EDB
Postgres® for the AI Generation

Demystifying K8s for Postgres DBAs
A Guide to Operators

Adam Wright
Product Manager
October 24, 2024

Senior Product Manager

adam.wright@enterprisedb.com

Former Database Consultant

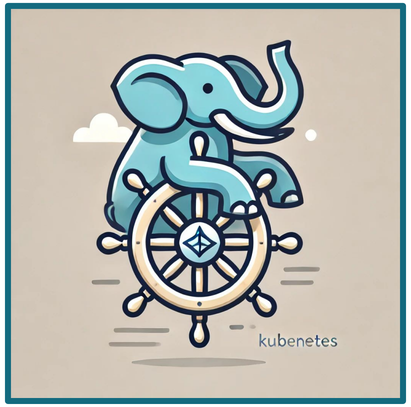
https://postgresql.life/post/adam_wright/

Agenda

- Introductions to K8s and Operators
- What does a Postgres DBA do all day?
- How does this change in K8s?
- What else should a DBA know about K8s? (time permitting)

K8s meet Postgres, Postgres meet K8s

Strong Open Source Communities



- PostgreSQL:
  - Permissive license (PostgreSQL License)
  - Governed by the PostgreSQL Global Development Group
  - Focus on vendor neutrality
  - 700+ contributors
- Kubernetes:
  - Apache 2.0 license (permissive)
  - Governed by Cloud Native Computing Foundation (CNCF)
  - Vendor-neutral governance
  - Thousands of contributors

Level Setting
K8s Basics

From Simple Namespace Containers to Orchestrated K8s

Containers

Packages the code for a software product along with the dependencies it needs at run time (i.e. when it is executed.).

Containers are managed in **Pods.**

Kubernetes

A **container management system** used for orchestrating software management and utilizing infrastructure more efficiently.

It makes deploying and managing containers at scale a reality.

Why Kubernetes?

K8s includes many system services needed for managing software

Services, Load Balancing, and Networking

Health checking

Storage management

Automated Scheduling

Scalability: scale-up/down

Rolling Deployments

Kubernetes benefits

Kubernetes & containers microservice-based architectures

Scalability & Reliability

Handle application needs efficiently

Portability

Avoid lock-in and take advantage of multi-cloud and hybrid-cloud strategies

Development Speed

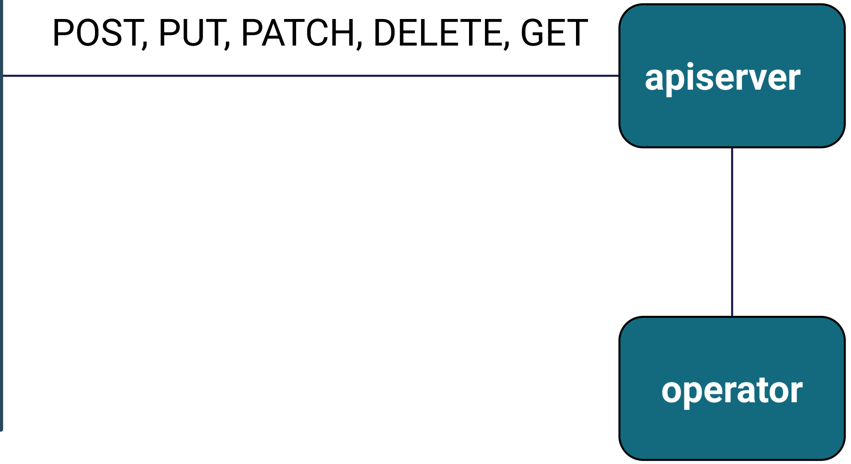
Embrace continuous delivery for a greater rate of innovation and competitiveness

What's a Kubernetes Operator?

Extends Kubernetes Controller and defines how a **complex** application works

- A kubernetes Operator automates actions of a human being, in a programmatic way
- A PostgreSQL cluster is a **complex** application
  - Deployment and configuration
  - All Postgres nodes in a cluster **are different**
  - Failure detection and Failover
  - Updates and switchovers
  - Backup and Recovery

