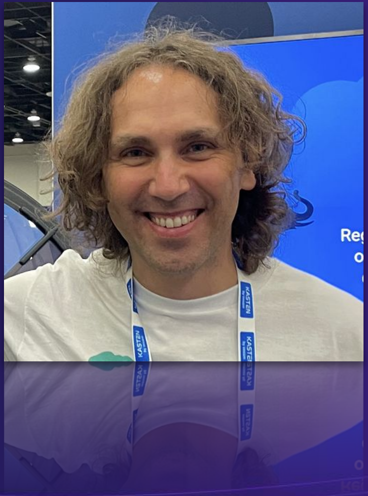
Kubernetes & Data

Gabriele Bartolini
VP Cloud Native at EDB

March 2023

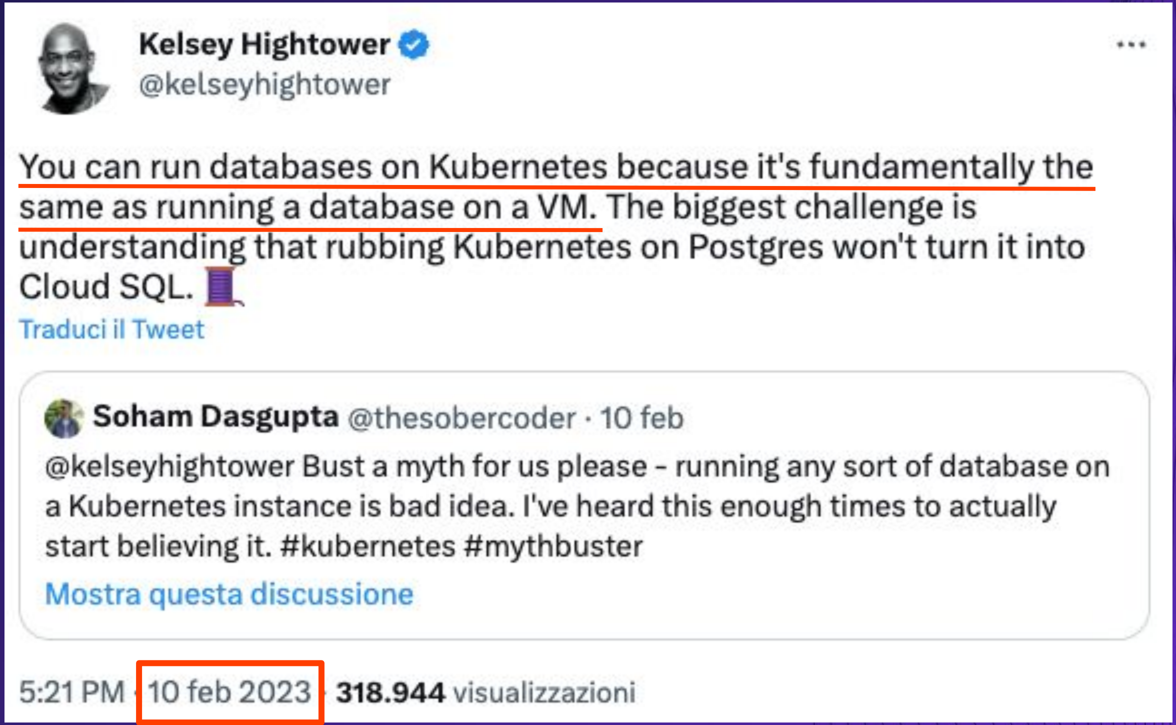
About me



- VP/CTO of Cloud Native at EDB
  - Previously at 2ndQuadrant
- PostgreSQL user since ~2000
  - Community member since 2006
  - Co-founder of PostgreSQL Europe
- DevOps evangelist
- Open source contributor
  - Barman (2011)
  - CloudNativePG (2022)

Follow me: **@_GBartolini_**

Kelsey Hightower
@kelseyhightower



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. 🧵

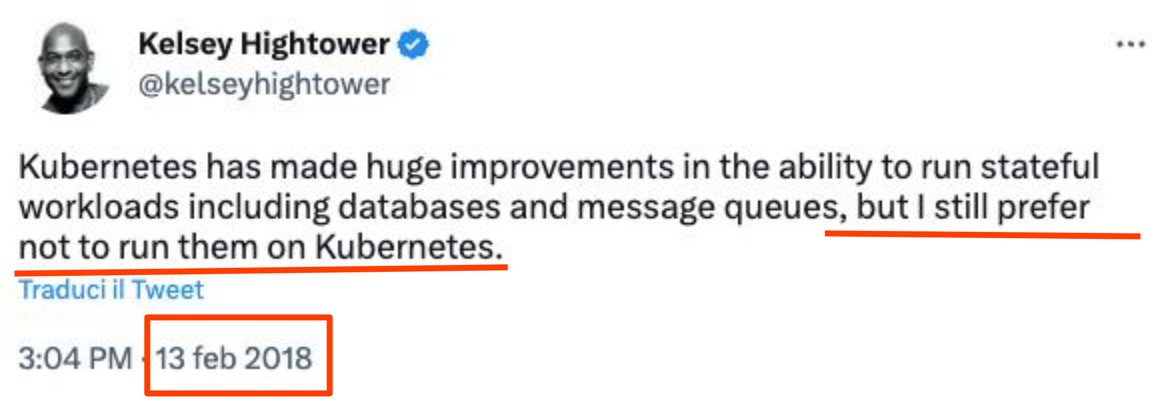
Traduci il Tweet

Soham Dasgupta @thesobercoder · 10 feb

@kelseyhightower Bust a myth for us please - running any sort of database on a Kubernetes instance is bad idea. I've heard this enough times to actually start believing it. #kubernetes #mythbuster

Mostra questa discussione

5:21 PM · 10 feb 2023 · 318.944 visualizzazioni

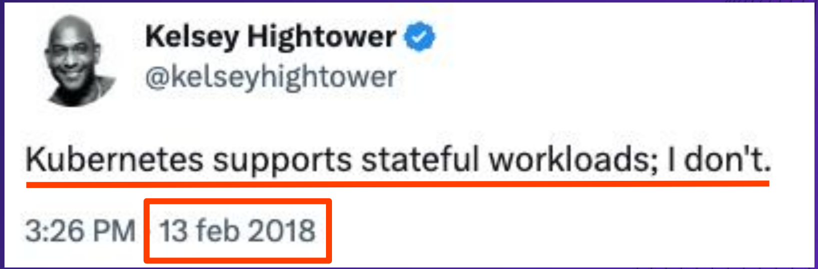


Kelsey Hightower
@kelseyhightower

Kubernetes has made huge improvements in the ability to run stateful workloads including databases and message queues, but I still prefer not to run them on Kubernetes.

Traduci il Tweet

3:04 PM · 13 feb 2018



Kelsey Hightower
@kelseyhightower

Kubernetes supports stateful workloads; I don't.

3:26 PM · 13 feb 2018

A majority (83%) attribute over 10% of their revenue to running data on Kubernetes

One-third of organizations saw their productivity increase twofold.

