORM





# EDB CloudNativePG Operator

# A kubernetes operator for Postgres



Kubernetes adoption is rising and it is already the de facto **standard** orchestration tool



PostgreSQL clusters “management the kubernetes way” enables many cloud native usage patterns, e.g. spinning up, disposable clusters during tests, one cluster per microservice and one database per cluster



CNP tries to encode years of experience managing PostgreSQL clusters into **an Operator which should automate all the known tasks** a user could be willing to do

Our PostgreSQL operator must simulate the work of a DBA

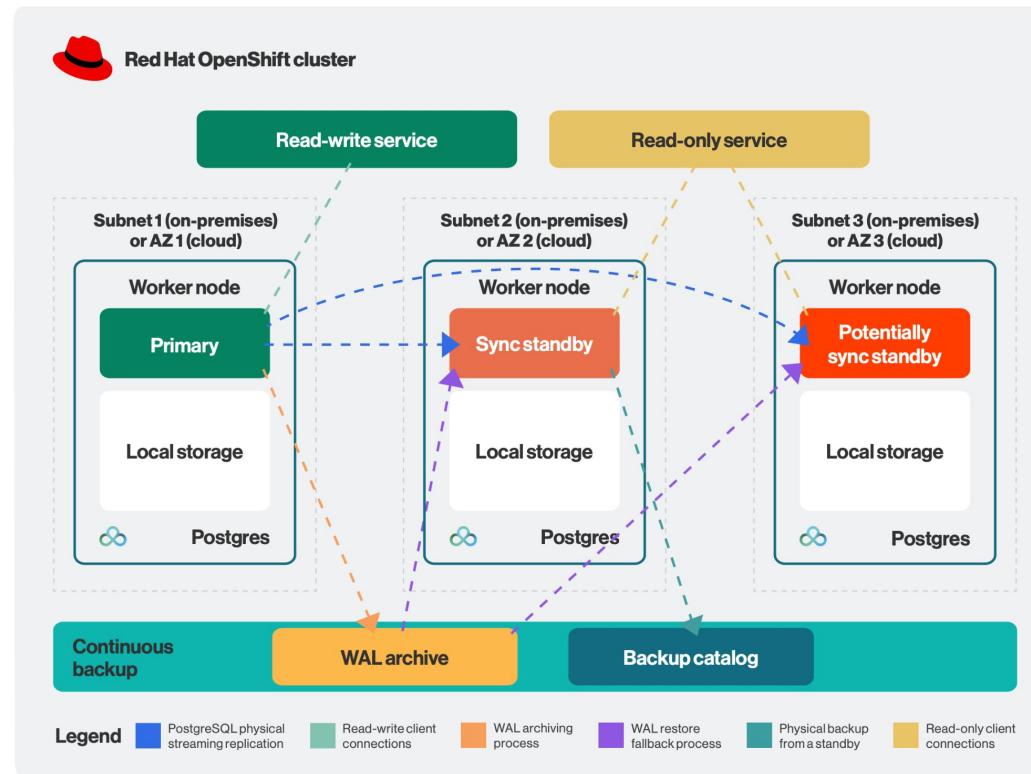


# Features

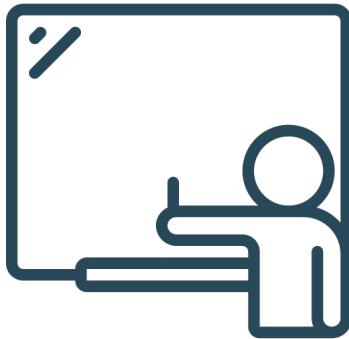
Deployment	Administration	Backup & Recovery	Monitoring	Security	High Availability
Kubernetes operator	Single node	Backup	Prometheus	TDE	Switchover
Kubernetes plugin	Cluster (Multi node)	Recovery	Grafana dashboards	Certificates	Failover
EDB Postgres (EPAS)	PostgreSQL configuration	PITR	Postgres Enterprise Manager	Data masking	Scale out / scale down
PostGIS	Logging	Volume Snapshots			Minor rolling updates
	Pooling				Major updates



# Architecture used in the Workshop



# Use the whiteboard for Q&A





# Interactive session

# It's time to go hands-on!

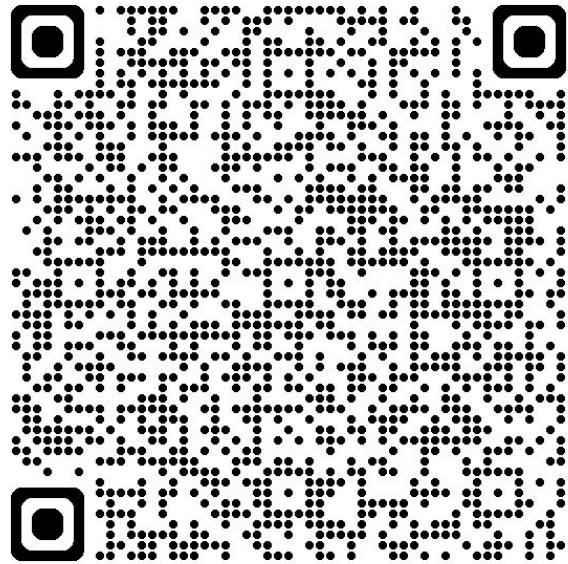


Hand-on documentation



Download this presentation

[bit.ly/43vqkcn](https://bit.ly/43vqkcn)





## Log in to your account

Username \*

Password \*



Welcome to Red Hat OpenShift



Starting workspace enterprisedb-workshop

Progress Logs Events

- 1 Initializing
- 2 Checking for the limit of running workspaces
- 3 Creating a workspace
- 4 Waiting for workspace to start
- 5 Open IDE

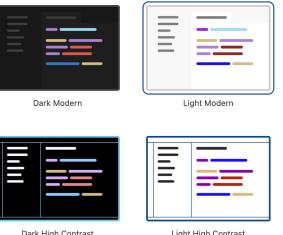
Get Started with VS Code for the Web

Customize your editor, learn the basics, and start coding

Choose your theme  
The right theme helps you focus on your code, is easy on your eyes, and is simply more fun to use.  
[Browse Color Themes](#)  
Tip: Use keyboard shortcut

Just the right amount of UI

Rich support for all your languages



Dark Modern Light Modern  
Dark High Contrast Light High Contrast  
[See More Themes...](#)

Do you trust the authors of the files in this workspace?



VS Code - Open Source provides features that may automatically execute files in this workspace.

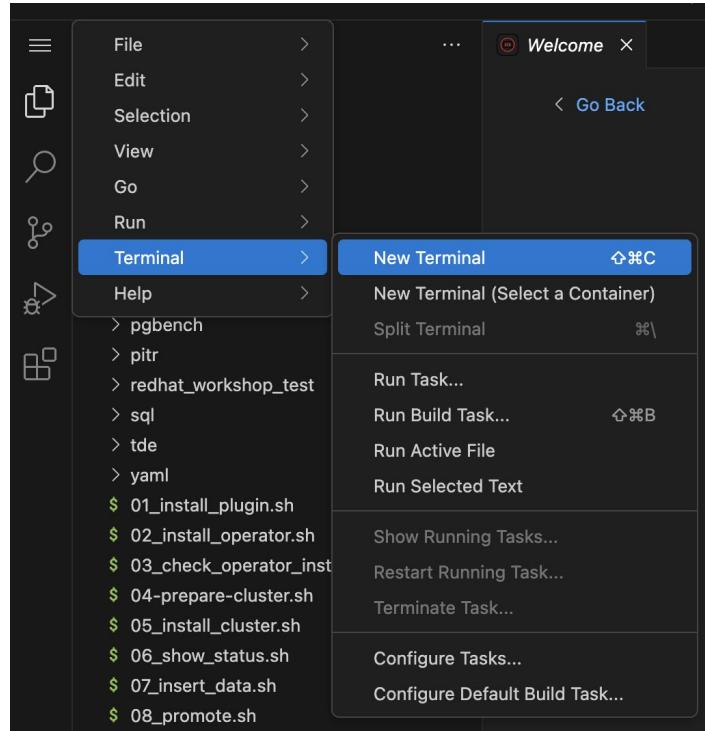
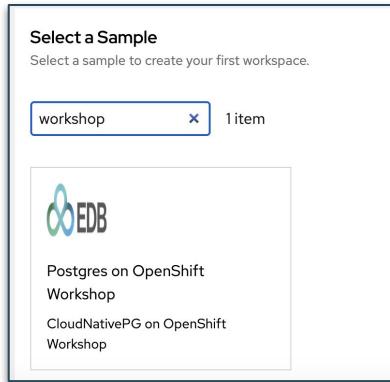
If you don't trust the authors of these files, we recommend to continue in restricted mode as the files may be malicious. See [our docs](#) to learn more.

