When a Postgres Operator decision is not forever:

Approaches and Experiences for migrating between Postgres Operators

Takis Stathopoulos, PhD
Enterprise Architect @ Percona
PostgreSQL Conference Germany 2026
Essen, 22/04/2026

PERCONA

Your Speaker

25 years of experience designing and deploying open source software solutions in the enterprise:

- **Enterprise Architect @ Percona**
- **Solutions Architect @ EU Integrator**
- **Solutions Engineer @ Scientific Publishing SaaS Global Leader**
- **IT Infrastructure Architect and Team Leader @ National Hellenic Research Foundation**
- **Research Engineer @ National Technical University of Athens**

PERCONA®

Enterprise Architects / Solution Engineers / Architects / Infra Team Leader

Jack of all trades master of none ?

PERCONA®

Where is the value?

Realistic users scenarios

Compliance & NFR requirements

Available offerings

PERCONA®

Agenda for today

Postgres in K8s. Is there a trend?

Postgres Operators Landscape (Snapshot)

- **Options**
- **Key architectural, technological and licensing decisions**
- **Real life reasons for switching Operator technology**

Migration considerations and tooling

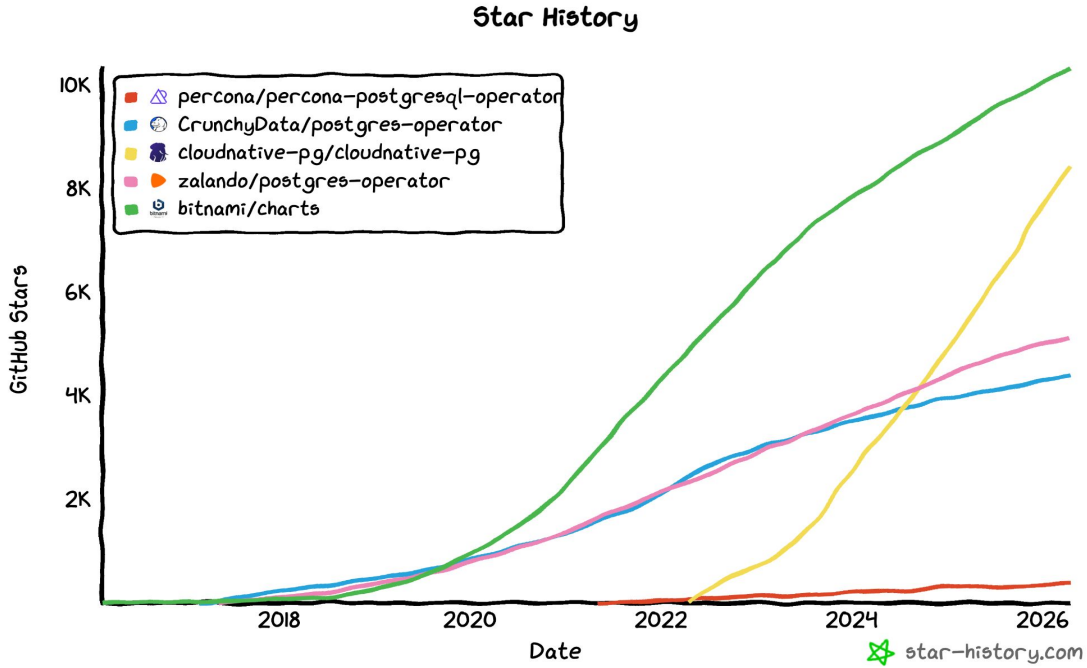
Deep dive on Operator migration case study

PERCONA®

Postgres in K8s
Is there a trend?