DoK Community

RESEARCH REPORT

Data on Kubernetes 2022

Insights from over 500 executives and technology leaders on how data on Kubernetes has a transformative impact on organizations, regardless of size or tech maturity

Timeline and team involvement

- **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: my 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

EDB™

“The ***same*** *as running a* ***database*** *on a* ***VM****”*

*I would add: “... provided **you ...**”*

- Know PostgreSQL
- Know Kubernetes
- Have a good **operator** like CloudNativePG

You = You organization, made up of one or more multidisciplinary teams

#1 - The right architecture for Kubernetes

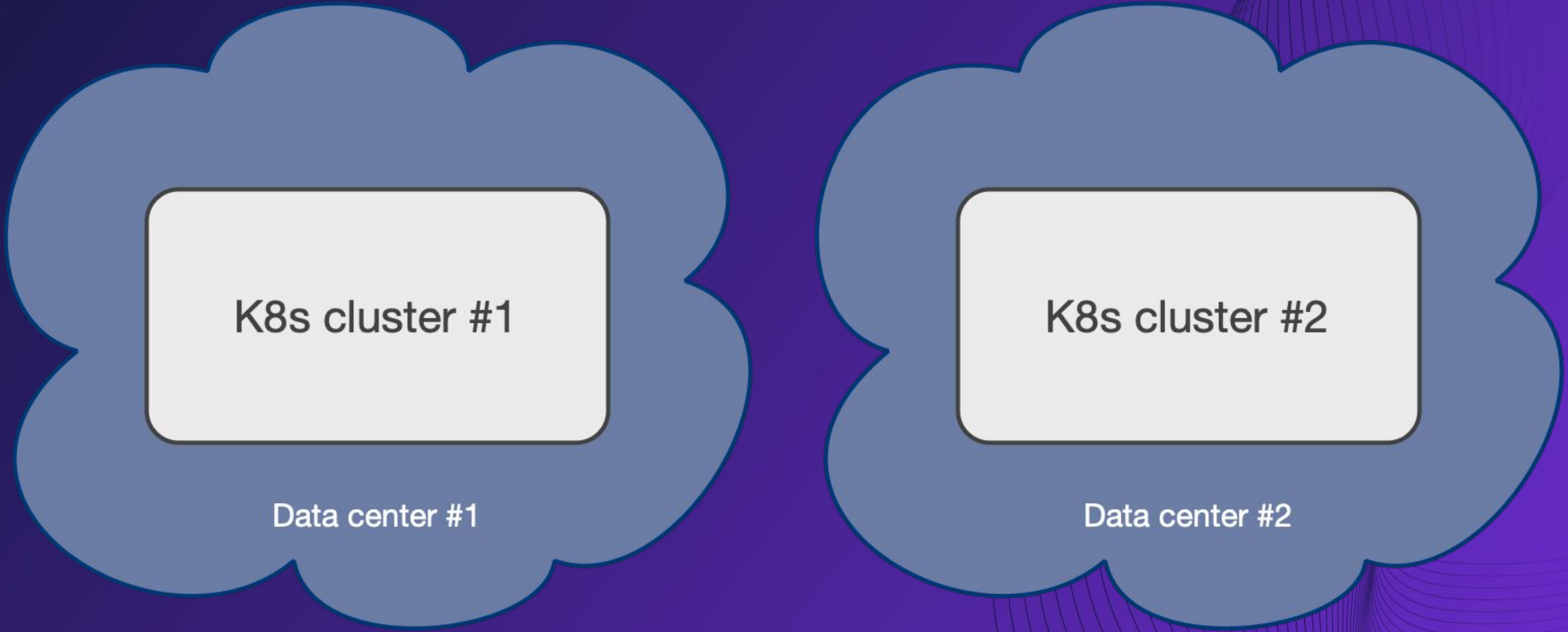
Kubernetes architectural concepts

- A Kubernetes Cluster (**k-cluster**)
- Availability zones (**AZ**)- also known as failure zones or data centers
  - Connected by redundant, low-latency, private network connectivity
  - At least 3 per k-cluster
- Kubernetes control plane to be distributed across the AZ
- Kubernetes worker nodes in each AZ running applications (workloads)
- Normally:
  - **1 k-cluster = 1 region with 3+ AZ**

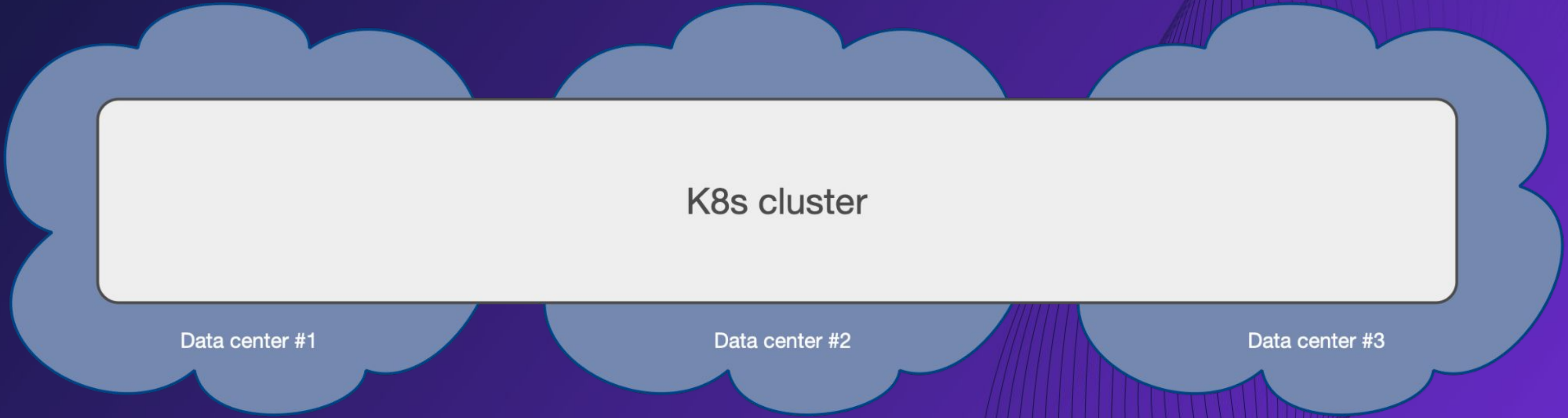
1 k-cluster = 1 region with 3+ AZ

- Taken for granted if you know Kubernetes
- All major public cloud providers offering managed K8s services have 3+ AZ
- What about on-premise deployments?
  - You need to plan in advance
  - Stay away from the “2 data center in a region” setup typical of “Lift-and-Shift” exercises
    - Often results in 2 separate Kubernetes clusters
      - Severely impacts the benefits of Kubernetes, particularly self-healing
      - Shifts maintenance and procedural complexity up to the application level

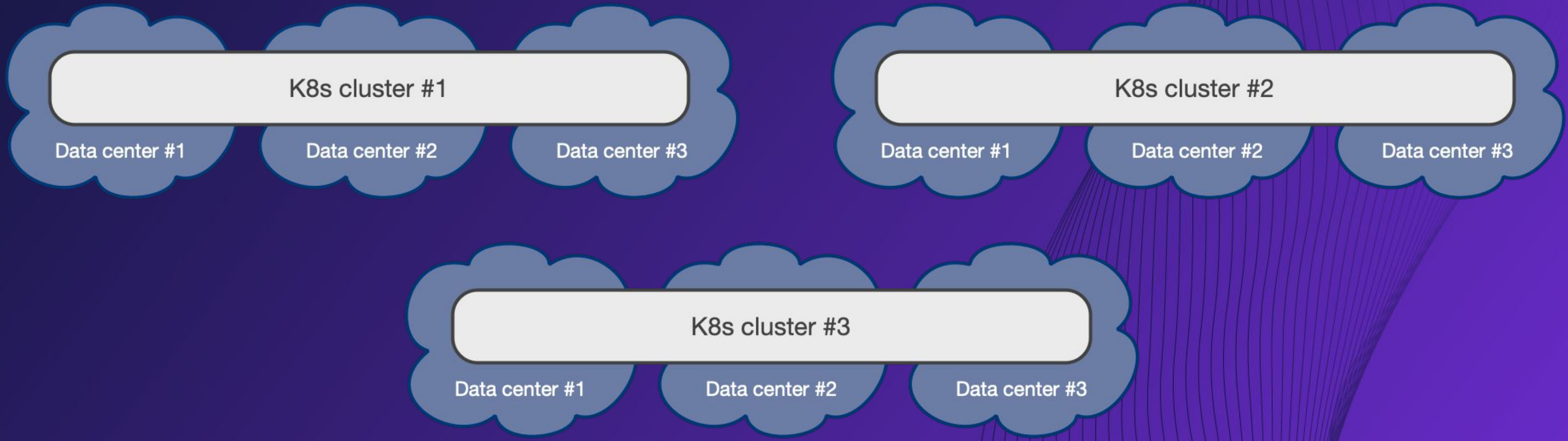
No!