/projects (Workspace)

*Browse workspace in restricted mode* *Trust workspace and enable all features*



# How to connect to the terminal



# Red Hat OpenShift environment

## OpenShift Console:

<https://console-openshift-console.apps.cluster-cx9nq.dynamic.redhatworkshops.io>

## OpenShift API for command line 'oc' client:

<https://api.cluster-cx9nq.dynamic.redhatworkshops.io:6443>

Users user1 .. user50 created with password edb-workshop

DevSpace: <https://devspaces.apps.cluster-cx9nq.dynamic.redhatworkshops.io/>

## Minio

User: minio

Password: edb-workshop

URL UI: <https://minio-ui-minio.apps.cluster-cx9nq.dynamic.redhatworkshops.io>



# Use case

# The environment



# Features shown during the demo

- Kubernetes plugin install
- Check the CloudNativePG operator status
- Postgres cluster install
- Insert data in the cluster
- Failover
- Backup
- Recovery
- Scale out/down
- Fencing
- Hibernation
- Rolling updates (minor and major)

Deployment

High Availability

Administration

Monitoring

Backup and Recovery

Last CloudNativePG tested version is 1.25

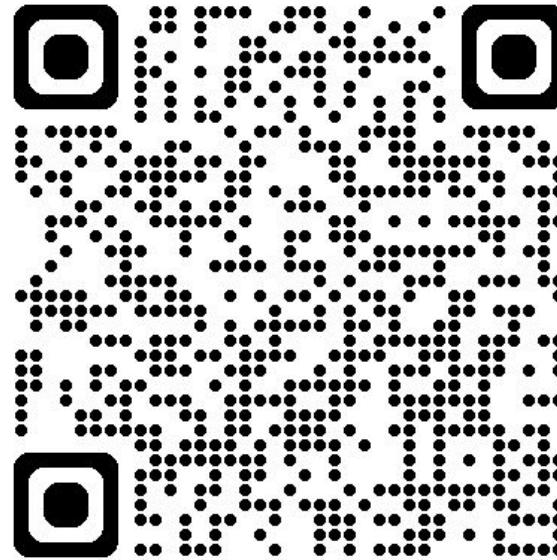


This demo is in



<https://github.com/sergioenterprisedb/edb-postgres-for-kubernetes-in-openshift>

<http://bit.ly/4duKxm7>



# Plug-in installation



# The “cnp” plugin for kubectl

- The official CLI for CloudNativePG
  - Available also as RPM or Deb package
- Extends the ‘kubectl’ command:
  - Customize the installation of the operator
  - Status of a cluster
  - Perform a manual switchover (promote a standby) or a restart of a node
  - Issue TLS certificates for client authentication
  - Declare start and stop of a Kubernetes node maintenance
  - Destroy a cluster and all its PVC
  - Fence a cluster or a set of the instances
  - Hibernate a cluster
  - Generate jobs for benchmarking via pgbench and fio
  - Issue a new backup
  - Start pgadmin

**NOT NEEDED DURING WORKSHOP**  
**For illustrative purposes.**



```
Every 2.0s: kubectl-cnpg --color always status cluster-example
```

MAC-FWCNWDWQ4C.local: Thu Jun 5 17:59:11 2025

#### Cluster Summary

```
Name           default/cluster-example
System ID:    7510182698349051927
PostgreSQL Image: gcr.io/cloudnative-pg/postgresql:16.2
Primary instance: cluster-example-1
Primary start time: 2025-05-30 10:24:37 +0000 UTC (uptime 149h34m35s)
Status:        Cluster in healthy state
Instances:     3
Ready instances: 3
Size:          207M
Current Write LSN: 0/C000000 (Timeline: 1 - WAL File: 00000001000000000000000B)
```

#### Continuous Backup status

```
Not configured
```

#### Streaming Replication status

##### Replication Slots Enabled

Name	Sent LSN	Write LSN	Flush LSN	Replay LSN	Write Lag	Flush Lag	Replay Lag	State	Sync State	Sync Priority	Replication Slot
---	---	---	---	---	---	---	---	---	---	---	---
cluster-example-2	0/C000000	0/C000000	0/C000000	0/C000000	00:00:00	00:00:00	00:00:00	streaming	quorum	1	active
cluster-example-3	0/C000000	0/C000000	0/C000000	0/C000000	00:00:00	00:00:00	00:00:00	streaming	quorum	1	active

#### Instances status

Name	Current LSN	Replication role	Status	QoS	Manager	Version	Node
---	---	---	---	---	---	---	---
cluster-example-1	0/C000000	Primary	OK	Burstable	1.26.0		docker-desktop
cluster-example-2	0/C000000	Standby (sync)	OK	Burstable	1.26.0		docker-desktop
cluster-example-3	0/C000000	Standby (sync)	OK	Burstable	1.26.0		docker-desktop



# Install CNPG plugin

NOT NEEDED DURING WORKSHOP  
For illustrative purposes.

- In the web terminal run the script 01\_install\_plugin.sh:

```
./01_install_plugin.sh
```

- Call the help for the CNPG Plugin, run:

```
kubectl-cnp help
```



# Check Operator installation



# Operator Installation demonstration

- Check the installed CNP Operator in the console
- Discover the features of the Operator in the OpenShift environment
- Check the installed CNP Operator in the web terminal



NOT NEEDED DURING WORKSHOP  
For illustrative purposes.

# Check the installed EDB CloudNAtivePG Operator in the console

- In the OpenShift console navigate to:
  - -> Operators
  - -> Installed Operators
  - -> Click on the Operator installed in your namespace, for example: user1:

The screenshot shows the Red Hat OpenShift web console interface. The left sidebar navigation includes 'Administrator', 'Home', 'Operators' (selected), 'OperatorHub', 'Installed Operators' (selected), 'Workloads', 'Networking', and 'Secrets'. The main content area is titled 'Installed Operators' and displays the following information:

Name	Managed Namespaces	Status	Last updated	Provided APIs
EDB Postgres for Kubernetes 1.24.3 provided by EDB	All Namespaces	<span style="color: green;">Succeeded</span> Up to date	21 May 2025, 15:21	Backups Cluster Image Catalog Cluster Image Catalog View 2 more...



# Discover the features of the Operator in the OpenShift environment

Red Hat OpenShift    kubeadmin ▾

Administrator ▾

Home >

Operators ▾

OperatorHub

Installed Operators

Workloads >

Networking >

Storage >

Builds >

Compute >

User Management >

Administration >

Project: openshift-operators ▾

Installed Operators > Operator details

EDB Postgres for Kubernetes  
124.3 provided by EDB

Actions ▾

Details YAML Subscription Events All instances Backups Cluster Image Catalog Cluster Image Catalog Pooler Scheduled Backups

Provided APIs

<b>Backups</b> PostgreSQL backup (physical base backup)  <a href="#">Create instance</a>	<b>Cluster Image Catalog</b> A cluster-wide catalog of PostgreSQL operand images  <a href="#">Create instance</a>	<b>Cluster</b> PostgreSQL cluster (primary/standby architecture)  <a href="#">Create instance</a>	<b>Image Catalog</b> A catalog of PostgreSQL operand images  <a href="#">Create instance</a>	<b>Pooler</b> Pooler for a Postgres Cluster (with PgBouncer)  <a href="#">Create instance</a>
---	--	--	---	--

Provider  
EDB

Created at  
21 May 2025, 15:21

Links  
EDB Postgres for Kubernetes  
<https://www.enterprisedb.com/products/postgresql-on-kubernetes-ha-clusters-k8s-containers-scalable>

Documentation  
[https://www.enterprisedb.com/docs/postgres\\_for\\_kubernetes/latest/](https://www.enterprisedb.com/docs/postgres_for_kubernetes/latest/)

Maintainers  
Jonathan Gonzalez V.  
[jonathan.gonzalez@enterprisedb.com](mailto:jonathan.gonzalez@enterprisedb.com)

Jonathan Battato  
[jonathan.battato@enterprisedb.com](mailto:jonathan.battato@enterprisedb.com)

Niccolo Fei  
[niccolo.fei@enterprisedb.com](mailto:niccolo.fei@enterprisedb.com)

Gabriele Bartolini  
[gabriele.bartolini@enterprisedb.com](mailto:gabriele.bartolini@enterprisedb.com)

Description

Main features:



# Check the installed CNPG Operator in the web terminal

- In the web terminal check the installation of the operator:

```
./02_install_operator.sh (will require admin privs on Openshift)
```

```
./03_check_operator_installed.sh
```

NOT NEEDED DURING WORKSHOP  
For illustrative purposes.



# Create the PostgreSQL cluster



# Bootstrap - different ways of creating a cluster

- Create a new cluster from scratch
  - “initdb”: named after the standard “initdb” process in PostgreSQL that initializes an instance
- Create a new cluster from an existing one:
  - Directly (“pg\_basebackup”), using physical streaming replication
  - Directly (logical backup/restore) using pg\_dump and pg\_restore
  - Indirectly (“recovery”), from an object store
    - To the end of the WAL
      - Can be used to start independent replica clusters in continuous recovery
    - Using PITR



# Configure and Install the Postgres cluster

- Prepare for cluster-creation (ensure minio secrets are in place)

```
./04-prepare-cluster
```

- Create a new 3-node cluster by running

```
./05_install_cluster.sh
```

- Check the status of the cluster (using the CNP plugin):

```
./06_show_status.sh
```



# Create table test with 1000 rows

- Once cluster is running ... (minimum the primary) run the script:

```
./07_insert_data.sh
```

Try it for yourself

10:00



# Promote & Upgrade the PostgreSQL cluster



# Rolling updates

- Update of a deployment with ~zero downtime
  - Standby servers are updated first
  - Then the primary:
    - supervised / unsupervised
    - switchover / restart
- When they are triggered:
  - Security update of Postgres images
  - Minor update of PostgreSQL
  - Configuration changes when restart is required
  - Update of the operator
    - Unless in-place upgrade is enabled



# Check the cluster status

- In terminal **1**: (prepare a terminal for status - and one to run the admin-commands):

- Run the command

```
./06_show_status.sh
```

- Review the output:
    - Check Postgres version:

- **imageName: quay.io/enterprisedb/postgresql:16.2**

- check "Continuous Backup status": "**Not configured**"

- Check the updated cluster configuration - file cluster-example-upgrade.yaml

```
less ./yaml/cluster-sample-upgrade.yaml
```

- Check Postgres version:

- **imageName: quay.io/enterprisedb/postgresql:16.4**

- Check the Backup section



# Run the Promote and Upgrade

- With this step we will:
  - Promote node-2 to become the primary
  - Run the postgres minor update from the version 16.2 to 16.4
  - We will configure the WAL files backup to the S3 storage
- In the web terminal **2**:
  - Check the upgrade status:  
`./06_show_status.sh`
- In the terminal **1**:
  - Run the script:  
`./08_promote.sh`
  - Run the script:  
`./09_upgrade.sh`

Try it for yourself

05:00



# Backup & Restore



# Backup and Recovery - Part 1

- Continuous physical backup on “backup object stores”
  - Scheduled and on-demand base backups
  - Continuous WAL archiving (including parallel)
  - From primary or a standby
  - Support for recovery window retention policies (e.g. 30 days)
- Recovery means creating a new cluster starting from a “recovery object store”
  - Then pull WAL files (including in parallel) and replay them
  - Full (End of the WAL) or PITR
- Both rely on Barman Cloud technology
  - AWS S3
  - Azure Storage compatible
  - Google Cloud Storage
  - MinIO



# Backup and Recovery - Part 2

- WAL management
  - Object store
- Physical Base backups
  - Object store
  - Kubernetes level backup integration (Velero/OADP, Veem Kasten K10, generic interface)
  - Kubernetes Volume Snapshots



# Kubernetes Volume Snapshot: major advantages

- Transparent support for:
  - Incremental backup and recovery at block level
  - Differential backup and recovery at block level
  - Based on copy on write
- Leverage the storage class to manage the snapshots, including:
  - Data mobility across network (availability zones, Kubernetes clusters, regions)
  - Relay files on a secondary location in a different region, or any subsequent one
  - Encryption
- Enhances Very Large Databases (VLDB) adoption



# Backup & Recovery via Snapshots: some numbers

Let's now talk about some initial benchmarks I have performed on volume snapshots using 3 `r5.4xlarge` nodes on AWS EKS with the `gp3` storage class. I have defined 4 different database size categories (tiny, small, medium, and large), as follows:

Cluster name	Database size	pgbench init scale	PGDATA volume size	WAL volume size	pgbench init duration
<i>tiny</i>	4.5 GB	300	8 GB	1 GB	67s
<i>small</i>	44 GB	3,000	80 GB	10 GB	10m 50s
<i>medium</i>	438 GB	3,0000	800 GB	100 GB	3h 15m 34s
<i>large</i>	4,381 GB	300,000	8,000 GB	200 GB	32h 47m 47s

The table below shows the results of both backup and recovery for each of them.

	Cluster name	1st backup duration	2nd backup duration after 1hr of pgbench	Full recovery time
	<i>tiny</i>	2m 43s		4m 16s
	<i>small</i>	20m 38s		16m 45s
	<i>medium</i>	2h 42m		2h 34m
	<i>large</i>	3h 54m 6s		2m 2s

<https://www.enterprisedb.com/postgresql-disaster-recovery-with-kubernetes-volume-snapshots-using-cloudnativepg>



# Create the full backup

- With this step we will:
  - Create the full backup of the postgres cluster in the MinIO storage:
- In the web terminal 1:
  - Run the script:

```
./10_backup_cluster.sh
```

- Check the backup status:

```
./11_backup_describe.sh
```



# Check Backup in the Openshift Console

- Navigate to:
  - -> Operators
  - -> Installed Operators
  - -> Press on the Operator installed in your namespace, for example: user1:
  - -> Go to the Backup section and show the created backup:

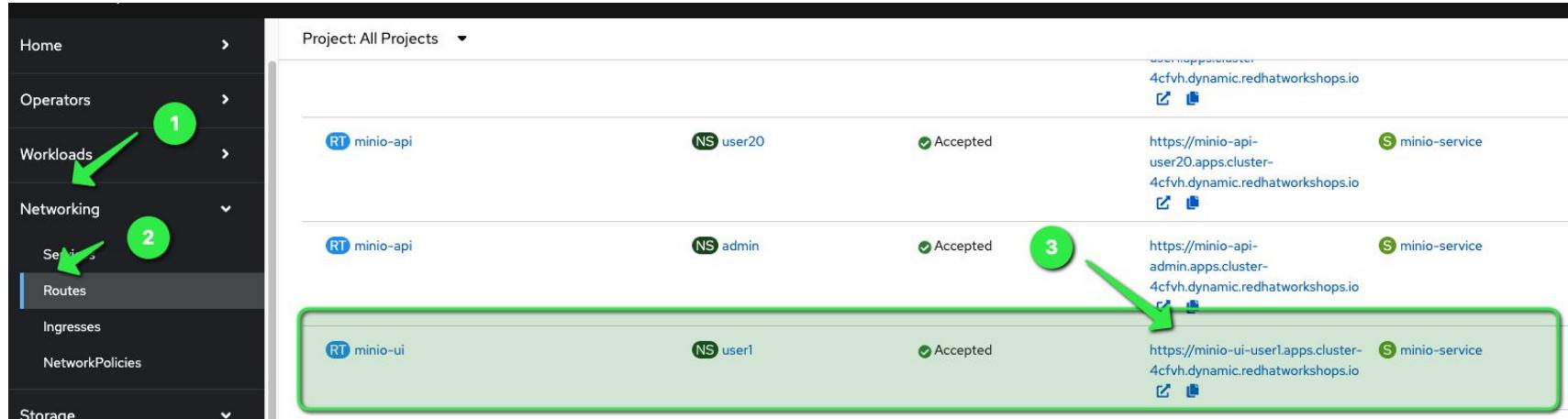
The screenshot shows the OpenShift console interface. On the left, there is a sidebar with navigation links: Home, Operators (selected), Workloads, Networking, Storage, and Builds. Under Operators, there is a sub-section for OperatorHub and Installed Operators (which is currently selected). The main content area has a header "Project: openshift-operators". Below the header, it says "Installed Operators > Operator details" for "EDB Postgres for Kubernetes" version 1.24.3 provided by EDB. There are tabs for Details, YAML, Subscription, Events, All instances, Backups (which is underlined in blue), Cluster Image Catalog, Cluster, Image Catalog, and Pod. The "Backups" section contains a search bar with "Name" and "Search by name..." placeholder text. It also has radio buttons for "Show operands in: All namespaces" (selected) and "Current namespace only". A table lists two backup entries:

Name	Kind	Namespace	Status
backup-test	Backup	NS edb-emea-user1	Phase: complete
cluster-user1-backup-test	Backup	NS edb-emea-user1	Phase: complete



# Check Backup in MinIO UI

- In the Openshift console navigate to Networking -> Routes
- Search for route minio-ui for your user and press the button with url:



# Check Backup in MinIO UI

- Connect as user **admin** with the password: **password**
- The page will appear:

The screenshot shows the MinIO Object Browser interface. On the left, a sidebar menu includes 'User' sections for 'Object Browser', 'Access Keys', 'Documentation', and 'Administrator' sections for 'Buckets', 'Policies', and 'Identity'. The 'Object Browser' item is highlighted with a blue background. The main content area is titled 'Object Browser' with a back arrow. It displays a search bar with the placeholder 'Start typing to filter objects in the bucket'. Below the search bar, a table lists objects in the 'cnp' bucket. The table has columns for 'Name' and 'Last Modified'. One object, 'cluster-example', is listed under 'Name'.

Name	Last Modified
cluster-example	

← Object Browser

Start typing to filter objects in the bucket

cnp

Created on: Mon, Apr 14 2025 18:54:07 (GMT+2) Access: PRIVATE 63.6 MiB - 6 Objects

Name Last Modified

cluster-example



# Restore the database from the backup

- With this step we will:
  - Create the new cluster cluster-restore
  - Restore the full backup created in the previous step in the new cluster:
- In the terminal 1:
  - Run the restore:  
**`./12_restore_cluster.sh`**
  - Check the creation status:  
`kubectl get pods -w` # after creation stop the execution with <ctrl>+c
  - Check the table test in the cluster-restore, run the script:  
**`oc exec -it cluster-restore-user<X>-1 - psql -U postgres -c "\d test"`**
  - Delete the cluster-restore-user<x> to avoid resource problems during the workshop:  
**`oc delete cluster cluster-restore-user<X>`**



# Backup demonstration

- Create the full backup
- Check Backup in the Openshift Console
- Check Backup in MinIO UI
- Restore the database from the backup

Try it for yourself

15:00



# Failover



# Run failover test

- With this step we will:
  - Delete the primary database of the cluster cluster-example
  - Check the cluster status in the another terminal window
- In the web terminal 1:
  - Run the script:  
`./13_failover.sh`
- In the web terminal **2**:
  - Check the failover cluster status:  
`./06_show_status.sh`

Try it for yourself

05:00



# Use case Scale-out and scale-down



# Scale-out the postgres cluster

- With this step we will:
  - Add the 1 standby to the cluster
- In the web terminal 1:
  - Run the script:

```
./14_scale_out.sh (using -replicas=X... another way would be to update the YAML)
```
- In the web terminal **2**:
  - Check the cluster status:

```
./06_show_status.sh
```



# Scale-down the postgres cluster

- With this step we will:
  - Remove 2 standby pods from the cluster

- In the web terminal 1:

- Run the script:

```
./15_scale_down.sh
```

- In the web terminal **2**:

- Check the cluster status:

```
./06_show_status.sh
```

## Try it for yourself

05:00



# Fencing



# Stop postgres process on the pod

- In the web terminal 1:
  - Run the script:

```
./30_fencing_on.sh
```

- In the web terminal **2**:
  - Check the cluster status:

```
./06_show_status.sh
```



# Start the postgres process on the pod

- In the terminal 1:

- Run the script:

```
./31_fencing_off.sh
```

- In the terminal **2**:

- Check the cluster status:

```
./06_show_status.sh
```

## Try it for yourself

05:00



# Hibernation



# Stop the postgres cluster

- In the terminal 1:
  - Run the script:

```
./32_hibernation_on.sh
```

- In the terminal **2**:
  - Check the cluster status:

```
./06_show_status.sh
```



# Start the postgres cluster

- In the terminal 1:
  - Run the script:

```
./33_hibernation_off.sh
```

- In the terminal **2**:
  - Check the cluster status:

```
./06_show_status.sh
```

Try it for yourself

05:00



# Major version Upgrade



# Delete cluster restore and upgrade cluster

- In the web terminal 1:
  - Delete the cluster cluster-restore:

```
kubectl delete cluster cluster-restore-user<x>
```

- In the web terminal 1:

```
./20_upgrade_major_version.sh
```



What more?  
(some additional features from EDB)

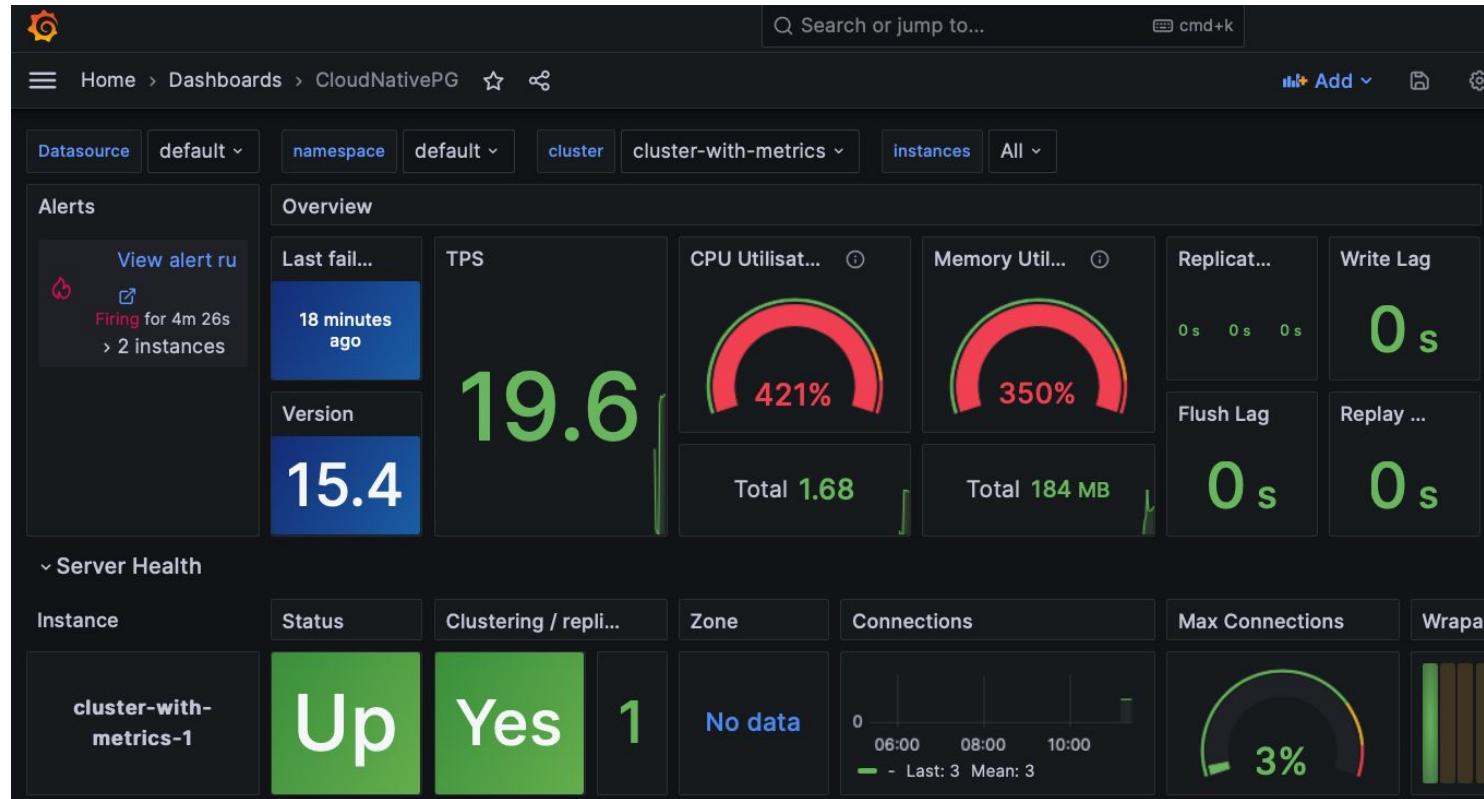


# What we didn't show you today ....

- PgBouncer (Pooler) integration
  - Create a PgBouncer deployment and automatically configure to the cluster.
- Monitoring using Prometheus and Grafana
  - Exporting to OpenMetrics (Prometheus)



# Grafana Dashboard



# Advanced Security



## Password policy management

DBA managed password profiles, compatible with Oracle profiles



## Audit compliance

Track and analyze database activities and user connections



## Virtual private databases

Fine grained access control limits user views



## EDB/SQL protect

SQL firewall, screens queries for common attack profiles



## Data redaction

Protect sensitive information for GDPR, PCI and HIPAA compliance



## Code protection

Protects sensitive IP, algorithms or financial policies

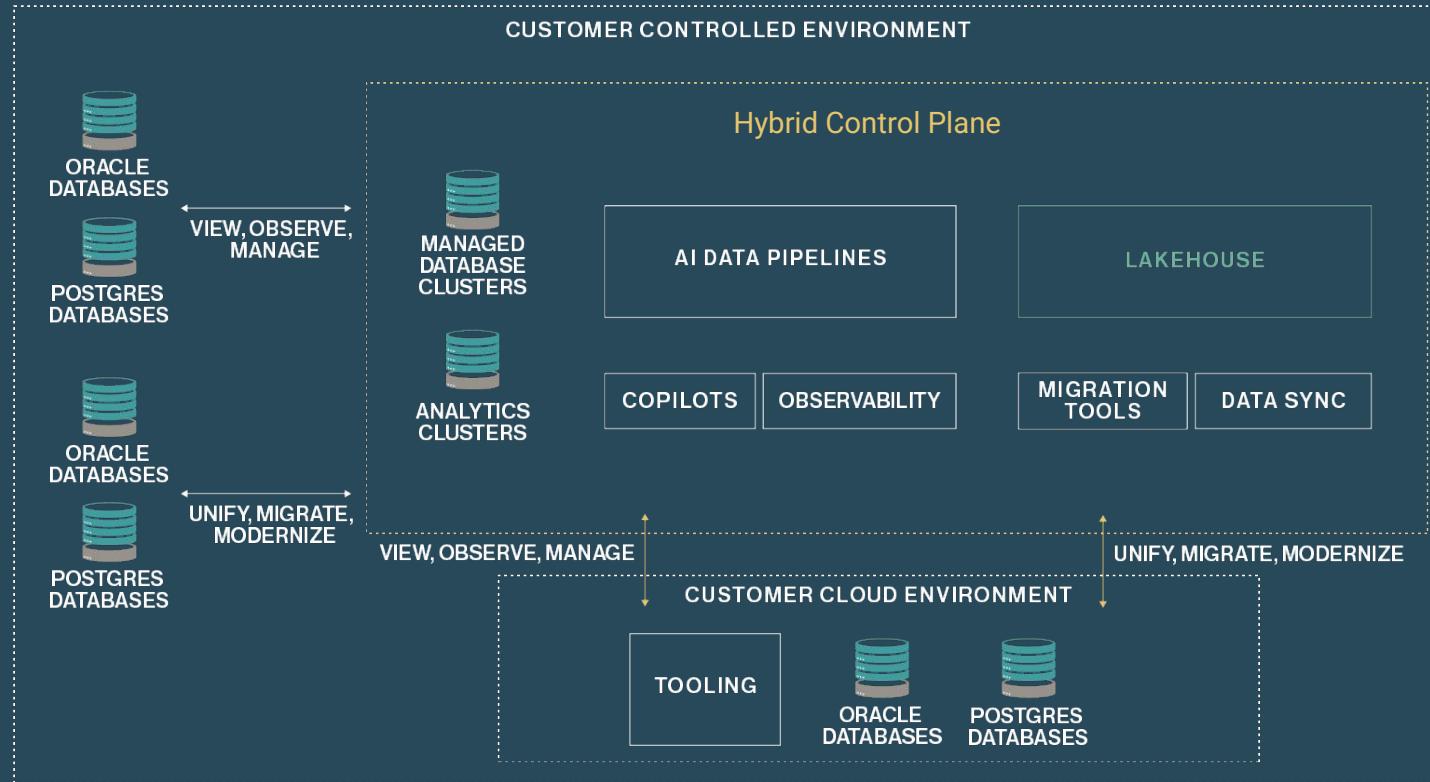


# Transparent Data Encryption (EDB-only features)

- Transparent Data Encryption (TDE) is a feature of EDB Postgres Advanced Server and EDB Postgres Extended Server that prevents unauthorized viewing of data in operating system files on the database server and on backup storage
- Data encryption and decryption is managed by the database and does not require application changes or updated client drivers
- EDB Postgres Advanced Server and EDB Postgres Extended Server provide hooks to key management that is external to the database allowing for simple passphrase encrypt/decrypt or integration with enterprise key management solutions, with initial support for:
  - Amazon AWS Key Management Service (KMS)
  - Google Cloud - Cloud Kay Management Service
  - Microsoft Azure Key Vault
  - HashiCorp Vault (KMIP Secrets Engine and Transit Secrets Engine)
  - Thales CipherTrust Manager
- Data will be unintelligible for unauthorized users if stolen or misplaced



# Hybrid Control Plane at a glance

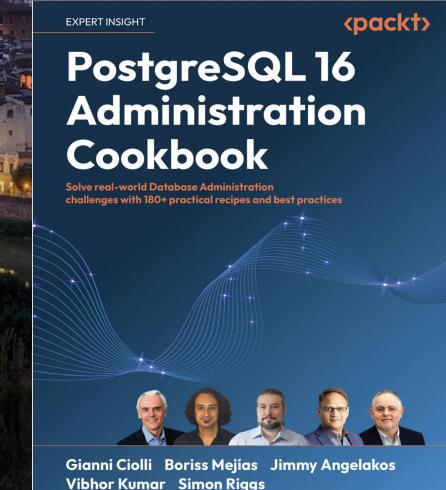




Share Your Opinion &  
enter for a chance to Win  
a Postgres Cookbook



Scan the QR Code & complete the survey  
to let us know your feedback!