PERCONA®

Operators' Github Stars

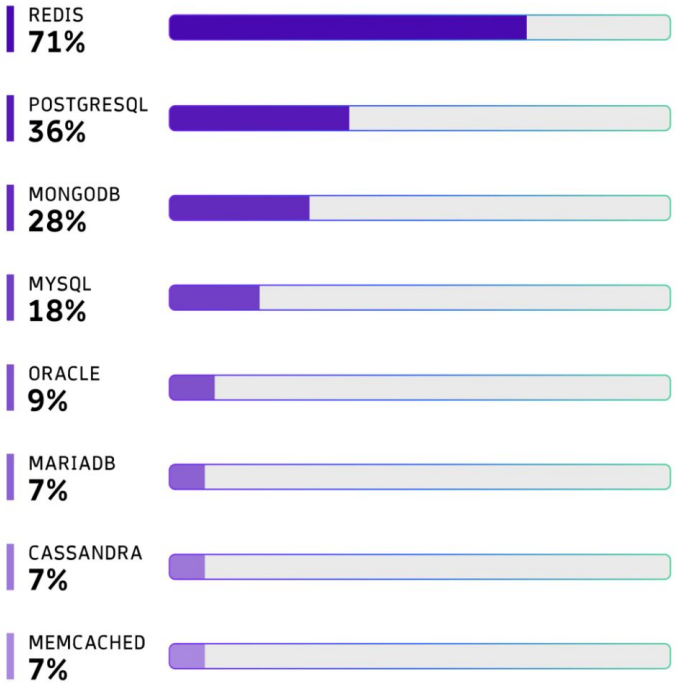


PERCONA

Kubernetes in the wild review

DATABASES

Redis is an in-memory key-value store and cache that streamlines processing, storage, and interaction with Kubernetes applications. Redis remains the clear leader in database and caching workload usage, and its prevalence has grown by 12 percentage points since 2022, with about 3 out of 4 organizations now running the technology in Kubernetes. Ranking second in database workloads, PostgreSQL saw a slight uptick in usage (up 6 percentage points from 2022).



- https://www.dynatrace.com/resources/ebooks/kubernetes-in-the-wild/

PERCONA

Qualitative experience

All significant Postgres discussions with enterprises right now have **either** a:

1. hyperscaler component, OR,
2. Postgres in Kubernetes component deployed, OR,
3. Postgres in Kubernetes component planned

PERCONA

Why Kubernetes?

Technical

Flexibility
Future proof
Open Source
Cloud's "*Operating System*"

Business

Cloud Independent
Cloud Hybrid
Digital Sovereighity
Commodity infrastructure

PERCONA®

When to consider K8s for Postgres

Good

Repeatable deployments
"As a service"
Apps in K8s already
Microservice applications

Bad

Huge main frame or monolith
"I heard k8s is good"
One DB, one app
No prior K8s experience

PERCONA®

Why Operators?

Operators map database primitives to k8s primitives

Day-1 simplified declarative step install, configure, etc.

Day-2 operations automated:

- Scaling
- Backups
- Updates
- Maintenance

Operator Installation

```
> git clone -b v2.8.2
https://github.com/percona/percona-postgresql-operator
cd percona-postgresql-operator

> kubectl apply --server-side -f deploy/crd.yaml

> kubectl apply -f deploy/rbac.yaml -n postgres-operator
```

Postgres cluster deployment

```
kubectl apply -f deploy/operator.yaml -n postgres-operator
```


PERCONA

Postgres Operators Landscape

PERCONA®

PERCONA®

Postgres is the land of opportunity!

Open vibrant ecosystem

Multiple options

More difficult selection

Postgres K8s Operators Landscape

1. Bitnami Postgres HA Helm Chart
2. Zalando PostgreSQL Operator
3. CloudNativePG
4. Crunchy Data PostgreSQL Operator
5. Percona Distribution for PostgreSQL Operator

Disclaimer: definitely there might be **ommisions** , not an exhaustive list!

Selection criteria: popularity, tech fundamentals

How are they different?