Yes!



Yes! Yes! Yes!

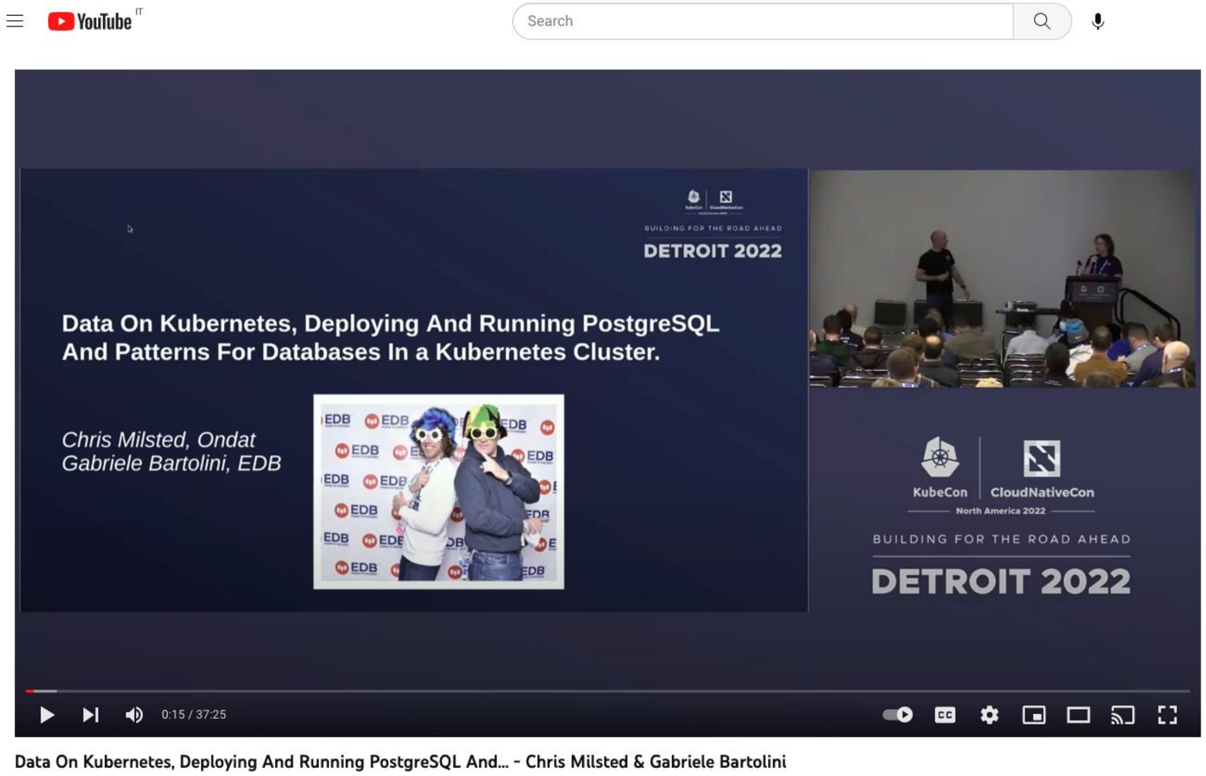

K8s cluster #1
Data center #1
Data center #2
Data center #3
K8s cluster #2
Data center #1
Data center #2
Data center #3
K8s cluster #3
Data center #1
Data center #2
Data center #3

#2 - Synchronizing the state

Synchronizing the state of a Postgres database

- Being a DBMS, PostgreSQL is a **stateful workload** in Kubernetes
- Stateless workloads achieve HA and DR mainly through traffic redirection
- Stateful workloads require the state to be replicated in multiple locations:
    - **Storage-level** replication
    - **Application-level** replication (in our case, application = Postgres)
- Postgres has a very robust and powerful native replication system
    - We've built it
    - Founded on the Write Ahead Log
    - Read-only standby servers
    - Supports also synchronous replication controlled at the transaction level
- **We recommend application-level** over storage-level replication for Postgres

KubeCon NA 2022 - talk with Chris Milsted (Ondat)

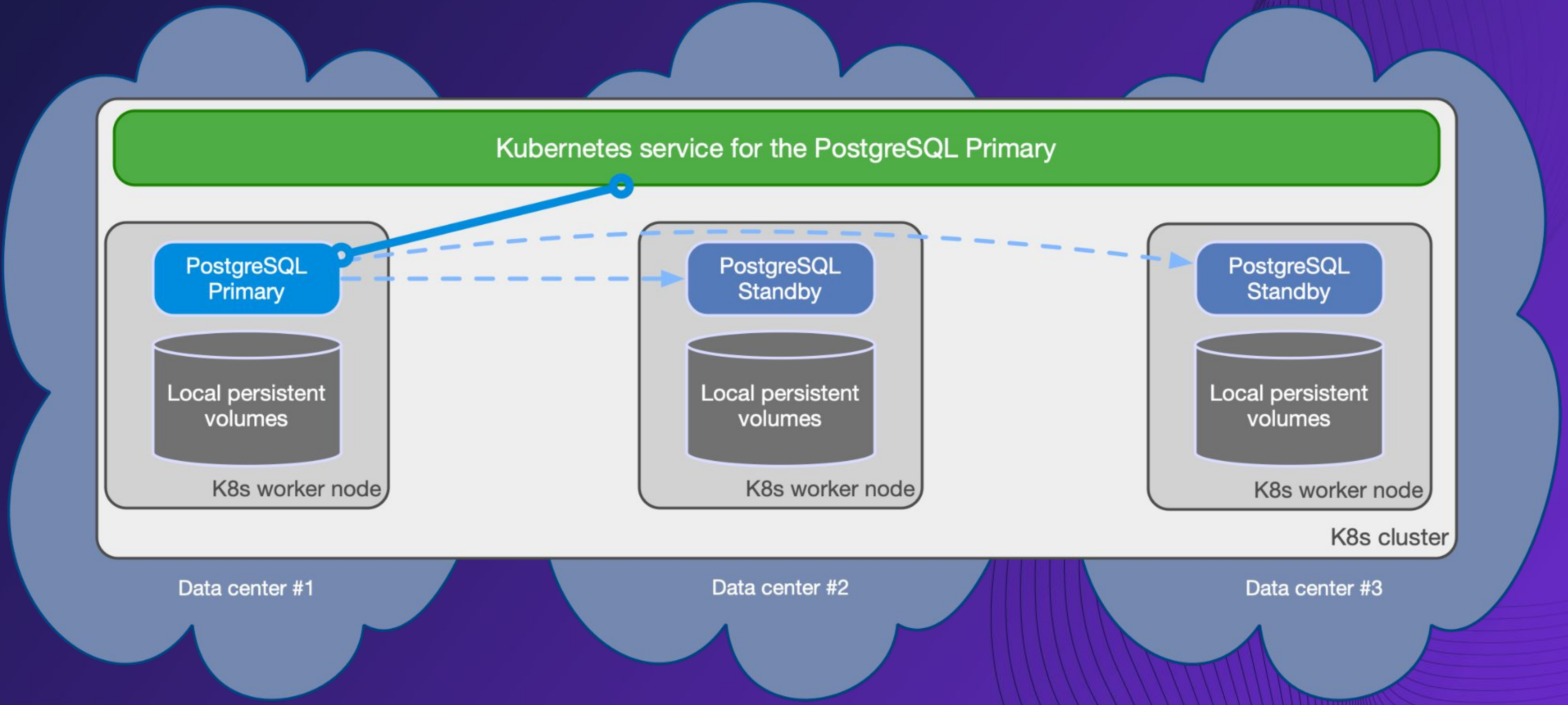


Data On Kubernetes, Deploying And Running PostgreSQL And Patterns For Databases In a Kubernetes Cluster.