By accepting this prize, you confirm that the prize conforms to your employer's internal rules, policies, and codes of conduct.

<https://wheelofnames.com/>



By accepting this prize, you confirm that the prize conforms to your employer's internal rules, policies, and codes of conduct.



Thank you for participating in the  
Postgres on Kubernetes Workshop

Please pick up your certificate :-)



# Thank you



# Annexes



# Advantage of deploying Postgres Databases in Kubernetes

## Automation & Orchestration

- 01 |
- Self-healing
  - Automated scaling
  - Rolling updates

## Self-healing

- 02 |
- Best resource utilization
  - Dynamic Resource allocation

## Rolling updates

- 03 |
- Cloud-agnostic
  - Consistent deployment

## Service discovery & networking

- 04 |
- Built-it service discovery
  - Load Balancing

## Automated backups and disaster recovery

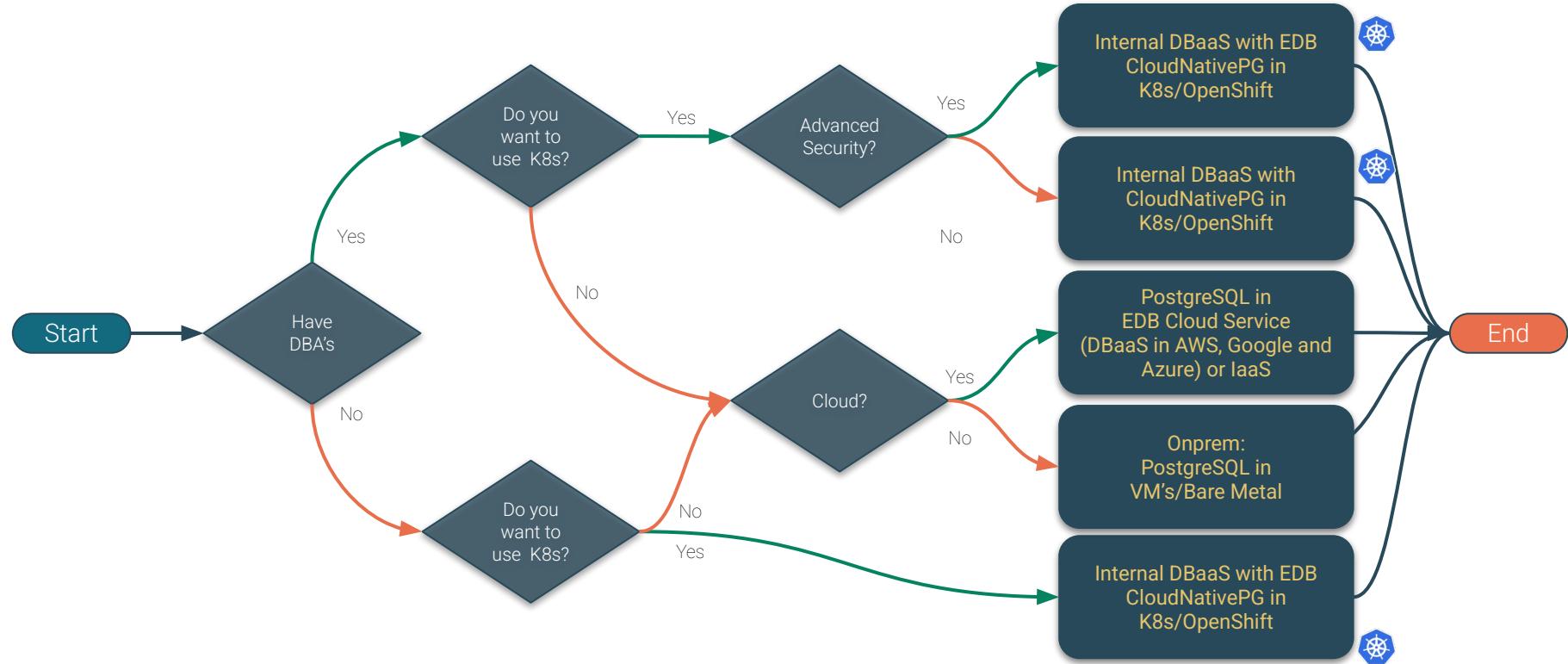
- 05 |
- Automated backups
  - Multi-region failover

## Security & access control

- 06 |
- RBAC
  - Secret management



# Decision-making for choosing the deployment platform



BANKING FINANCIAL

TECHNOLOGY

TELCO



**BBVA**

**AON**



**AMERICAN  
EXPRESS**



**Santander**



**IAG**  
Insurance  
Australia  
Group



**SONY**

**tomtom**

**Braintree**  
a PayPal service



**RSA**



**SAMSUNG**

**Postmates**



**NOKIA**

**DELL EMC**



**OPTUS**

**T Systems**

**VONAGE**



**vodafone**

**Telstra**

**verizon**

# Kubernetes timeline

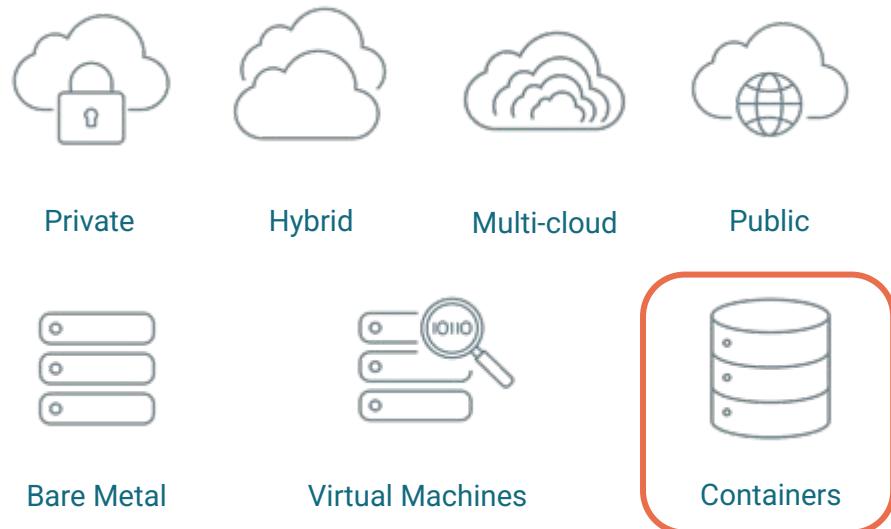
- 2014, June: Google open sources Kubernetes
- 2015, July: Version 1.0 is released
- 2015, July: Google and Linux Foundation start the CNCF
- 2016, November: The operator pattern is introduced in a blog post
- 2018, August: The Community takes the lead
- 2019, April: Version 1.14 introduces **Local Persistent Volumes**
- 2019, August: EDB team starts the Kubernetes initiative
- 2020, June: we publish this blog about benchmarking local PVs on bare metal
- 2020, June: Data on Kubernetes Community founded
- 2021, February: EDB Cloud Native Postgres (CNP) 1.0 released
- 2022, May: **EDB donates CNP** and open sources it under CloudNativePG
- 2025, January: CloudNativePG was recognized as an official **#CNCF** project



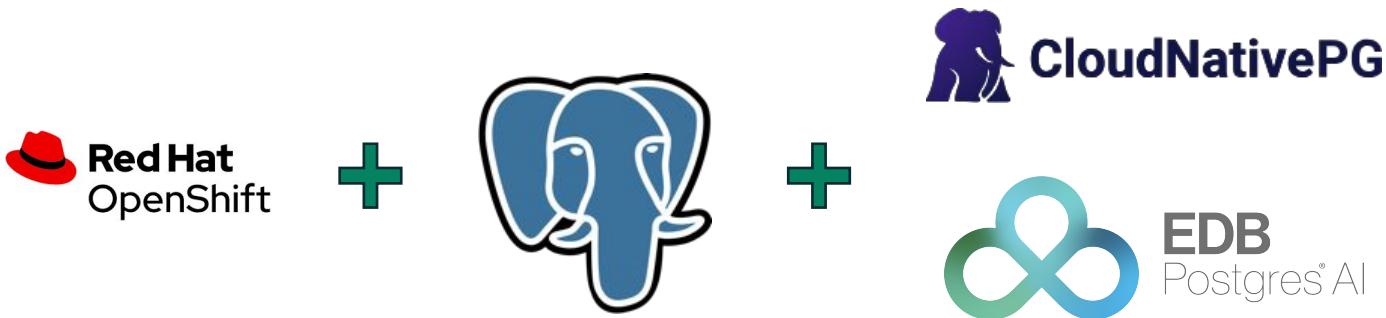
# Enabling the same PostgreSQL everywhere

From self-managed to fully managed DBaaS in the Cloud

- Same applications
- Faster innovation
- Performance and scalability
- Stability, security and control
- Seamless integration
- Obsolescence



# Win Technology



## Autopilot

It automates the steps that a human operator would do to deploy and to manage a Postgres database inside Kubernetes, including automated failover.