Describe your app to K8s



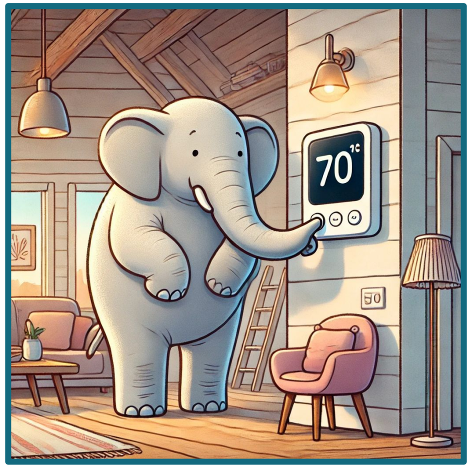
POST, PUT, PATCH, DELETE, GET

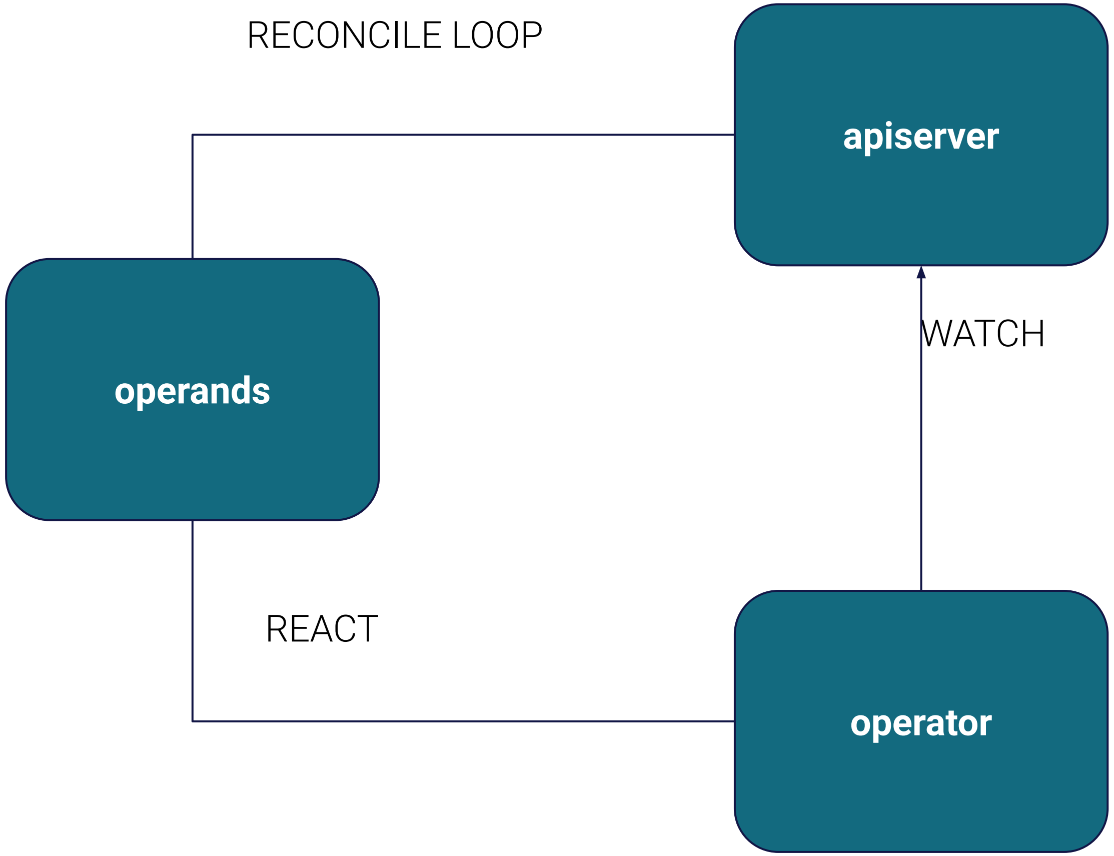
apiserver

operator

Reconciliation Loop

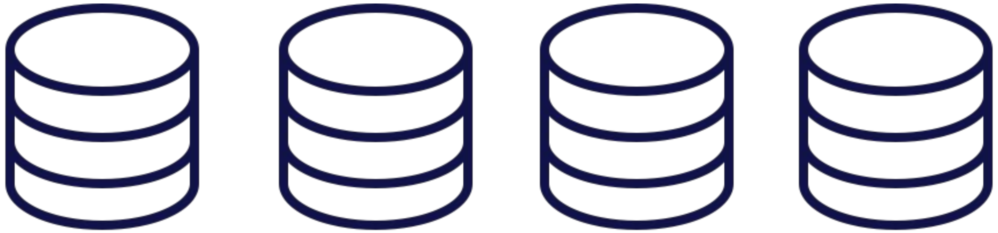
Foundation of Self-Healing



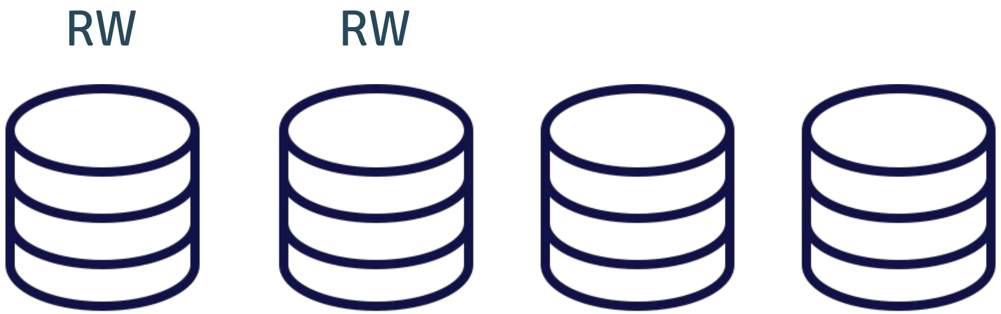


The Reconciliation Loop in Action

Desired state



Actual state



Level Setting
Postgres Operators

Operatorpalooza

Find your perfect Postgres match (order is random)

CloudNativePG

Crunchy Data Postgres Operator

Zalando Postgres Operator

STACKGRES.io

PERCONA OPERATOR FOR POSTGRESQL

portworx® by Pure Storage

How to deploy an Operator?

Using manifests

- **CloudNativePG**

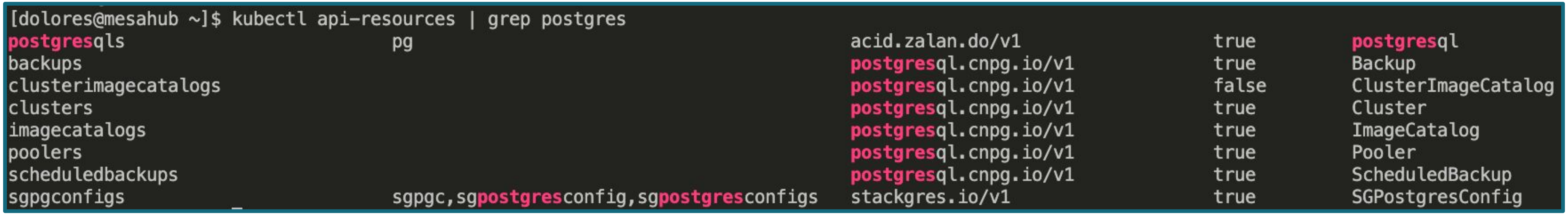
  -
    ```
    kubectl apply --server-side -f https://raw.githubusercontent.com/cloudnative-pg/cloudnative-pg/main/releases/cnpg-1.24.1.yaml
    ```
- **Stackgres**

  -
    ```
    kubectl create -f https://stackgres.io/downloads/stackgres-k8s/stackgres/1.13.0/stackgres-operator-demo.yml
    ```
- **Zalando**

  -
    ```
    kubectl apply -k github.com/zalando/postgres-operator/manifests
    ```

Installed a Postgres Operator: What happened?

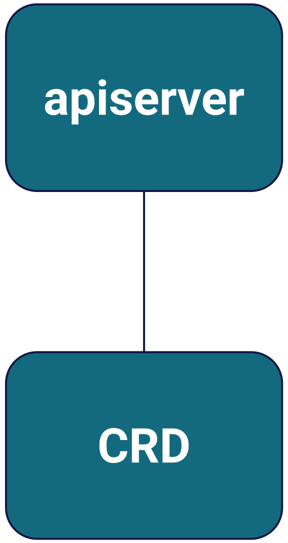
New K8s API Objects, Controller is Activated, Reconciliation Loops Begin



```
[dolores@mesahub ~]$ kubectl api-resources | grep postgres
postgresqls                         pg                                         acid.zalan.do/v1                  true         postgresql
backups                                                                        postgresql.cnpg.io/v1             true         Backup
clusterimagecatalogs                                                           postgresql.cnpg.io/v1             false        ClusterImageCatalog
clusters                                                                       postgresql.cnpg.io/v1             true         Cluster