Chris Milsted, Ondat
Gabriele Bartolini, EDB

Data On Kubernetes, Deploying And Running PostgreSQL And... - Chris Milsted & Gabriele Bartolini

Yes!


Kubernetes service for the PostgreSQL Primary
PostgreSQL Primary
PostgreSQL Standby
PostgreSQL Standby
Local persistent volumes
Local persistent volumes
Local persistent volumes
K8s worker node
K8s worker node
K8s worker node
K8s cluster
Data center #1
Data center #2
Data center #3

#3 - The right storage for you

Storage management

- Storage is the most critical component for a database
- Direct support for Persistent Volume Claims (PVC)
  - We deliberately do not use Statefulsets
- The PVC storing the PGDATA is central to CloudNativePG
  - Our motto is: “PGDATA is worth a 1000 pods”
- Storage agnostic
- Freedom of choice
  - Local storage
  - Network storage
- Automated generation of PVC
  - Support for PVC templates
  - Storage classes

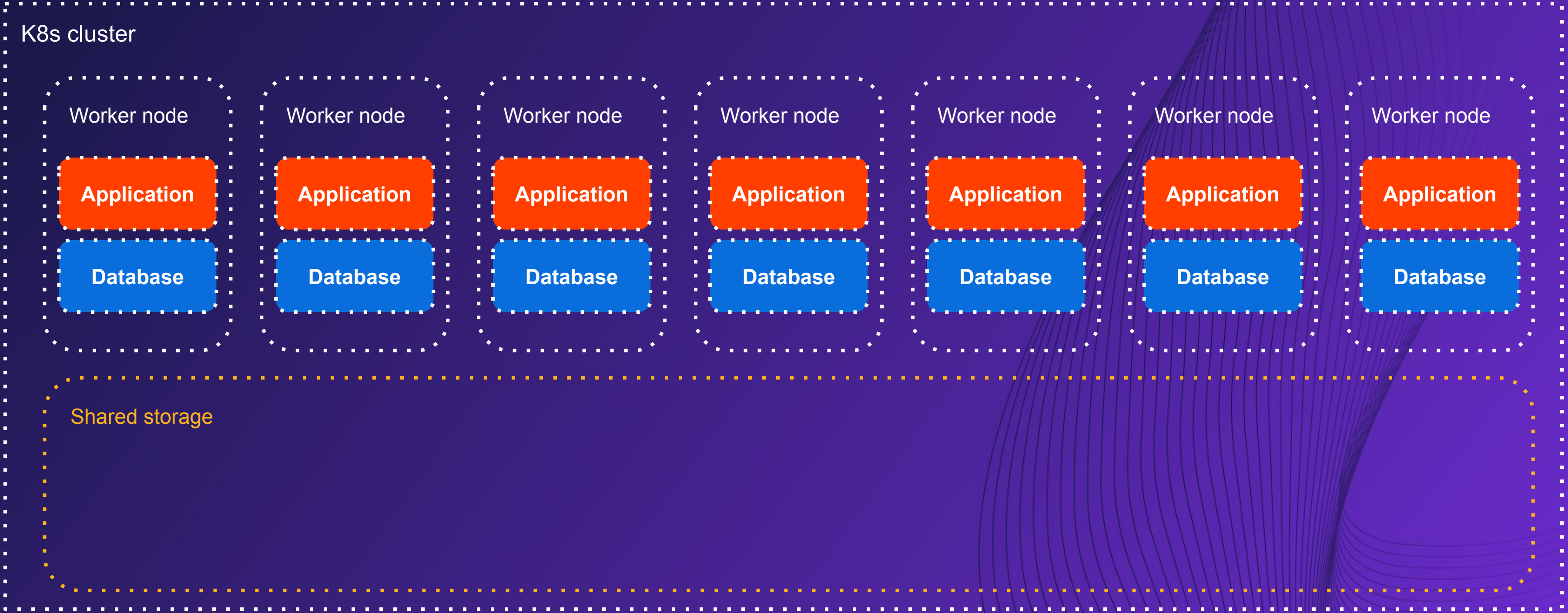
Main components

- Kubernetes cluster
- Availability zone
- Application pod
- Postgres pod
- Kubernetes worker node
- Network storage
- Local storage
    - i.e. dedicated and local to the worker node

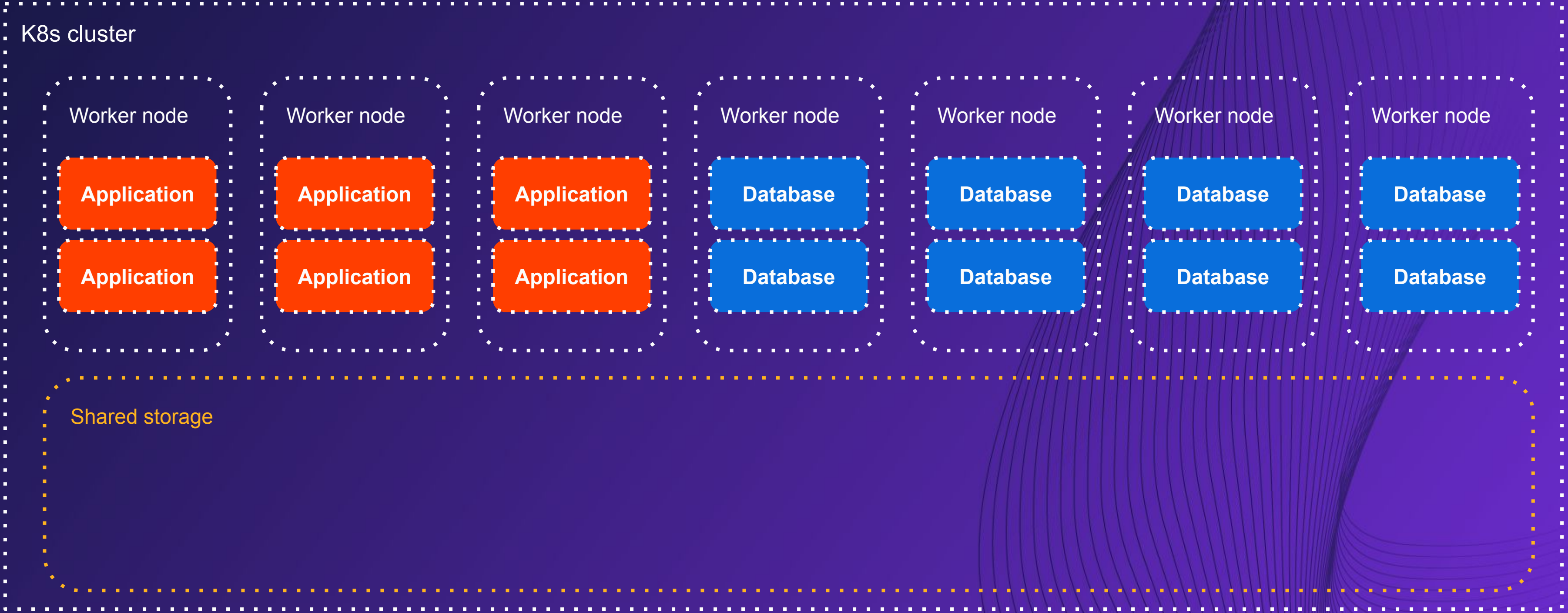
Scheduling Postgres instances with CloudNativePG