THEY ARE ALL POSTGRES

High availability:

(Replication)
Topology management
(Connection management)

Backup tooling, Monitoring

Specialized capabilities:

Extensions handling (e.g. PostGIS), OpenShift
Compliance: TDE, auditing, etc.

Operator Licensing

Container Images Licensing

Commercial Support

PERCONA®

Licensing

Operator Licensing != Container Image Licensing

PERCONA®

Bitnami Postgres Chart

Historic option. Is this an operator? Quite popular until recently.

https://github.com/bitnami/charts/tree/main/bitnami/postgresql

High Availability (Optional)

- Repmgr
- Pgpool II option

Backup:

- Venero.io (PV based)

License

- Apache 2.0

Monitoring

- Prometheus (Bitnami Operator)

Special capabilities:

- popular choice for deploying Postgres (until recently)
- Secure Images (proprietary)

Cons

- Simple(r) feature sets (e.g. HA, backup, etc.)

What is the catch?

PERCONA

A Curated Set of Packages for the Bitnami Community

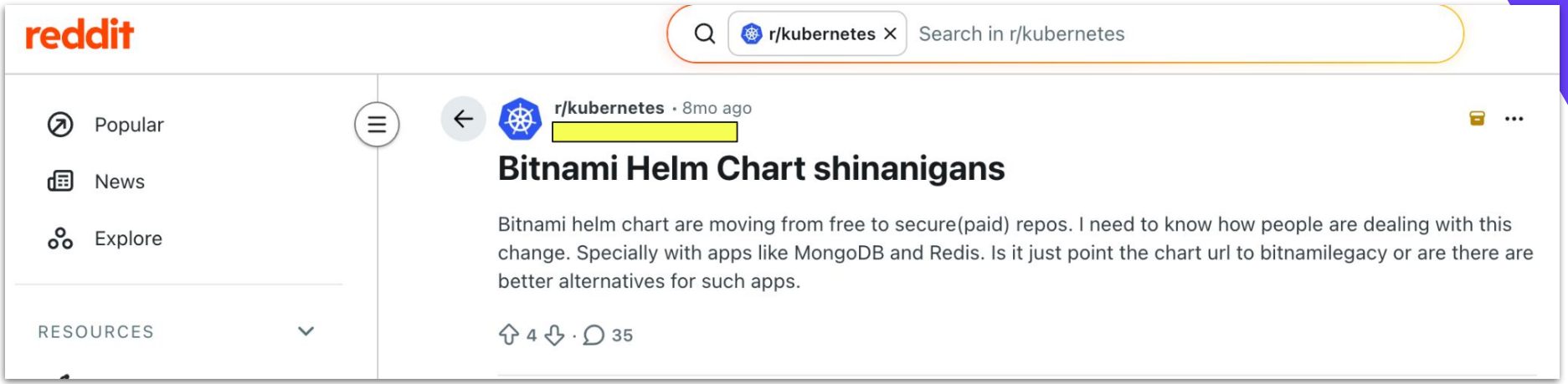
Alongside these security-first features in Bitnami Secure Images, Broadcom is also evolving the free, community Bitnami catalog toward the new hardened package composition. For the first time, the community will have access to more hardened, reduced-footprint images with fewer CVEs, greatly improving the security posture of all Bitnami users.

https://news.broadcom.com/app-dev/broadcom-introduces-bitnami-secure-images-for-production-ready-containerized-applications

To support this transition, beginning August 28th, 2025, the Bitnami team at Broadcom will begin to deprecate support for non-hardened Debian-based software images available in its free tier and gradually remove the non-latest images from its catalog. This will result in community access to a focused set of more hardened, more secure images. These free images are intended for development and only available on the “latest” tag. Additionally, Bitnami will not be changing its open source nature, and continues to make its source available under Apache 2.0 license.