imagecatalogs                                                                  postgresql.cnpg.io/v1             true         ImageCatalog
poolers                                                                        postgresql.cnpg.io/v1             true         Pooler
scheduledbackups                                                               postgresql.cnpg.io/v1             true         ScheduledBackup
sgpgconfigs                         sgpgc,sgpostgresconfig,sgpostgresconfigs   stackgres.io/v1                   true         SGPostgresConfig
```

Extend the K8s API with complex application logic

Operator CRD



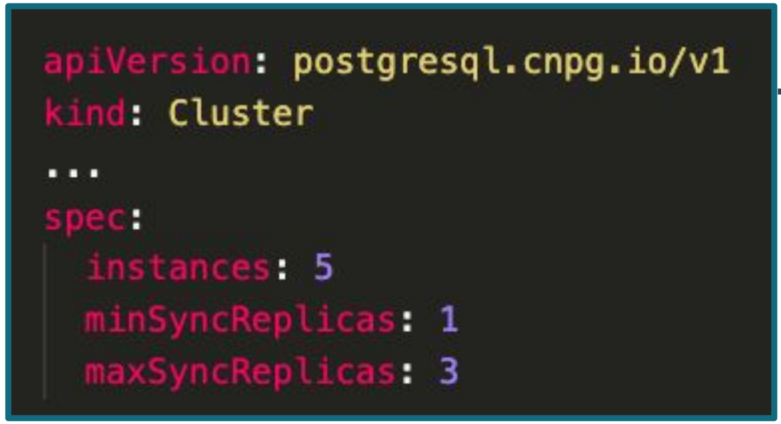
```
instances:
        description: Total number of instances in the cluster
        format: int32
        type: integer
maxSyncReplicas:
        default: 0
        description: The target value for the synchronous replication quorum, that can
        be decreased if the number of ready standbys is lower than this. Undefined or 0
        disables synchronous replication.
        format: int32
        minimum: 0
        type: integer
minSyncReplicas:
        default: 0
        description: Minimum number of instances required in synchronous replication
        with the primary. Undefined or 0 allow writes to complete when no standby is
        available.
        format: int32
        minimum: 0
        type: integer
```

Putting it all together

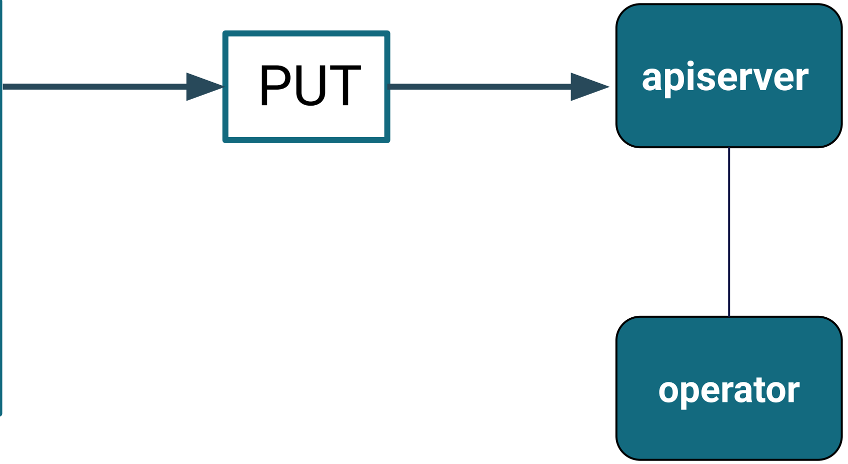
Example: Operator actions when Field is updated in the Cluster Spec

- Logger Initialization `Critical for debugging`
- Condition Validation `if cluster.Spec.MaxSyncReplicas > 0 && cluster.Spec.Instances < (cluster.Spec.MaxSyncReplicas+1) {`
- Deep Copy `origCluster := cluster.DeepCopy()`
- Syncing `cluster.Spec.Instances != cluster.Status.Instances`
- Patch Application