- Entirely declarative!
- Affinity section in the `Cluster` specification
  - pod affinity/anti-affinity
  - node selectors
  - tolerations against taints placed on nodes

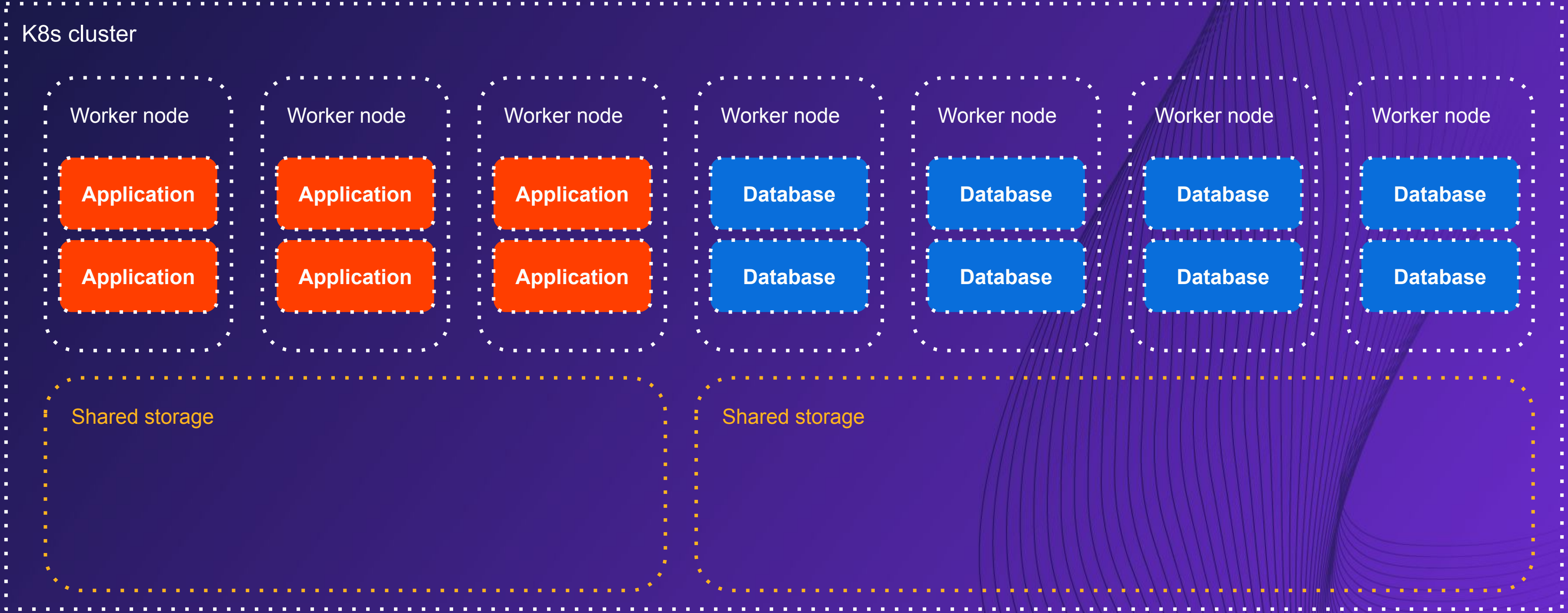
Shared workloads, shared storage #1



Shared workloads, shared storage #2