## Security

A grayscale photograph of a security guard from behind. The guard is wearing a light-colored jacket with the word "SECURITY" printed in large, bold, white capital letters on the back. He is holding a dark walkie-talkie up to his ear with his right hand. The background is a plain, light-colored wall.

CloudNativePG is secured by default.





It doesn't rely on statefulsets and uses its own way to manage persistent volume claims where the PGDATA is stored.

## Data persistence



## Designed for Kubernetes

It's entirely declarative, and directly integrates with the Kubernetes API server to update the state of the cluster — for this reason, it does not require an external failover management tool.



# Decision-making for choosing the deployment platform



# When to choose Kubernetes over VMs?

**01** | Cloud Native Applications that already run in Kubernetes

**02** | Scalable, replicated databases

**03** | Applications requiring automated failover and self-healing

**04** | Teams skilled in Kubernetes who want a unified infrastructure

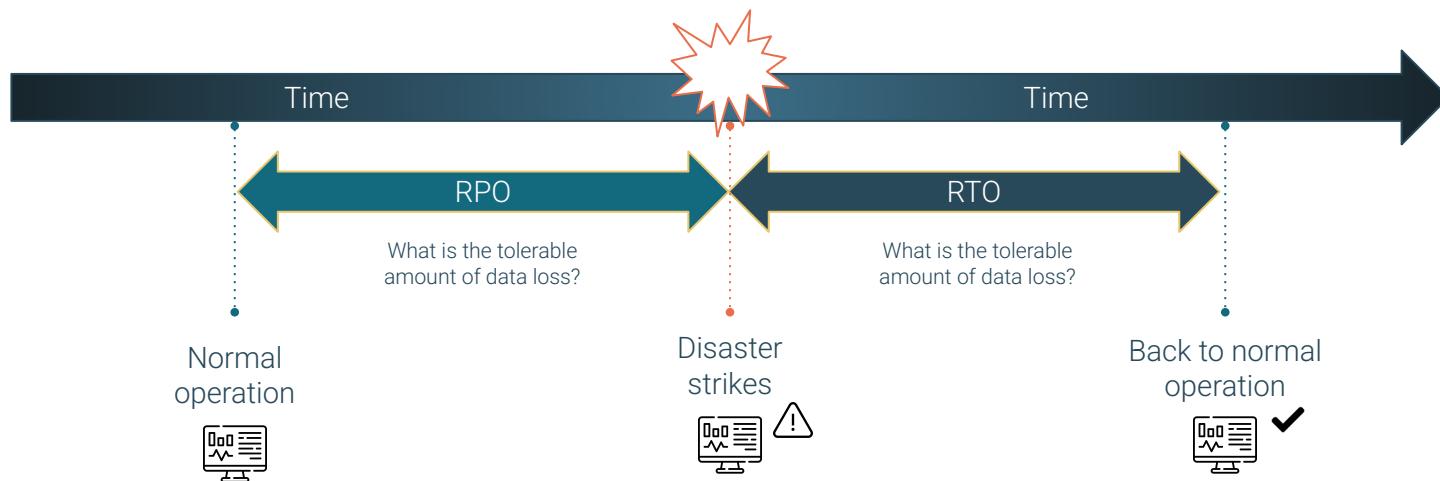


# Architectures



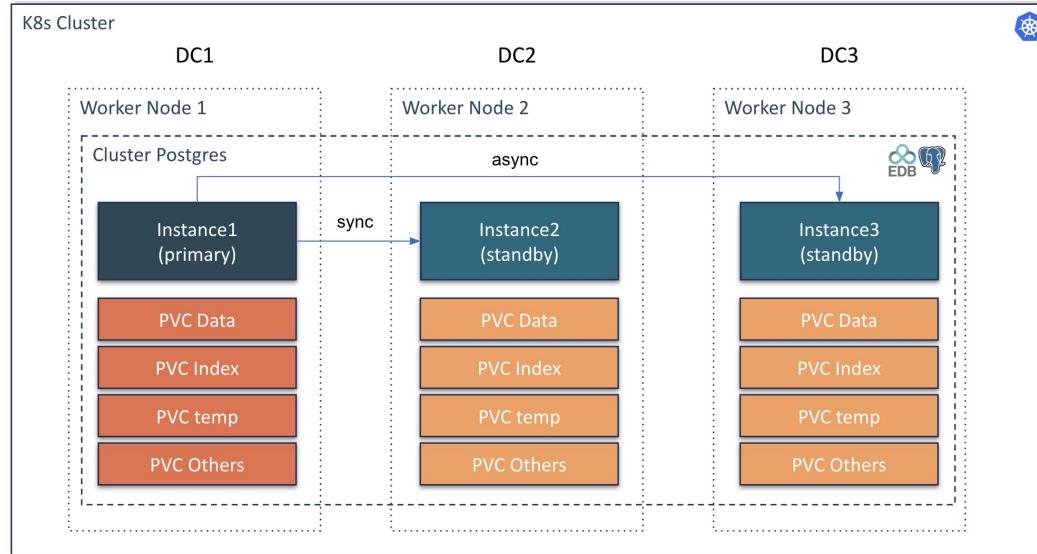
# Concepts

- Recovery Point Objective (**RPO**) and Recovery Time Objective (**RTO**) are key concepts in disaster recovery and business continuity planning, particularly related to data loss and system downtime.



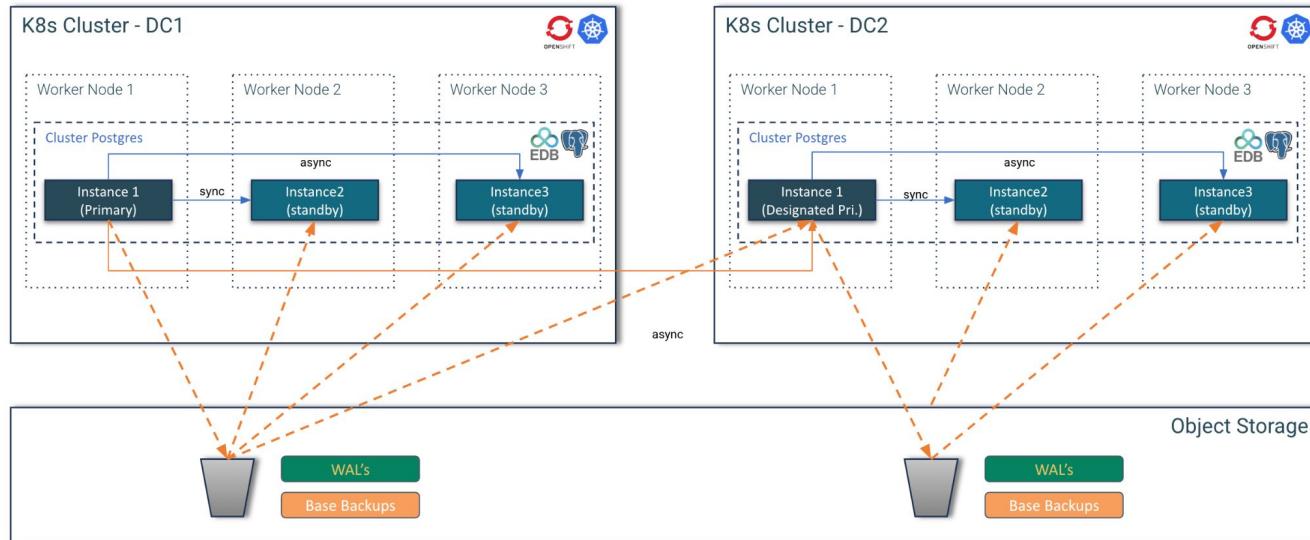
# Red Hat Recommendation

Red Hat recommend stretched clusters ONLY when latencies don't exceed 5 milliseconds (ms) round-trip time (RTT) between the nodes in different locations, with a maximum RTT of 10 ms.