```
kubectl apply -f
```



```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
...
spec:
  instances: 5
  minSyncReplicas: 1
  maxSyncReplicas: 3
```



WARNING - no standard CRDs (**yet**)

CloudNativePG

Instances:
description:
Number of instances required in the cluster
type: integer
format: int32
default: 1
minimum: 1

Crunchy Data Postgres Operator

Replicas:
description: the number of cluster replicas to create for newly created clusters, typically users will scale up replicas on the pgo CLI command line but this global value can be set as well

Zalando postgres-operator

numberOfInstances:
total number of instances for a given cluster. The operator parameters max_instances and min_instances may also adjust this number. Required field.

Demo

Changes as a DBA

What does a DBA do all day?

https://www.techrepublic.com/article/what-does-a-dba-do-all-day/

- Installation, configuration, upgrade, and migration
- Database security
- Storage and capacity planning
- Performance monitoring and tuning
- Troubleshooting
- Backup and Recovery

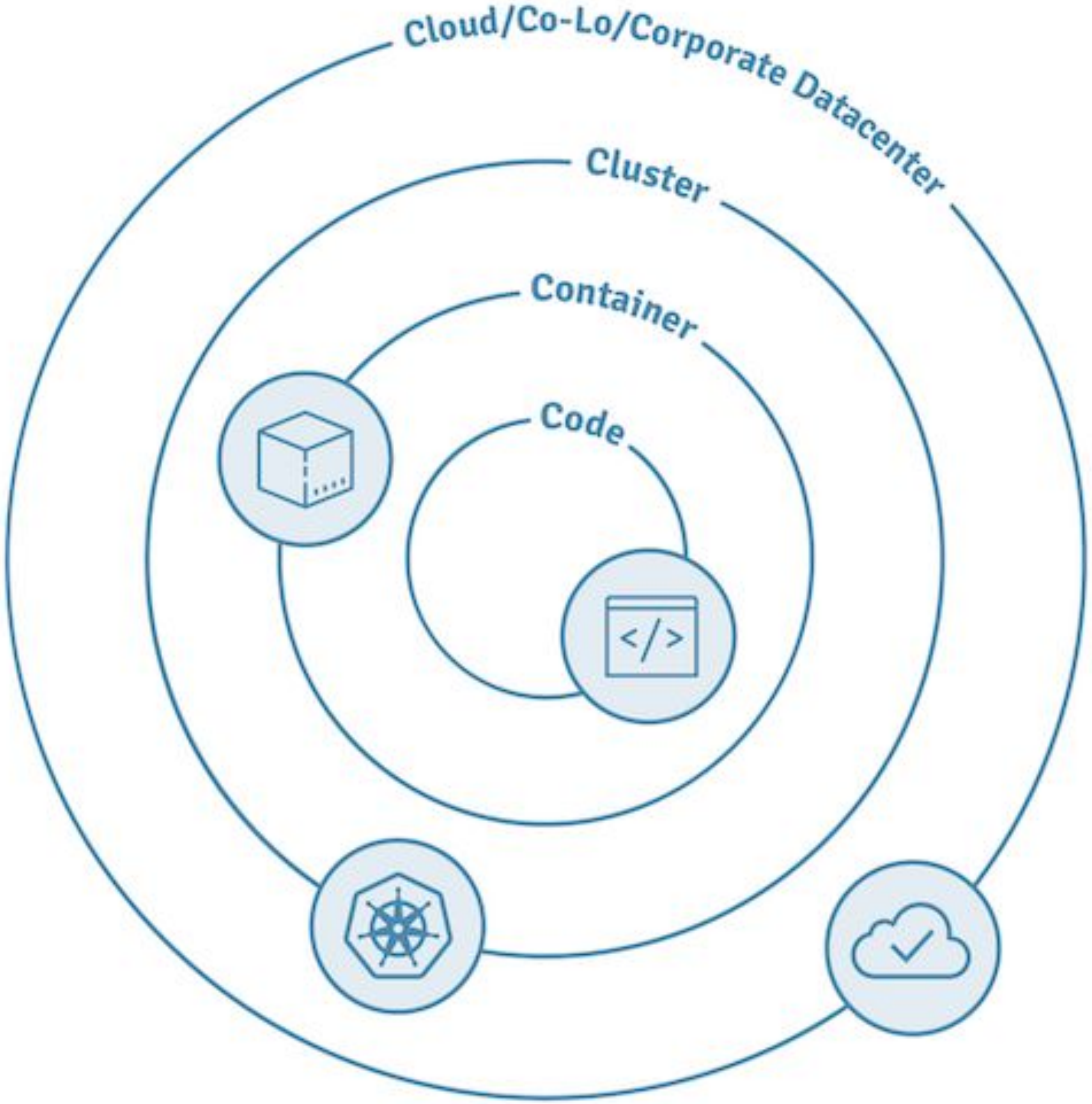
Installation, configuration, upgrade, and migration

- Installation - Declarative & Easy 🛠️
- Configuration - Declarative & Easy 🛠️
- Migration - Can be Easy…or Not 🤔
- Upgrade - Handle with Care ⚠️

Database Security

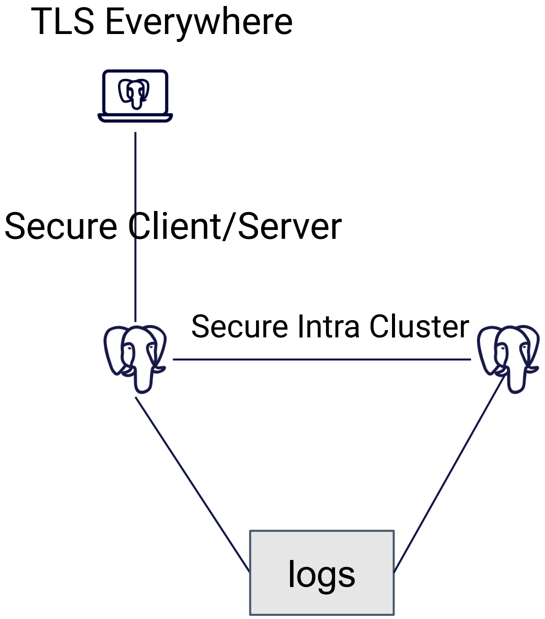
The 4 C’s Security Model in K8s

- The Cloud Layer
- The Cluster Layer
- The Container Layer
- The Code Layer



Database Security

A good operator is **secure by default**

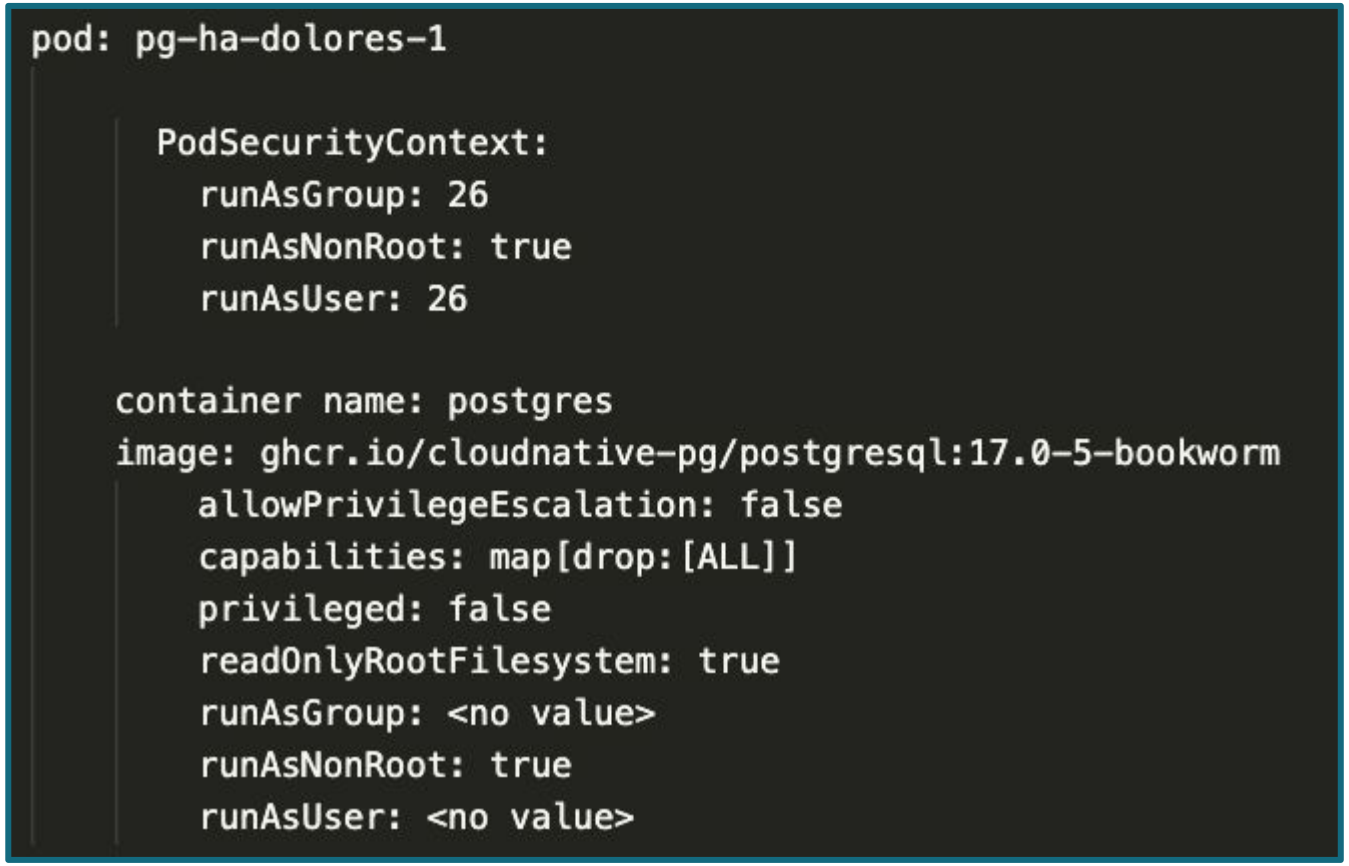


TLS Everywhere

Secure Client/Server

Secure Intra Cluster

logs

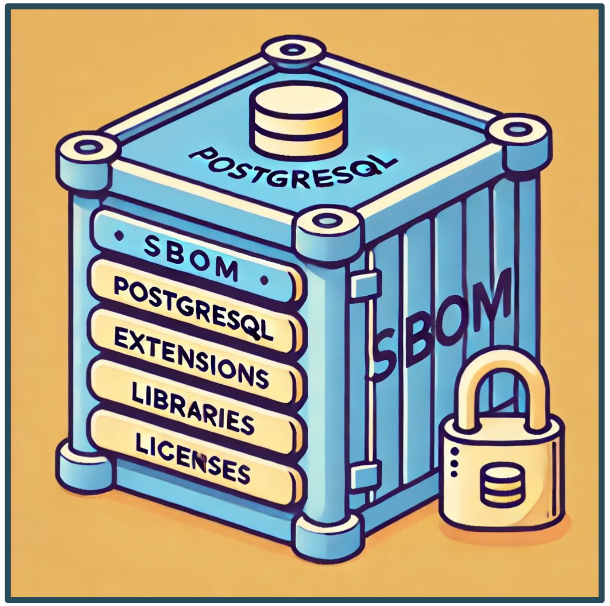