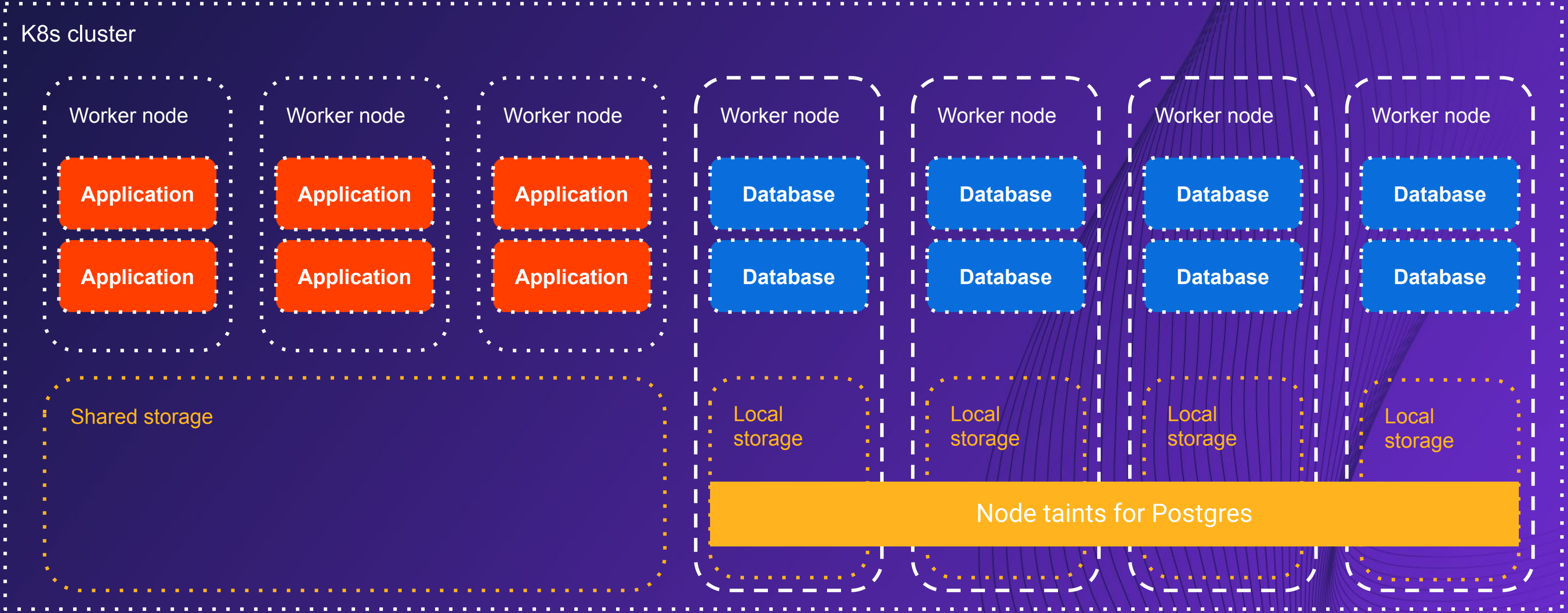


Shared workloads, shared storage #3



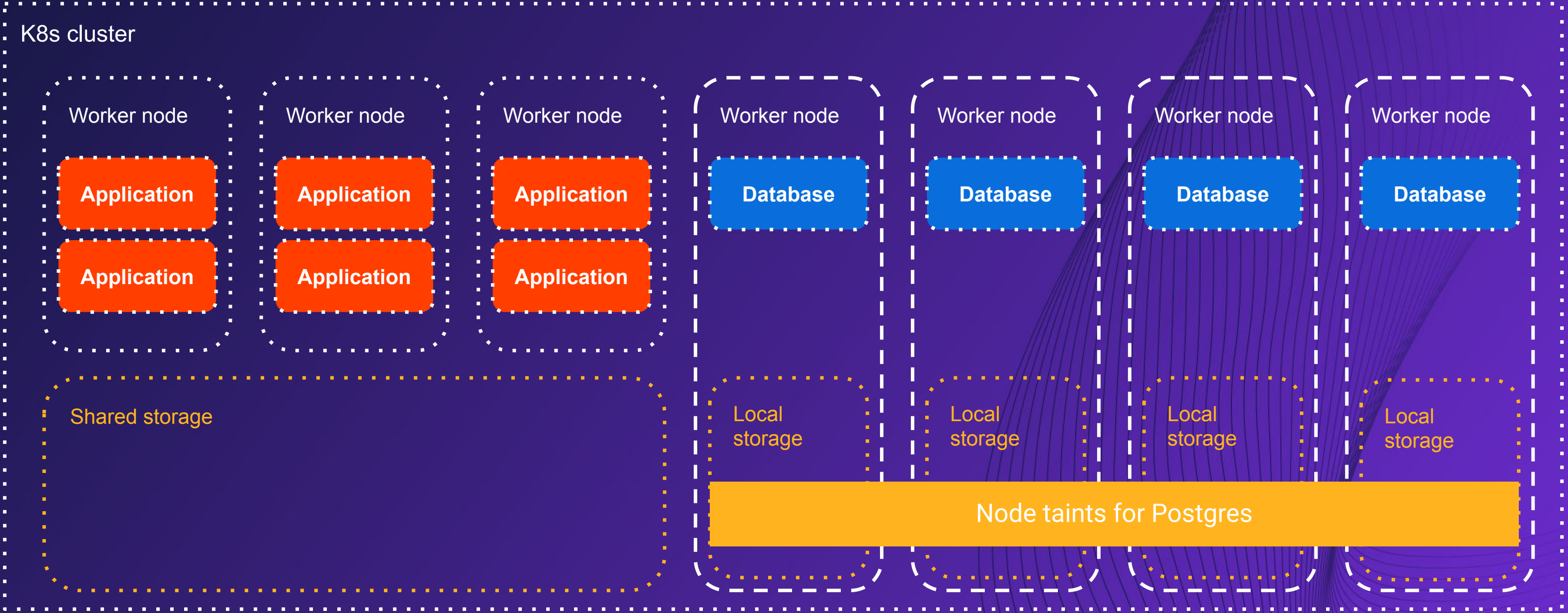
Shared workloads, local storage

Good value for money!

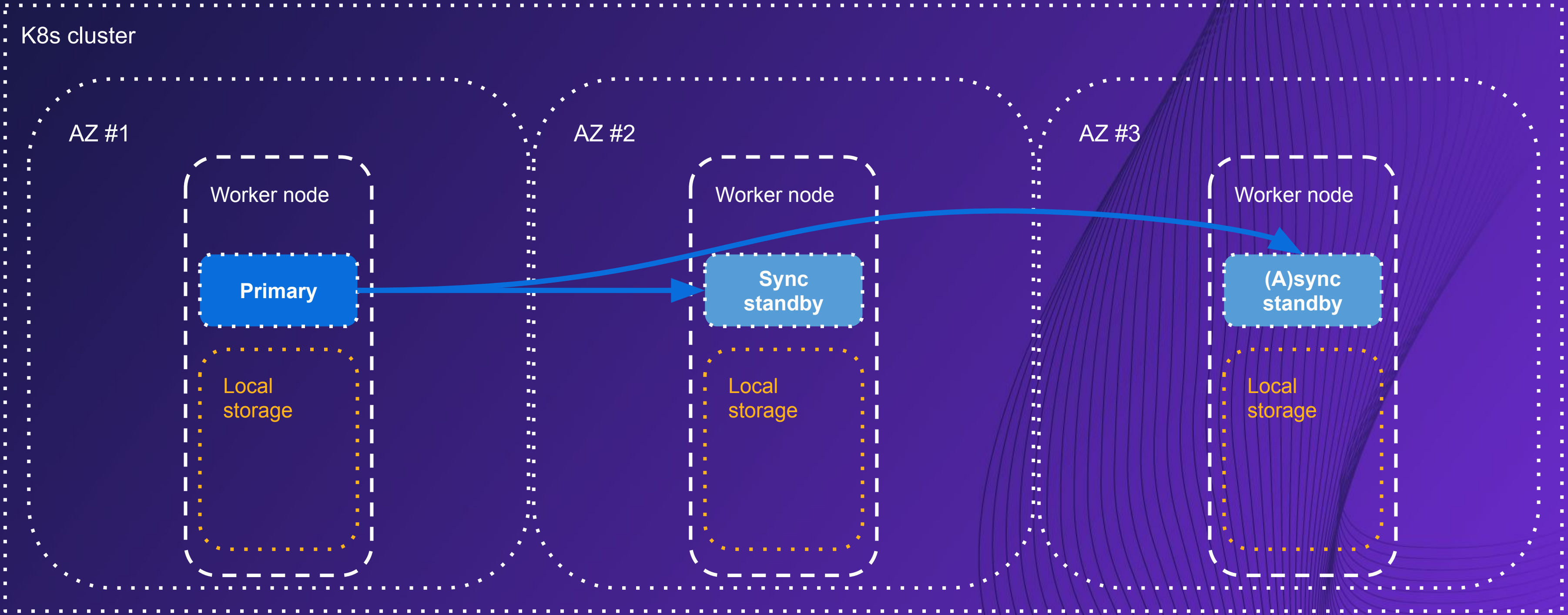


Dedicated workloads, local storage

★★ Best Postgres results!