When an operator is not forever

Risk #1: Operator or container images license changes



Note: Most Operators are build over a specific set of container images features

PERCONA

Cloud Native PG (CNPG)

cloudnative-pg.io

High Availability:

- Replication: Streaming Replication,
- Topology: K8s native/CNPG specific
- App connectivity: pgbouncer

Backup:

- CNPG-I plugins.
- **Barman** now available. Additional ones coming
- Volume Snapshot capabilities

Monitoring

- Prometheus

License

- CNCF sandbox project

Special features

- Very popular choice for deploying Postgres
- Kubernetes native and specific
- Replica Cluster (Disaster Recovery)

Proprietary options:

Downstream forks: Oracle compatibility, additional platform support and support options (E.g. IBM Power, OpenShift), TDE, additional clustering solutions

PERCONA

Zalando Postgres Operator

https://github.com/zalando/postgres-operator

High availability:

- Replication: Streaming Replication,
- Topology: Patroni
- Application connectivity: pgbouncer

Backup:

- pg_basebackup
- WAL-G or WAL-E via Spilo images

Monitoring

- Sidecar support

Licensing

- MIT License

Special Features

- Operator UI
- Logical backups
- Standby Cluster (Disaster Recovery)
- VM-compatible option

Commercial Support

Options available

PERCONA

PGO the Postgres Operator from Crunchy Data

https://github.com/crunchydata/postgres-operator

High availability:

- Replication: Streaming Replication,
- Topology: Patroni
- Application connectivity: pgbouncer

Backup:

- pgbackrest

Monitoring

- Operator monitoring stack: Pgmonitor, Grafana, etcv.

Licensing

- Apache License Version 2.0

Special Features

- Standby Cluster (Disaster Recovery)
- VM-compatible option

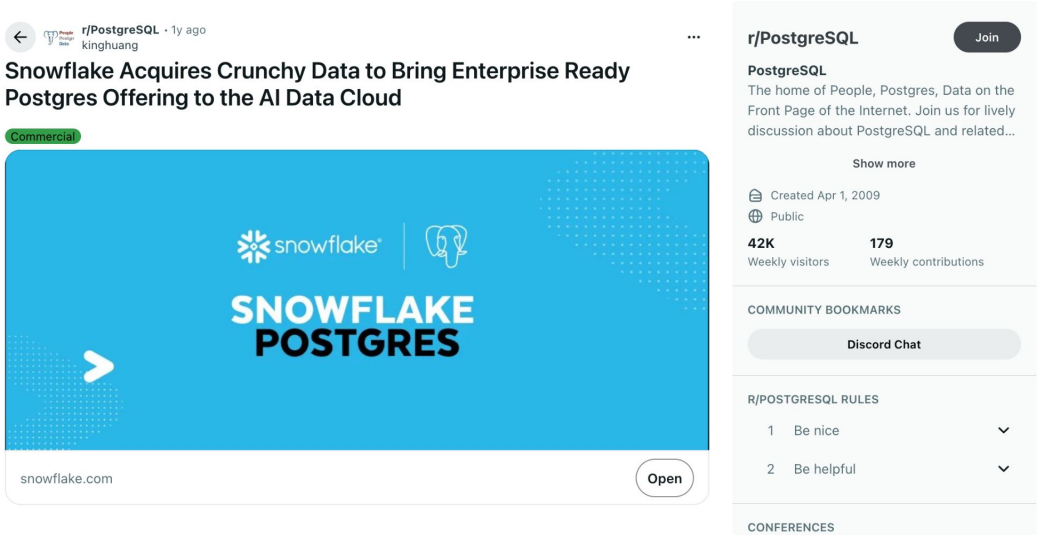
Hardened Postgres (TDE) as proprietary option

Crunchy Data Developer Program terms

PERCONA

When an operator is not forever

Risk #2: Change of vendor strategy



Users saying PGO will not be actively supported after Dec. 31, 2026

PERCONA

Percona Operator for Postgres

https://github.com/percona/percona-postgresql-operator

High availability:

- Replication: Streaming Replication,
- Topology: Patroni
- Application connectivity: pgbouncer