```
pod: pg-ha-dolores-1

   PodSecurityContext:
    runAsGroup: 26
    runAsNonRoot: true
    runAsUser: 26

  container name: postgres
  image: ghcr.io/cloudnative-pg/postgresql:17.0-5-bookworm
    allowPrivilegeEscalation: false
    capabilities: map[drop:[ALL]]
    privileged: false
    readOnlyRootFilesystem: true
    runAsGroup: <no value>
    runAsNonRoot: true
    runAsUser: <no value>
```

Software Bill of Materials (SBOM)

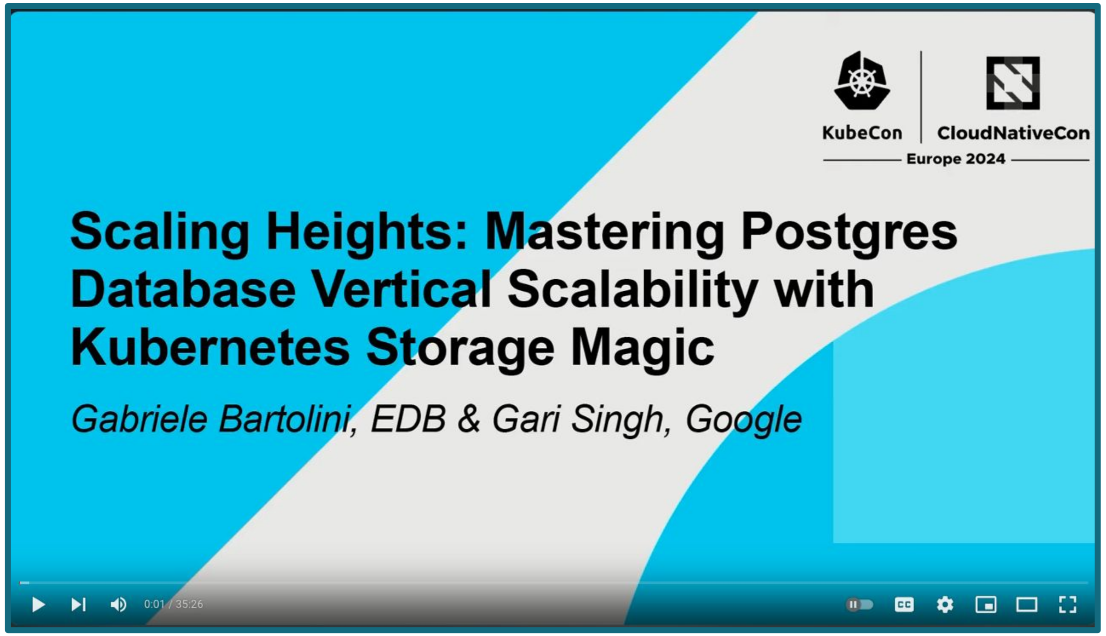
What’s in your container?

- Containers shouldn’t be black boxes
- SBOM provides a detailed list of software components
- Helps identify vulnerabilities (CVEs) and licensing issues
- Critical for using community/vendor images in enterprise environments
- Can be provided by container distributor or pulled in and scanned
- See the DoD IronBank Repository for pipeline example



Storage and capacity planning

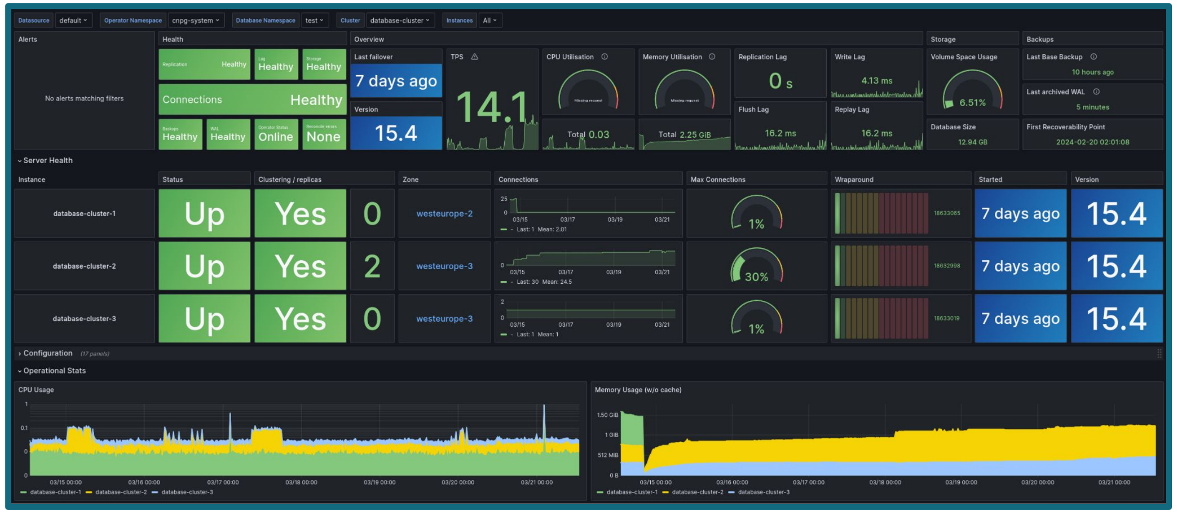
- Storage is still critical
    - All storage performance concepts for VM and Bare Metal apply in K8s
- Storage Classes and Persistent Volumes:
- Operators should support dynamic provisioning support for Persistent Volume Claims (PVC)
    - Automatic generation of PVC
    - Support for PVC templates
    - Reuse of storage for Pods in the same cluster
- Storage Types:
    - Local storage
    - Network storage



Scaling Heights: Mastering Postgres Database Vertical Scalability with Kubernetes Storage Magic

Gabriele Bartolini, EDB & Gari Singh, Google

Performance monitoring and tuning



Datasource: default | Operator Namespace: cnpg-system | Database Namespace: test | Cluster: database-cluster | Instances: All

Alerts: No alerts matching filters

Health: Replication Healthy · Lag Healthy · Storage Healthy · Connections Healthy · Backups Healthy · WAL Healthy · Operator Status Online · Reconcile errors None

Overview: Last failover 7 days ago · Version 15.4 · TPS 14.1 · CPU Utilisation Total 0.03 · Memory Utilisation Total 2.25 GiB · Replication Lag 0 s · Write Lag 4.13 ms · Flush Lag 16.2 ms · Replay Lag 16.2 ms

Storage: Volume Space Usage 6.51% · Database Size 12.94 GB

Backups: Last Base Backup 10 hours ago · Last archived WAL 5 minutes · First Recoverability Point 2024-02-20 02:01:08

Server Health

Instance	Status	Clustering / replicas		Zone	Connections	Max Connections	Wraparound	Started	Version
database-cluster-1	Up	Yes	0	westeurope-2	Last: 1 Mean: 2.01	1%	18633065	7 days ago	15.4
database-cluster-2	Up	Yes	2	westeurope-3	Last: 30 Mean: 24.5	30%	18632998	7 days ago	15.4
database-cluster-3	Up	Yes	0	westeurope-3	Last: 1 Mean: 1	1%	18633019	7 days ago	15.4

Configuration (17 panels)

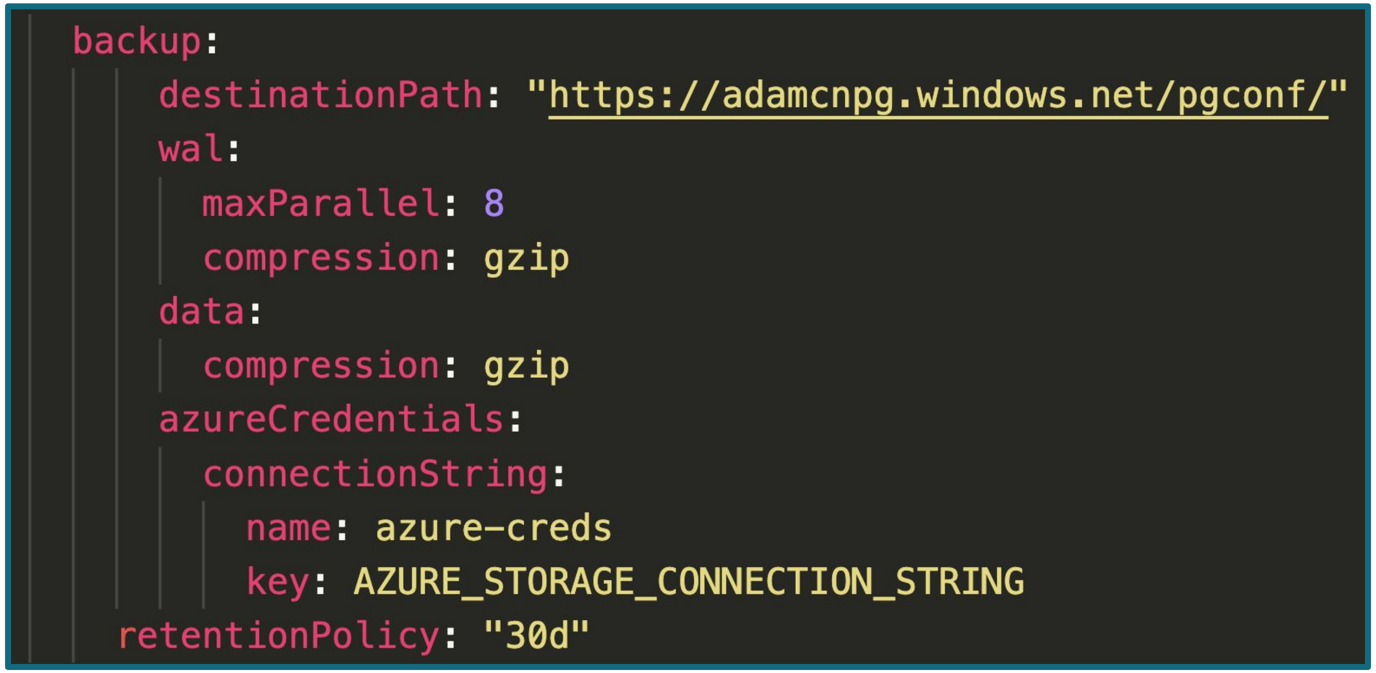
Operational Stats

- Benchmarking in K8s is critical
  - *Before* production
- Common stack for K8s workloads:
  - Prometheus and Grafana
- Enterprise monitoring solution?
  - Does it require an agent in the container
  - or K8s aware at the host level?
- Logging typically different in K8s:
  - Get logs off of containers
- pgBadger reports might still be an option
- `pg_stat_statements` and other extensions still valuable

Backup and Recovery

Operator Managed

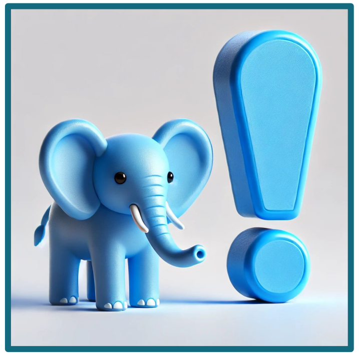
- No standard among K8s Operators
- Operators may offer a global view or be tightly coupled with a deployment
- **Diverse backup solutions:** Separate backup APIs and tools (e.g., core backup capabilities, Barman, pgBackRest, WAL-E, Volume Snapshots) are used
- **Varied capability levels:** Operators offer different features, such as:
  - Point-in-Time Recovery (PiTR)
  - Automated cluster creation from backups
  - Logical backups
  - Parallel WAL archiving
  - Cloud bucket checks
  - K8s Volume snapshots
- **Unmanageable at scale**: Cluster-by-cluster backups may not be practical at scale, leading to questions about their effectiveness.
- **Namespace-level backups:** Considering backups from an application perspective, rather than solely at the cluster level, may be beneficial at scale.