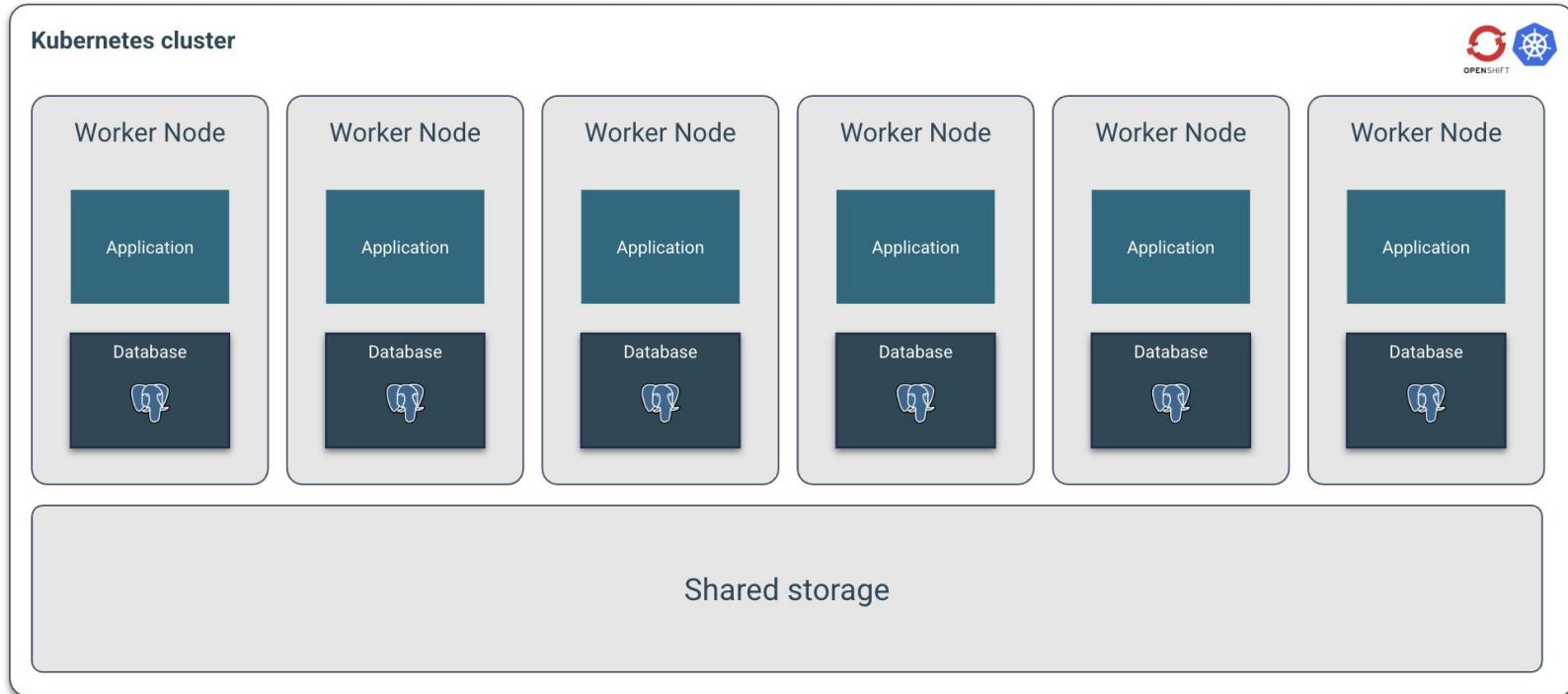


# Two separate single data center Kubernetes clusters

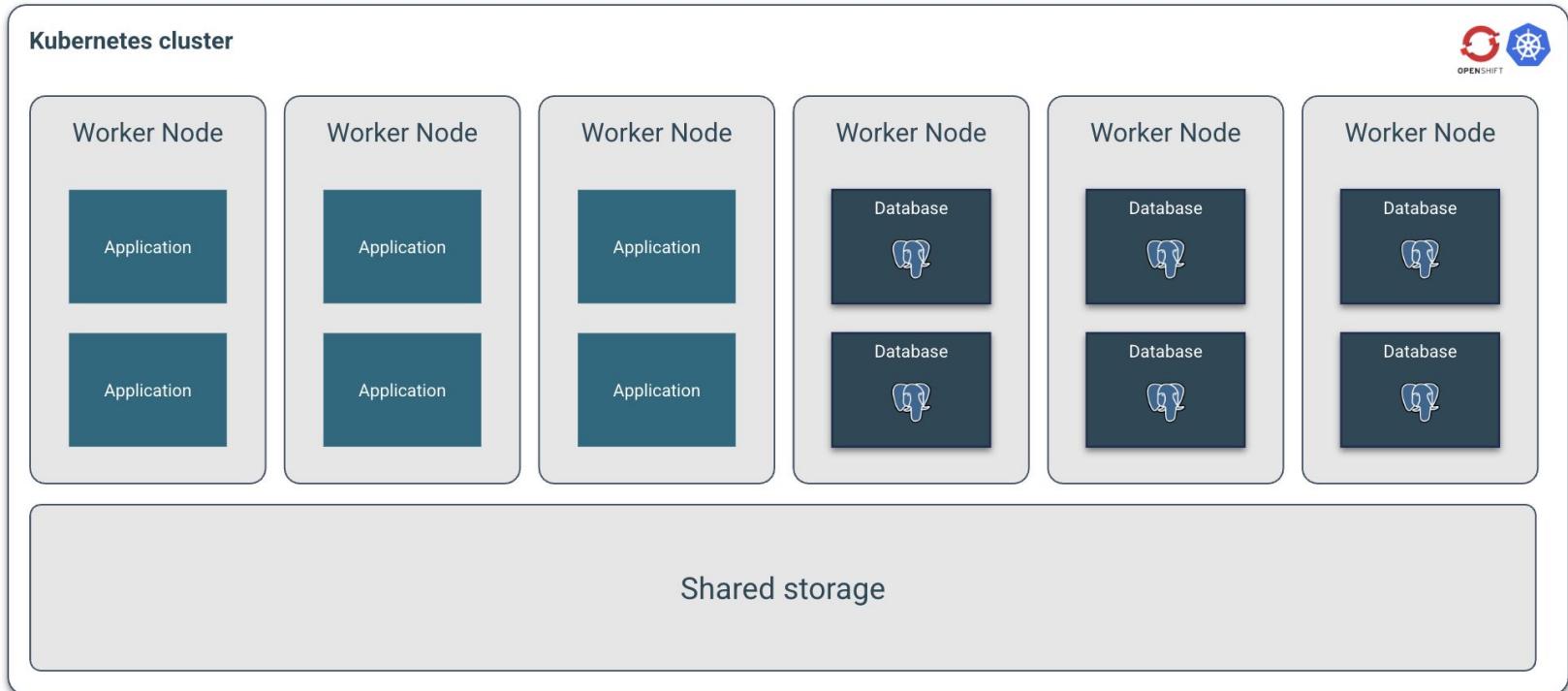
In case you cannot go beyond two data centers and you end up with two separate Kubernetes clusters, don't despair.