Backup:

- Pgbackrest
- PVC snapshots

Monitoring

- PMM

Licensing

- Apache License Version 2.0

Special Features

- PGO v5 hard fork
- Standby Cluster (Disaster Recovery)
- VM-compatible option
- Transparenet Database Encryption **pg_tde** open source option in next release(s)

Single type Open Source Container Images

PERCONA

When an operator is not forever

R(eality) #3: Environment and Requirements change

Regulatorly

- PCI v3 to PCI v4
- TDE requirements per security policies
- Faster CVE handling

Environemnt updates

- Hybrid and multi-technology environments
- Updated Cloud strategies
- Company-wide Open Source approaches

PERCONA

Options are available!

Downstream options available

- Proprietary or not

Multiple options available

- Different architectural pattenrs

Good for users and features!

PERCONA

PERCONA®

Migration projects parameters

Dataset

- Data size
- Data refresh rate

Cutover SLA

- Recovery Time Objective >= 0
- Recovery Point Objective = 0

Operational constraints

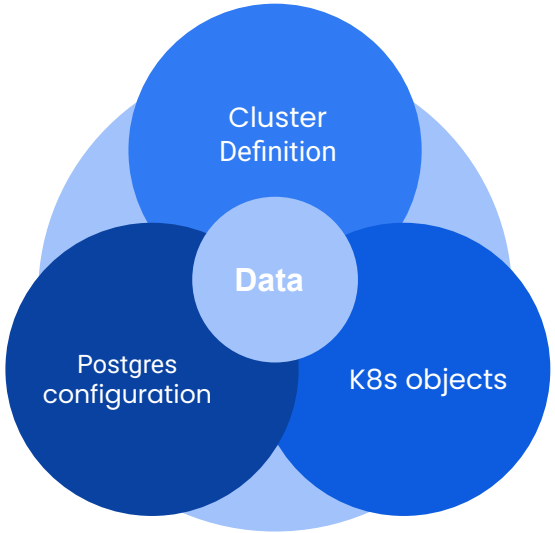
- Familiarity with source/target technologies
- Additional training, runbooks, etc
- Tooling
- Extensions

PERCONA

Migration project scope

Data migration is paramount but it's not the only one

Postgres Clusters Definitions	Custom per scenario / specific project
Postgres Configuration Parameters	Ideally declarative at the Operator level. Depends on technology/Operator
K8s artifacts, e.g. k8s secrets:	K8s artifacts, e.g. k8s secrets: depends on technology, can be very straightforward.
Postgres Data:	It's still Postgres!

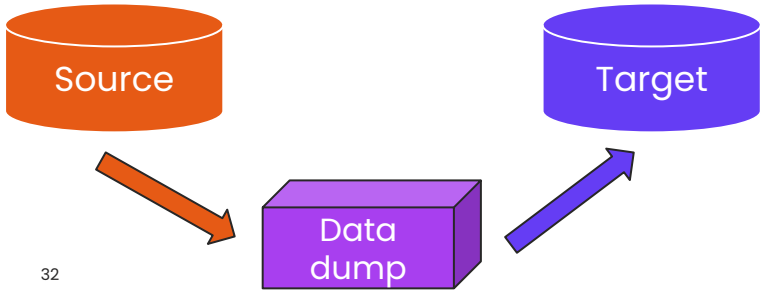


PERCONA

Migrate Postgres data between Operators

dump/restore

Postgres is still Postgres!