```yaml
backup:
  destinationPath: "https://adamcnpg.windows.net/pgconf/"
  wal:
    maxParallel: 8
    compression: gzip
  data:
    compression: gzip
  azureCredentials:
    connectionString:
      name: azure-creds
      key: AZURE_STORAGE_CONNECTION_STRING
 retentionPolicy: "30d"
```

Postgres DBA Specific…Extensions

Enable new workloads, But a Cost

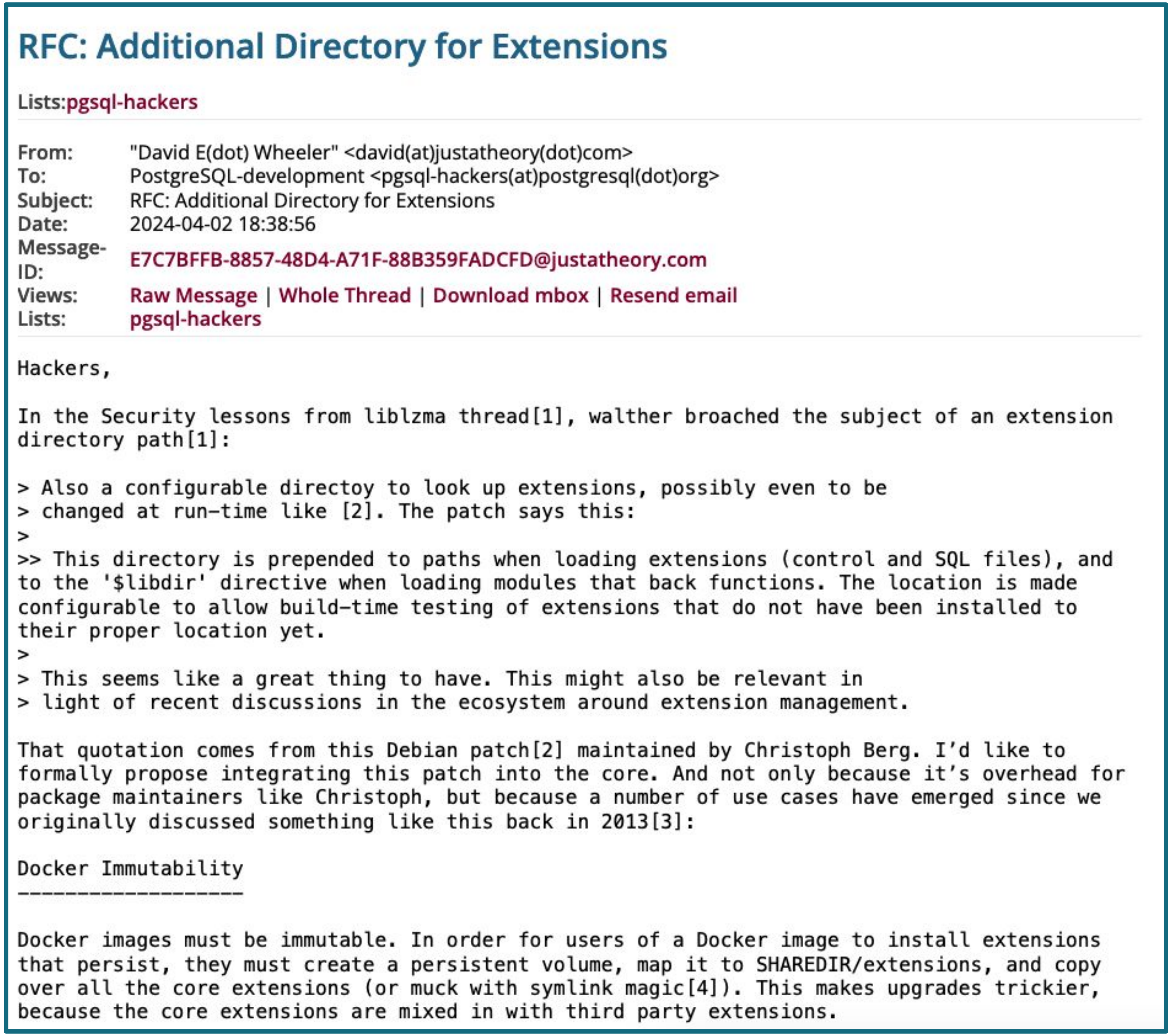


- **Dependency Bloat:** Each extension introduces new dependencies
  - Larger image sizes, more complexity
  - More frequent patches
- **Build Complexity:** Accommodating various combinations of extensions can make image building and management difficult

Future Hope

Extensions Search Path

- PostgreSQL can enable multiple directories with self-contained extensions
- Directories no longer tied to OS distribution
- Can be added on to the main immutable container image
- K8s - Currently in Alpha, Allow adding OCI artifacts (image+ portion of image)



RFC: Additional Directory for Extensions

Lists:pgsql-hackers

From: "David E(dot) Wheeler" <david(at)justatheory(dot)com>
To: PostgreSQL-development <pgsql-hackers(at)postgresql(dot)org>
Subject: RFC: Additional Directory for Extensions
Date: 2024-04-02 18:38:56
Message-ID: E7C7BFFB-8857-48D4-A71F-88B359FADCFD@justatheory.com
Views: Raw Message | Whole Thread | Download mbox | Resend email
Lists: pgsql-hackers

```
Hackers,

In the Security lessons from liblzma thread[1], walther broached the subject of an extension
directory path[1]:

> Also a configurable directoy to look up extensions, possibly even to be
> changed at run-time like [2]. The patch says this:
>
>> This directory is prepended to paths when loading extensions (control and SQL files), and
to the '$libdir' directive when loading modules that back functions. The location is made
configurable to allow build-time testing of extensions that do not have been installed to
their proper location yet.
>
> This seems like a great thing to have. This might also be relevant in
> light of recent discussions in the ecosystem around extension management.

That quotation comes from this Debian patch[2] maintained by Christoph Berg. I’d like to
formally propose integrating this patch into the core. And not only because it’s overhead for
package maintainers like Christoph, but because a number of use cases have emerged since we
originally discussed something like this back in 2013[3]:

Docker Immutability
-------------------

Docker images must be immutable. In order for users of a Docker image to install extensions
that persist, they must create a persistent volume, map it to SHAREDIR/extensions, and copy
over all the core extensions (or muck with symlink magic[4]). This makes upgrades trickier,
because the core extensions are mixed in with third party extensions.
```

Demo

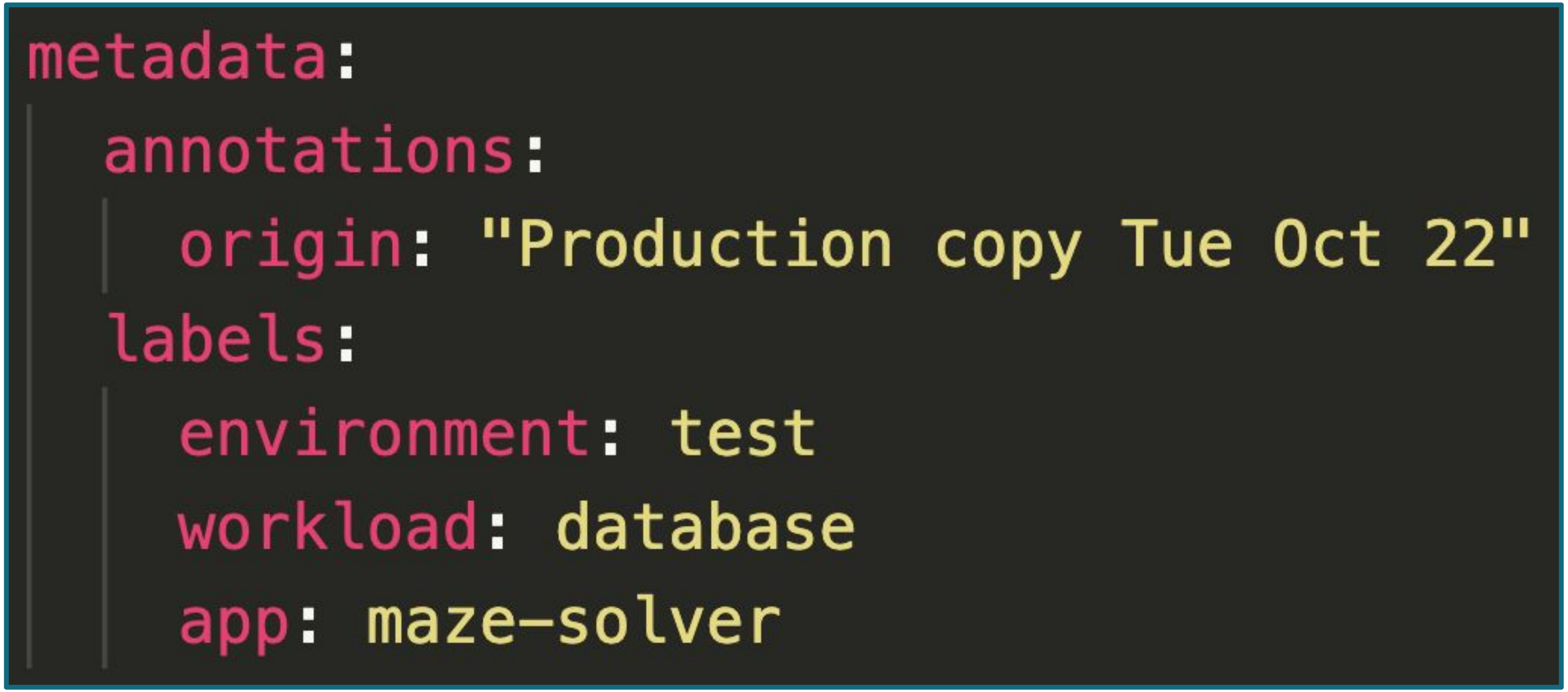
What else should a DBA know about K8s?

Labels and Annotations

- Resources in Kubernetes are organized in a flat structure
  - No hierarchies
  - No relationships
- Resources and objects can be linked through:
  - **Labels**: group objects to later **select** them
  - **Annotations**: add **non-identifying** information (not used for selection)
    - Useful for systems integration

Labels and Annotations

Default and Custom Labeling: Operator Capabilities Vary