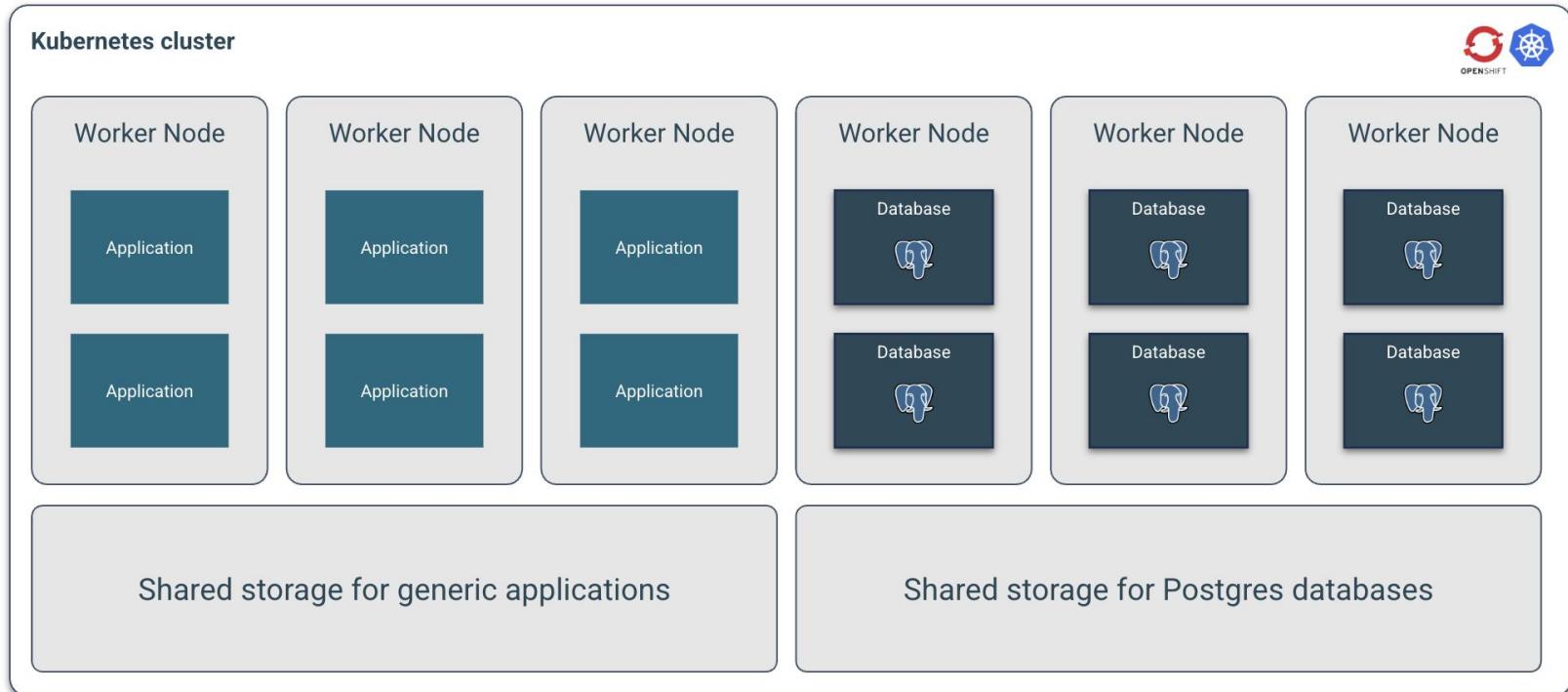
# Shared workload, shared storage



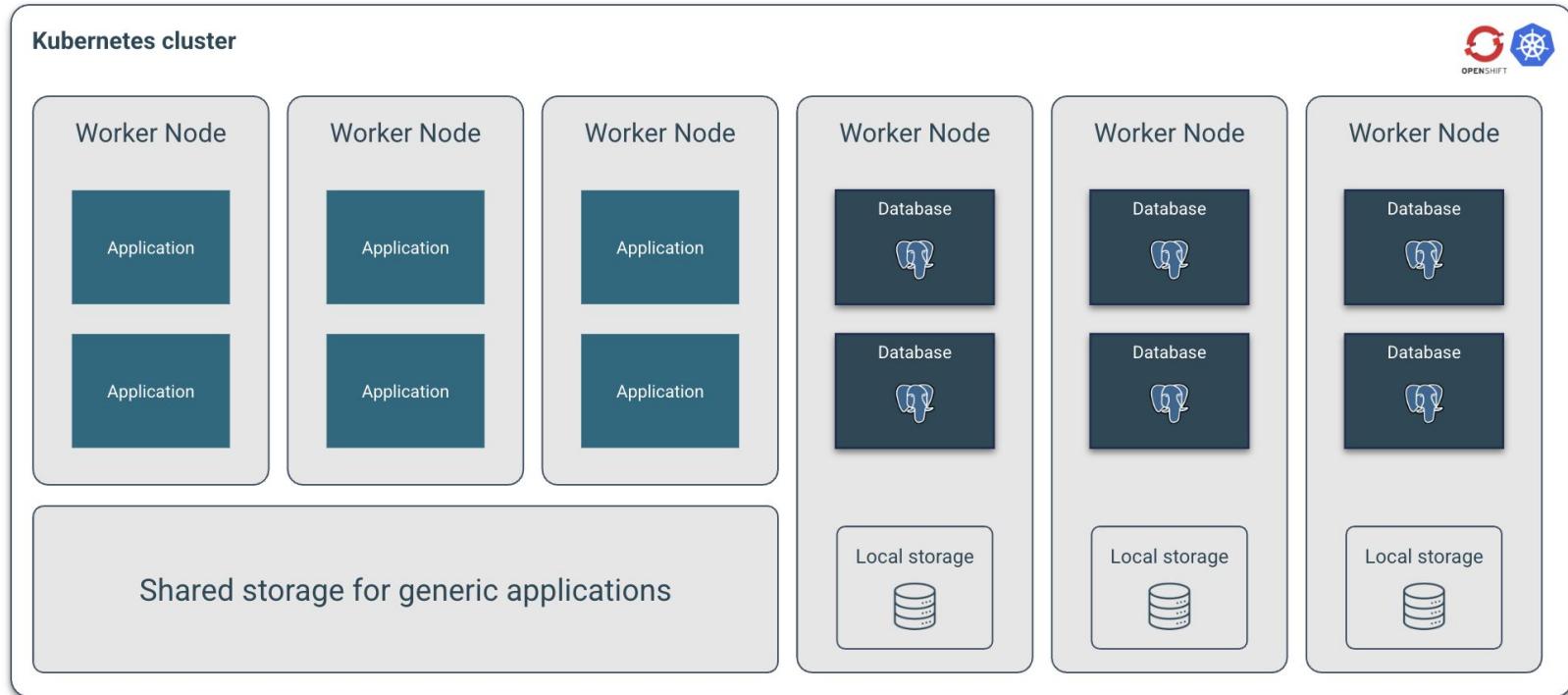
# Shared workload, shared storage



# Shared workload, shared storage



# Shared workloads, local storage



# Recommended architectures

<https://www.cncf.io/blog/2023/09/29/recommended-architectures-for-postgresql-in-kubernetes/>



## Recommended architectures for PostgreSQL in Kubernetes

By Gabriele Bartolini

September 29, 2023

Member post by Gabriele Bartolini, VP of Cloud Native at EDB

"You can run databases on Kubernetes because it's fundamentally the same as running a database on a VM", [tweeted Kelsey Hightower just a few months ago](#). Quite the opposite from what the former Google engineer and advocate said back in 2018 on Twitter: "Kubernetes supports stateful workloads; I don't."



Kelsey Hightower  
@kelseyhightower

You can run databases on Kubernetes because it's fundamentally the same as running a database on a VM. The biggest challenge is understanding that rubbing Kubernetes on Postgres won't turn it into Cloud SQL. ■

Truth is that I agree with him now as much as I agreed with him back then. At that time, the holistic offering of storage capabilities in Kubernetes was still immature (local persistent volumes would become GA only the year after), the operator pattern – which in the meantime has proven to be crucial for stateful applications like databases – was yet to become widely accepted, and the [Data on Kubernetes Community](#) was more than two years away (second half of 2020).

Nowadays, the situation is completely different. And I am sure that many people who've worked hard in the last few years to bring stateful workloads in Kubernetes agree with me that Kelsey's recent powerful words will contribute to reversing the public perception and facilitate our mission – provided we keep doing great.



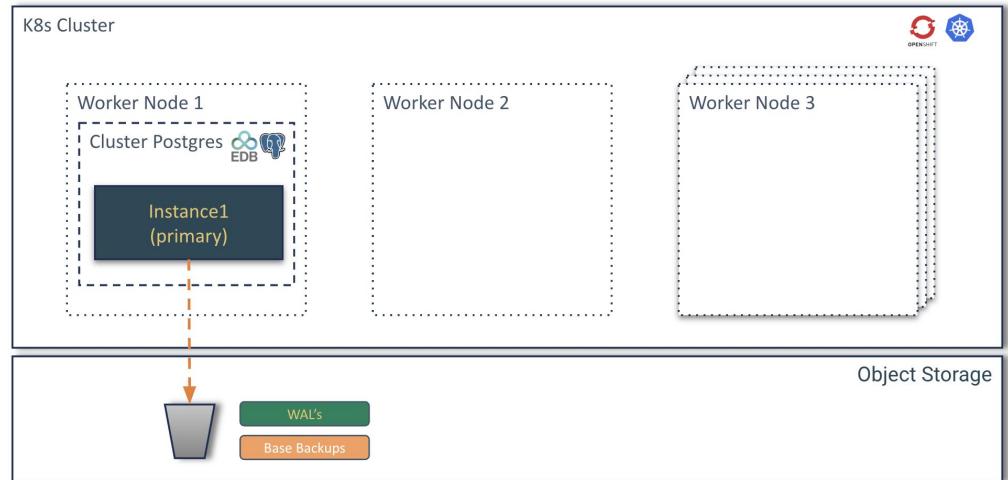
# Use Cases



# Use case 1 architecture

A single database is the simplest setup, involving one instance of a database server.

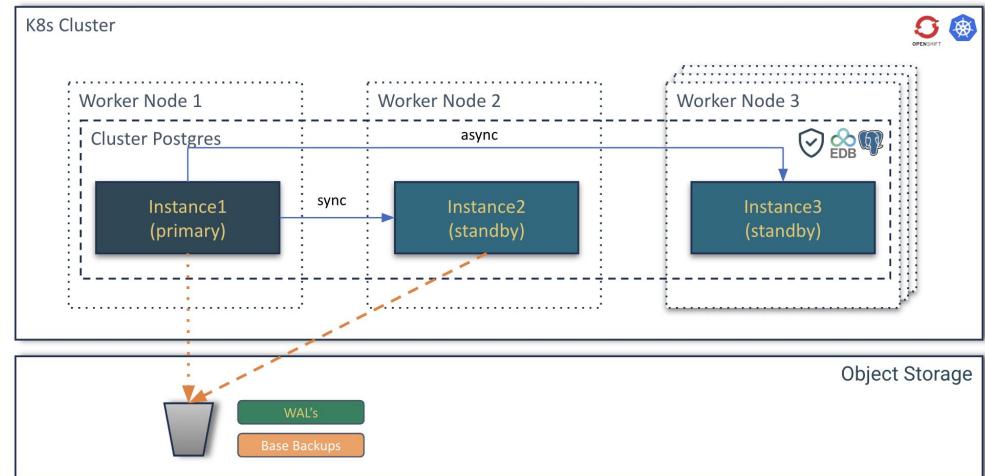
- Development and testing environments
- Small applications with low traffic
- Non-critical data analysis
- Applications with high tolerance for downtime
- Cost-sensitive projects



# Use case 2 architecture

An HA database setup aims to minimize downtime by having redundant components. If one component fails, another takes over automatically or with minimal intervention. This usually involves techniques like clustering, replication, or mirroring within the same data center or availability zone.

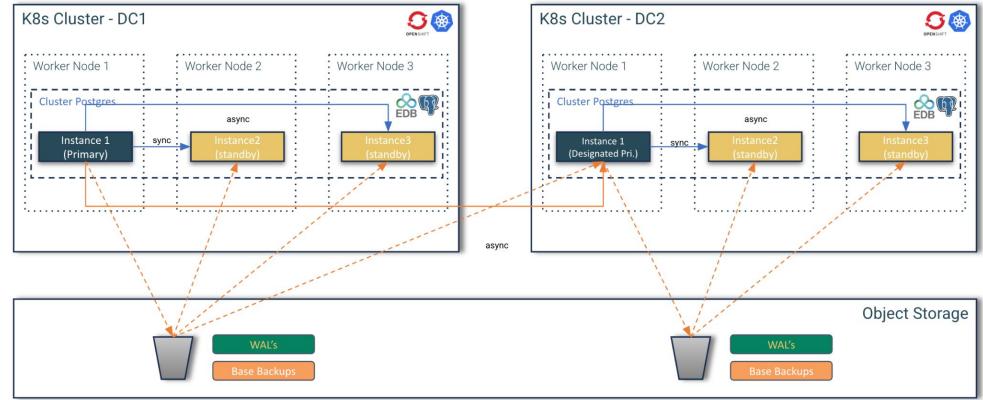
- Business critical Applications
- Applications with stringent SLAs
- Real-time systems
- Improving user experience
- Minimizing planned downtime



# Use case 3 architecture

A DR database setup focuses on protecting data and ensuring business continuity in the event of a large-scale disaster affecting an entire data center or region (e.g., natural disasters, power outages, cyberattacks). This typically involves replicating data to a geographically separate location.

- Regulatory compliance
- Protecting against catastrophic data loss
- Ensuring business continuity for mission-critical systems



# Use case 3 architecture