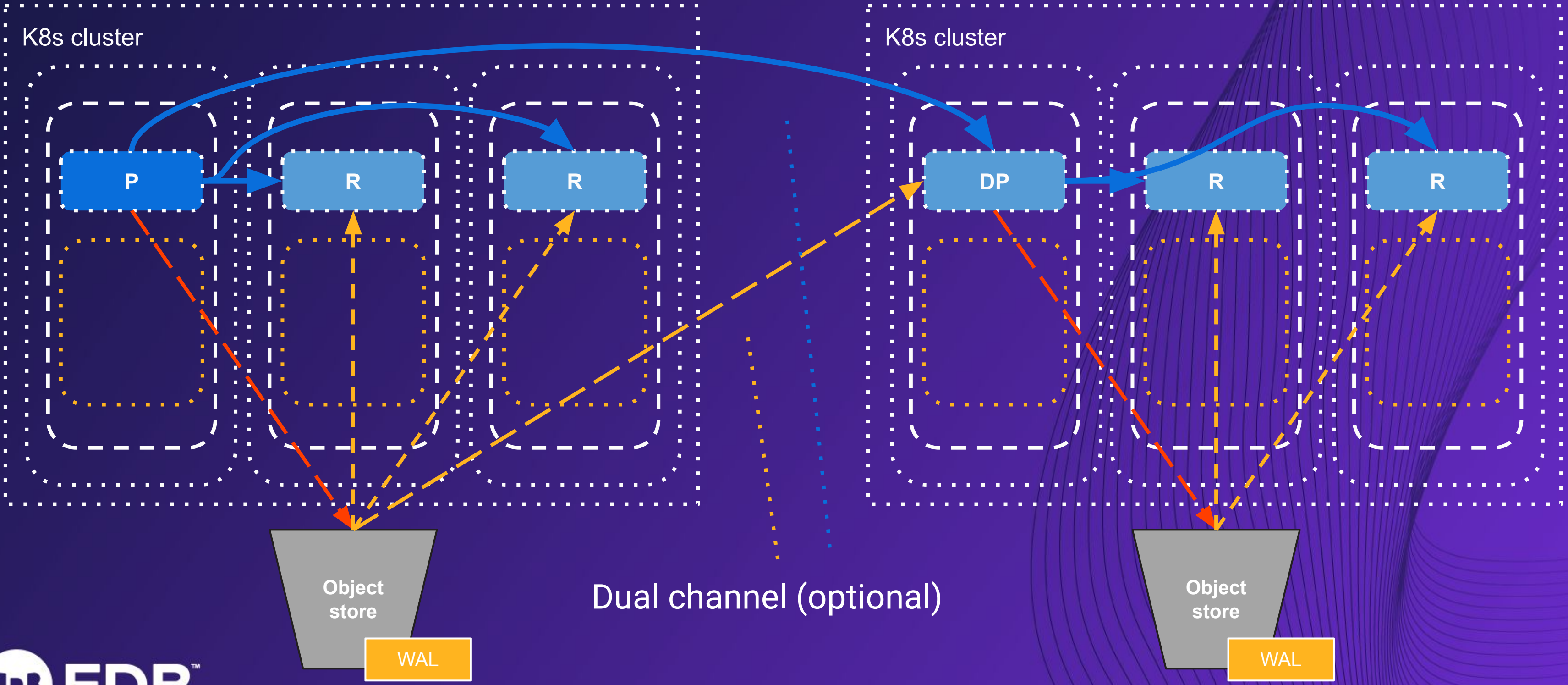


Shared nothing architecture



Shared nothing architecture (hybrid/multi)

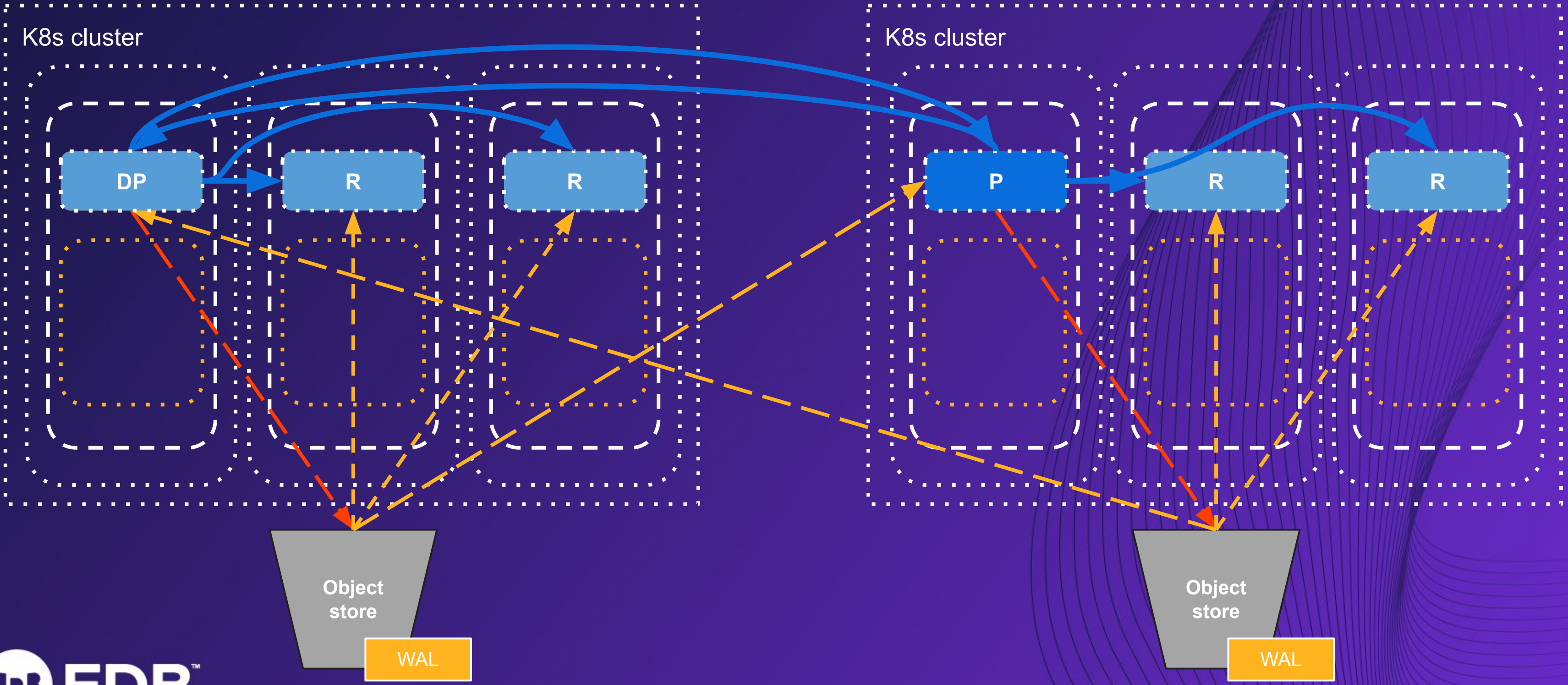
"Replica cluster" feature in CloudNativePG



Dual channel (optional)

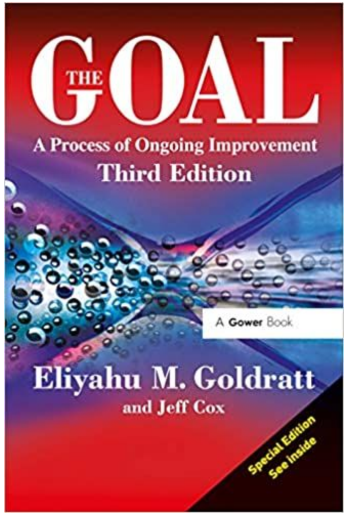
Shared nothing architecture (hybrid/multi)

"Replica cluster" feature in CloudNativePG



#4 - The “Goal”

(“Your goal”)



Identify your business continuity goals

- Recovery Point Objective (RPO)
  - Time it takes for you to safely store each WAL file in separate locations
- Recovery Time Objective (RTO)
  - Time it takes for you to promote a standby as primary after a failure
    - Single k-cluster (region)
    - To a different k-cluster (region)
  - Time it takes for you to issue a PITR operation from a backup
- Identify your SPOFs!
- Practice! Measure! Improve!

RPO with CloudNativePG

- Recovery Point Objective (RPO)
  - WAL files are archived to object stores at least every 5 minutes, depending on the workload
  - RPO <= 5 minutes
- Recovery Time Objective (RTO)
  - Same k-cluster:
    - Automated failover
    - Recommended setup: 3 instances with 1 sync standby
    - Instantaneous detection by Kubernetes
      - (we had to introduce delayed failover configuration)
    - RTO = time taken by a standby to exit recovery and become primary
      - Normally between 5 seconds and a minute
      - Depends on the workload and lag of a standby
  - Different k-cluster:
    - Use replica clusters with WAL shipping and/or streaming
    - Current: manual detection and triggering of the promotion

RPO with CloudNativePG

- HA replicas:
  - Asynchronous replicas: RPO ~ 0
  - Synchronous replicas RPO = 0
- Local object store:
  - WAL files are archived to object stores at least every 5 minutes
    - Depending on the workload
  - RPO <= 5 minutes
- Global object store:
  - (Stored in another region)
  - Local object store RPO + relay of WAL file to another region
  - RPO <= 10 minutes

RTO with CloudNativePG

- Same k-cluster:
  - Automated failover
  - Recommended setup: 3 instances with 1 sync standby
  - Instantaneous detection by Kubernetes
    - (we had to introduce delayed failover configuration)
  - RTO = time taken by a standby to exit recovery and become primary
    - Normally between 5 seconds and a minute
    - Depends on the workload and lag of a standby
- Different k-cluster:
  - Use replica clusters with WAL shipping and/or streaming
  - Current: manual detection and triggering of the promotion
- PITR varies on the database size and the amount of WAL to replay