```yaml
metadata:
  annotations:
    origin: "Production copy Tue Oct 22"
  labels:
    environment: test
    workload: database
    app: maze-solver
```

Kubernetes Worker Nodes Can Have Labels for Any Purpose

Combine with nodeSelector in your Spec to prefer or require

- **Define Node Roles**: `node-role.kubernetes.io/worker, node-role.kubernetes.io/database`
- **Hardware Specifications:** `hardware.l2cache=high, cpu-type=intel`
- **Location:** `topology.kubernetes.io/zone=us-east-1, region=us-east2`
- **Environment Context:** `environment=production, environment=uat`
- **Compliance and Security:** `compliance=PCI, security-level=high`
- **Custom Operational Needs:** `maintenance-window=night, preferred-workload=db`

K8s Networking for DBAs

Confusing at first, but if you are just kicking the tires…

- Expect lots of churn i.e., dynamic networks, no more entering in the IP address of your database server
- I’m evaluating the Operator, I deployed a Postgres cluster, give me some easy options to connect:
  - **Option A: Using local K8s cluster (e.g., KIND),**
    - `kubectl port-forward service/pg-ha-dolores-rw 5432:5432 > port-forward.log 2>&1 &`
    - `psql -h 127.0.0.1 -p 5432`
  - **Option B: NodePort**
    - `psql -h <Node-IP> -p 3007`
  - **Option C: Use ‘kubectl exec’** to open a shell session inside your Postgres container or run psql from inside the container
    - `kubectl exec -it pg-ha-dolores-1 --container postgres -- bash`
    - `kubectl exec -it pg-ha-dolores-1 --container postgres -- psql`

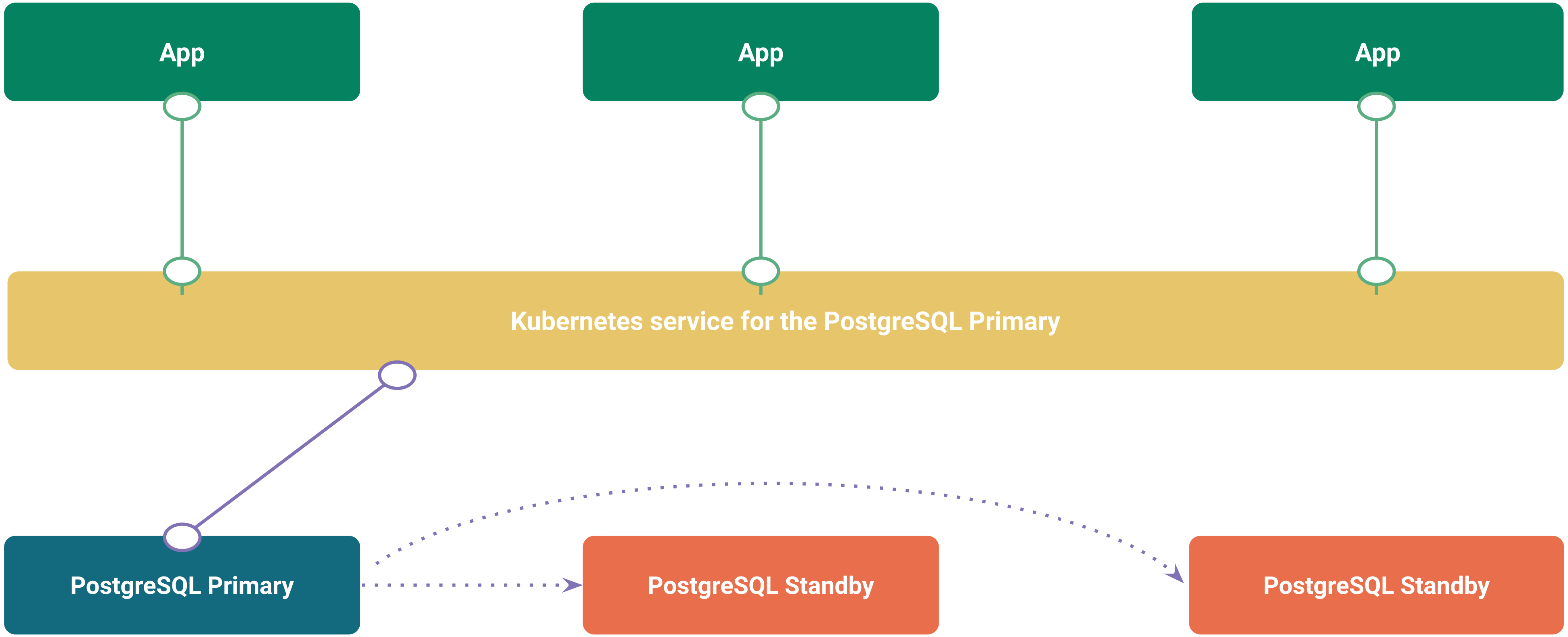
K8s Networking for DBAs

I’m done kicking the tires…

- **Where is my application traffic coming from?**
  - Internal to K8s
  - External and Internal to K8s
- **Where Container Network Interface (CNI) is my company using?**
  - Capabilities vary widely, most support simple ALLOW/DENY
- **Service Types**
  - **ClusterIP:** Default types, connects apps and Postgres within the cluster
  - **NodePort:** Exposes the service on each Node’s IP on a specific port
  - **LoadBalancer:** Use with cloud providers to manage external Postgres access.
  - **ExternalName:** Maps an internal DNS name to an external service using a CNAME record

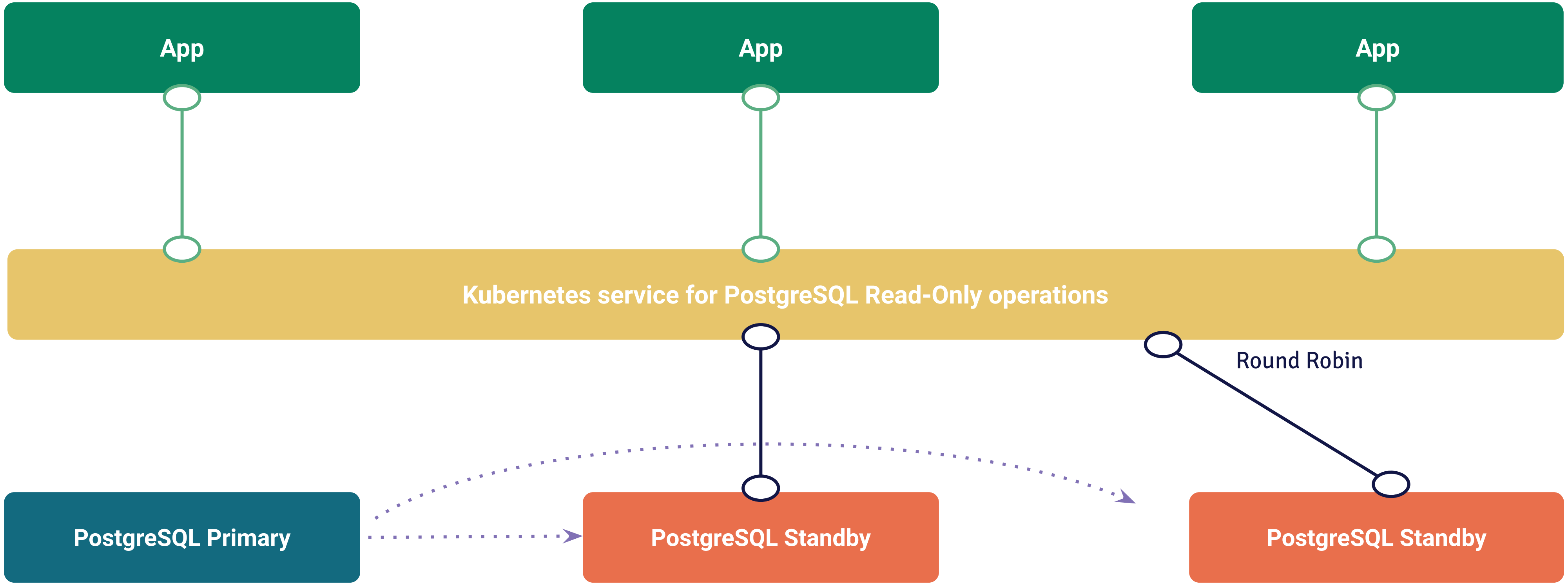
Networking Recommendation for Postgres

Leverage Services



Networking Recommendation for Postgres

Leverage Services

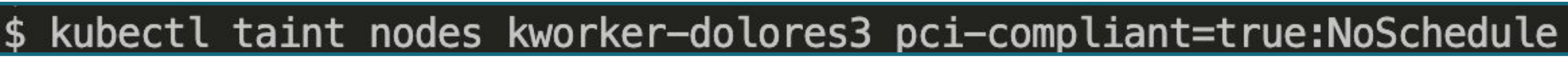


Taints and Tolerations

- Taints and Tolerations are best for **enforcing hard constraints**
  - Pods with databases storing cardholder information can only run on PCI-compliant nodes

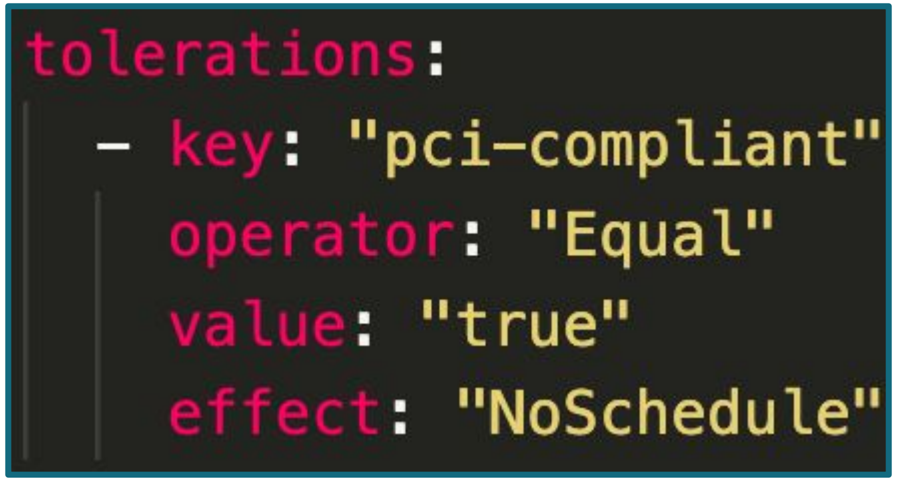
Taints and Tolerations

K8s Worker Node is off-limits to Pods without the pci-compliant toleration