Pros

- Simple
- Works between any kind of operator

Cons

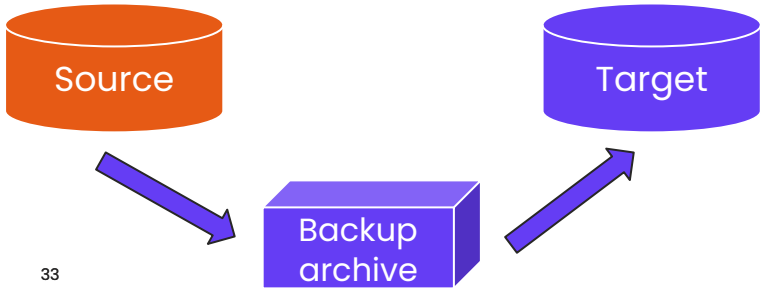
- Requires manual scripting
- Potentially slow
- Not suitable for large workloads

PERCONA

Migrate Postgres data between Operators

Physical Backup/Restore based

Depends on the backup/restore tool used, e.g. Barman, pgbackrest, WAL-E



Fast, specific combinations should work out of the box

Others might require more custom work

PERCONA

Migrate Postgres data between Operators

Replication based

Operators support Disaster Recovery Options

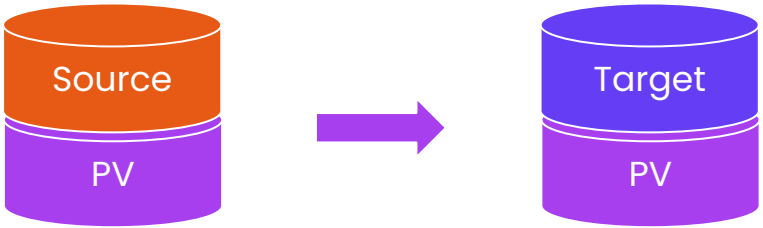
Zero downtime, easy, configuration depends on Operator DR approach of source/target cluster.

PERCONA

Migrate Postgres data between Operators

Physical Volumes based

Convert data from an existing Physical Volume to a new Physical Volume



Requires source/target technologies compatibility

Potential downtime during the cutover

Pros: minimal additional storage needed. Considered for large workloads.

PERCONA

MIGRATION SCENARIO CASE STUDY

PERCONA®

POC Environment

Inspired by Slava's Sarzhan blog post:

https://www.percona.com/blog/migrate-to-freedom-choosing-a-truly-open-source-postgresql-operator/

Source:

- **Crunchy Data PostgreSQL Kubernetes Operator:** v5.8.6
- **PostgreSQL:** 17

Target:

- **Percona PostgreSQL Kubernetes Operator: v2.8.0**
- **PostgreSQL: 17**

PERCONA

PERCONA®

Step 1: prepare

1. **Collect PGO pgbackrest configuration**

Set PGO behavior to namespace(s)-wide scope.
Mitigated to next Percona Operator version.

```
#PGO configuration
backups:
    pgbackrest:
      configuration:
        - secret:
            name: pgo-s3-creds
      global:
        repo1-path: /pgo-migration-testing/crunchydata

      repos:
        - name: repo1
          s3:
            bucket: pgo-migration-testing
            endpoint: s3.amazonaws.com
            region:  us-east-1
          schedules:
            full: "0 0 * * 0"
```


PERCONA

Step 2: install and bootstrap from pgbackrest backup

1. **Install Percona Operator and Cluster.**
2. **Bootstrap from PGO's pgbackrest backup (set `dataSource`)**
3. **Define new target for Percona Backup (avoid conflicts)**

Simple and quick.

Items to note: During installation set behavior to namespace(s)-wide scope. Next Percona Operator versions will mitigate this requirement. Target/Source Operator versions awareness.

```
#Install Target Cluster
kubectl apply --server-side -f
https://raw.githubusercontent.com/percona/percona-p
ostgresql-operator/v2.8.0/deploy/bundle.yaml -n
percona-postgres-operator
```

```
#Bootstrap from PGO pgbackrest backup
dataSource:
    pgbackrest:
      stanza: db
      configuration:
        - secret:
            name: pgo-s3-creds
      global:
        repo1-path: /pg-operator-testing/crunchydata
      repo:
        name: repo1
        s3:
          bucket: pg-operator-testing
          endpoint: s3.amazonaws.com
          region:  us-east-1
```