EDB™

Key takeaways

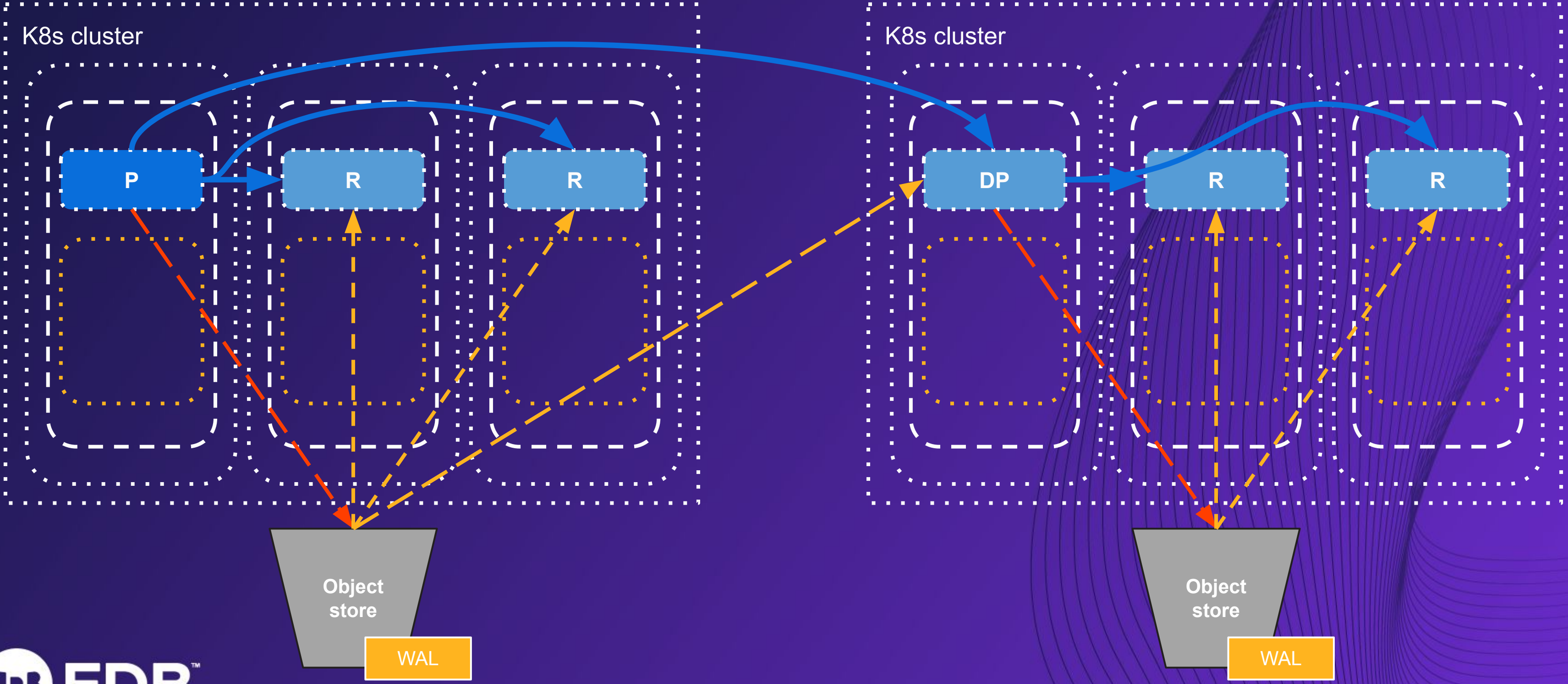
1. Take advantage of 3+ AZ K-Clusters
2. Rely on PostgreSQL Primary/Standby clusters - like you did on VMs
3. Choose your storage carefully - like you did on VMs
4. Plan your infrastructure around your goals
   - RPO
   - RTO
   - Benchmarks
5. Shared nothing architecture, if you can
   - Otherwise, at least separate PostgreSQL workloads from the rest of your cluster
6. Application and database must be in the same K-Cluster
   - Applications are automatically rerouted to the primary via the updated service

EDB™

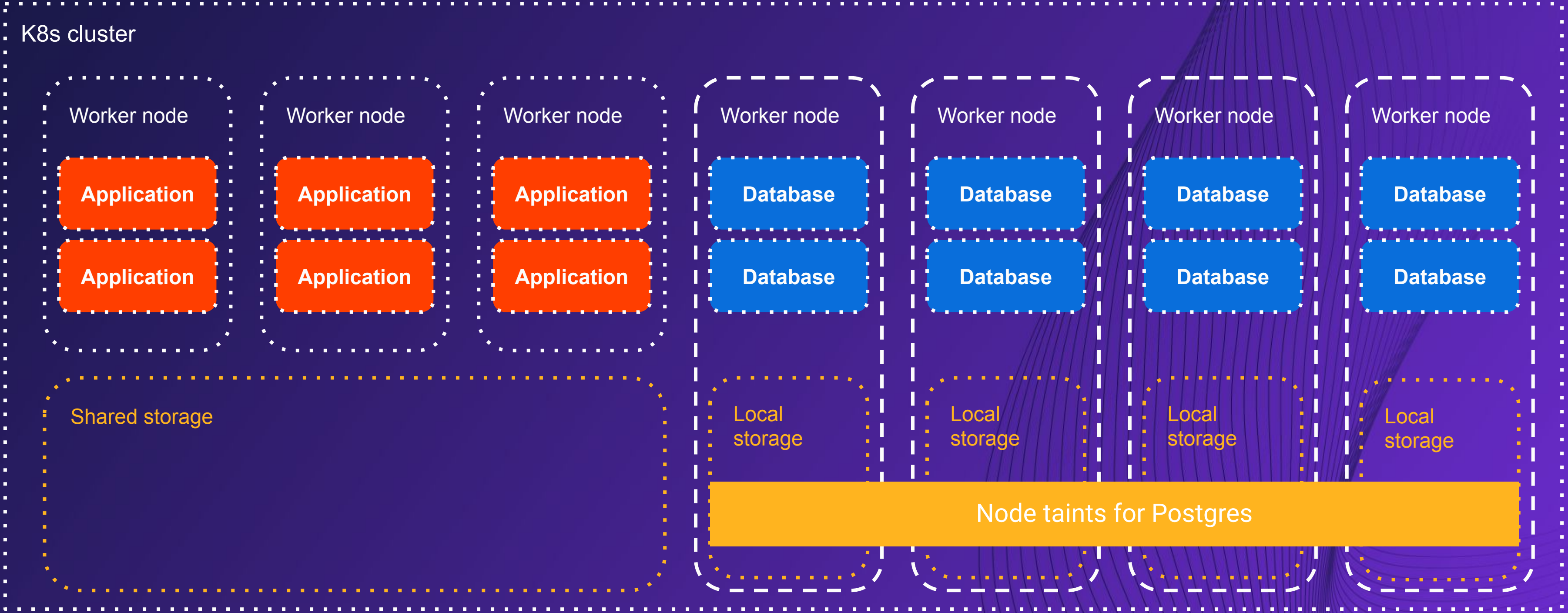
#1 architecture

★★★★

"Replica cluster" feature in CloudNativePG



Shared workloads, local storage


K8s cluster
Worker node
Application
Application
Worker node
Application
Application
Worker node
Application
Application
Worker node
Database
Database
Worker node
Database
Database
Worker node
Database
Database
Worker node
Database
Database
Shared storage
Local storage
Local storage
Local storage
Local storage
Node taints for Postgres