```
$ kubectl taint nodes kworker-dolores3 pci-compliant=true:NoSchedule
```

Toleration in the Pod allows it to be scheduled on the tainted node, now the Pod will only run on PCI-compliant nodes



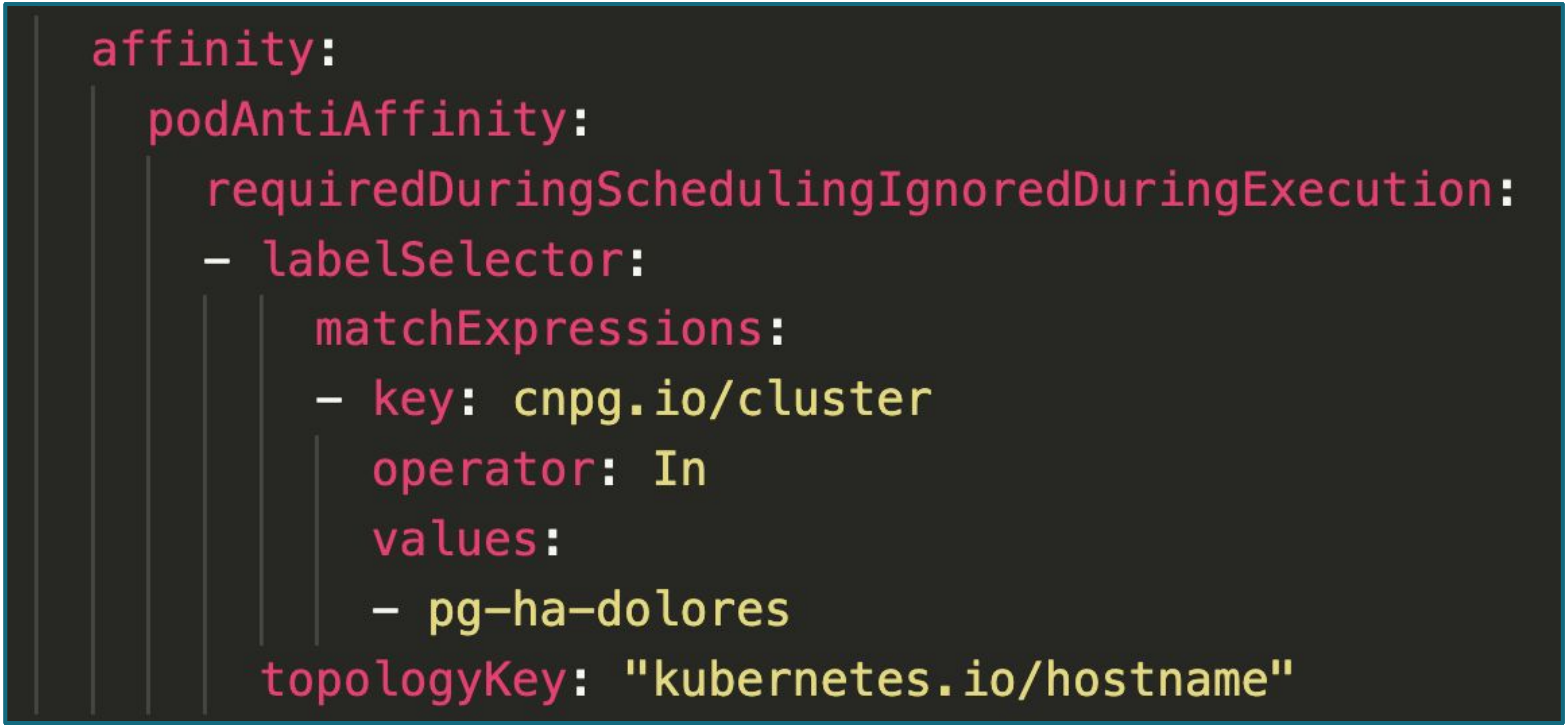
```
tolerations:
  - key: "pci-compliant"
    operator: "Equal"
    value: "true"
    effect: "NoSchedule"
```

Affinity and Anti Affinity

Coupling and Decoupling

- Affinity: Place Pods close together
  - **Example:** Prefer or require the App be scheduled near its corresponding Postgres Pod
  - **In "the cloud":** Ensure the app and Postgres Pods are in the same Availability Zone (AZ) for low-latency access
  - In your data center: Place the App and Postres Pods on Nodes in the same rack
- Anti-Affinity: Spread Pods across nodes
  - **Example:** Preer or require Postgres Pods to be scheduled on different K8s Nodes to increase fault tolerance
  - **In "the cloud":** Ensure each Postgres Pod runs in a different AZ to improve resilience.
  - **In your data center:** Place Postgres Pods on different racks to avoid single points of failure.
- Special Node; Connection Poolers and Proximity
  - **Example:** Prefer or require connection pooler Pods to be scheduled close to their corresponding Postgres Pods to reduce latency

Pod Anti-Affinity



```yaml
affinity:
  podAntiAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
    - labelSelector:
        matchExpressions:
        - key: cnpg.io/cluster
          operator: In
          values:
          - pg-ha-dolores
      topologyKey: "kubernetes.io/hostname"
```

Last One Is Random…But, Managing Postgres Versions

- Easy (Small scale - only a few deployments)
  - Managing PG versions is straightforward with just a few instances
- Challenging (At scale - many deployments)
  - Different major and minor versions across many deployments becomes complex, especially when versions are hardcoded in the spec

Conclusions

🌟 Hopefully, you picked up some useful tips and insights today!

🚀 Ready to explore more? Dive right in with a managed K8s Service or tools like KIND & Rancher Desktop and deploy a Postgres Operator