PERCONA®

Step 1: prepare

1. Collect PGO replication configuration
    - Collect PGO backup configuration (optionally) as previously for initial sync
2. Make sure Target Cluster has access to Source Cluster replication
    - Collect replication configuration

```
#Source cluster replication endpoint,e.g.LoadBalancer IP

kubectl get service source-cluster-ha -o
jsonpath='{.status.loadBalancer.ingress[0].ip}:{.spec.por
ts[0].port}{"\n"}' -n cpgo
34.27.90.225:5432
```

```
#export replication certificates

kubectl get secret source-cluster-cluster-cert -o json -n
cpgo | \
yq '{"apiVersion": .apiVersion, "kind": .kind, "data":
.data, "metadata": {"name": .metadata.name}, "type":
.type}' -o yaml > ~/source-cluster-cluster-cert.yaml


kubectl get secret source-cluster-replication-cert -o
json -n cpgo | \
yq '{"apiVersion": .apiVersion, "kind": .kind, "data":
.data, "metadata": {"name": .metadata.name}, "type":
.type}' -o yaml > ~/source-cluster-replication-cert.yaml
```

Step 2: deploy

1. Same initial steps as the pgbackrest based method
2. Import replication certificates to target cluster namespace
3. Import replication secrets to Target Cluster configuration
4. Set the target cluster as standby
5. Once replication caught up, make the
   - Source standby
   - Promote target

```
#Import replication certificates

kubectl apply -f ~/source-cluster-cluster-cert.yaml -n
percona-postgres-operator
kubectl apply -f ~/source-cluster-replication-cert.yaml
-n percona-postgres-operator
```

```
#Import replication secrets in target configuration
  secrets:
    customReplicationTLSSecret:
      name: source-cluster-replication-cert
    customTLSSecret:
      name: source-cluster-cluster-cert
#Configure target cluster as source standbay
  standby:
    enabled: true
    # Public IP of source-cluster for “Streaming replication”
    host: 34.27.90.225
    # PostgreSQL port of source-cluster for “Streaming replication”
    port: 5432
    # AWS pgBackrest repo, which is used by
source-cluster
    repoName: repo1
```

PERCONA®

Step 1: retain source cluster volume

1. Change retention policy after cluster deletion: set to retain
2. Delete the source cluster
3. Install Target Operator
4. Label source cluster PV

```
#Retain the source cluster volume
kubectl patch pv pvc-a9891ba9-d2f7-4d12-a6ef-a3051e0f89db
-n cpgo -p
'{"spec":{"persistentVolumeReclaimPolicy":"Retain"}}'
```

```
#Delete source cluster and operator
kubectl delete postgrescluster source-cluster -n cpgo
kubectl delete -k kustomize/install/default
```

```
#Label PV
kubectl label pv pvc-a9891ba9-d2f7-4d12-a6ef-a3051e0f89db
pgo-postgres-cluster=percona-postgres-operator-cluster
```


PERCONA

Step 2: install Target Cluster

1. Reference retained volume in cr.yaml of target cluster
2. Install the cluster

Pros: reuses the storage
Cons: depends on source/target versions, potentially more risky

Needs testing!

```
#target cluster cr.yaml snippet
dataVolumeClaimSpec:
      accessModes:
      - ReadWriteOnce
      selector:
        matchLabels:
          pgo-postgres-cluster:
percona-postgres-operator-cluster
      resources:
        requests:
          storage: 10Gi
```


PERCONA

Closing remarks

PERCONA®

Key takeaways

1. Changes are bound to happen
2. Switching Operator technologies is not a corner case
3. The good news: It's still Postgres!
4. But it's still a migration project: Proper Planning, Testing, Cutover

PERCONA


Danke schön
https://www.linkedin.com/in/pgstathopoulos/
SCAN ME

PERCONA