
KubeCon
CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022
TiKV
TiKV
HELM

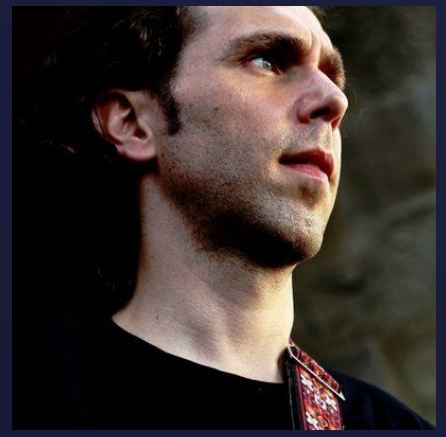
KubeCon | CloudNativeCon North America 2022

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

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

Chris Milsted, Ondat
Gabriele Bartolini, EDB



Part 1 - Data in/on Kubernetes

Resiliency

The U.S. Department of Homeland Security defines resilience as, “the ability to adapt to changing conditions and withstand and rapidly recover from disruption due to emergencies”

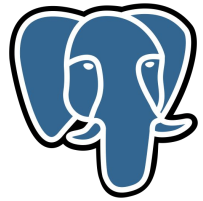
In IT we also tend to think about Planned events as well (e.g. maintenance and migrations to give examples). Expanding the above, we could say resilience is the ability of an organisation to absorb disruption and still continue operating effectively, even when these disruptions can be planned and self-inflicted.

Frameworks such as NIST outline a set of plans which can be prepared such as a Business Continuity Plan, Disaster Recovery Plan and Cyber Incident Response Plan. These will identify business operations, some of which could depend on PostgreSQL clusters running in a kubernetes cluster.

For a business activity, a Maximum Tolerable Downtime (MTD) should be defined by the business. The MTD set by the business can be used to calculate Recovery Time Objective (RTO) and Recovery Point Objective (RPO) to be applied your PostgreSQL database.

Part 2 - PostgreSQL and CloudNativePG

PostgreSQL in one slide



- Aka Postgres
- 25+ years of innovation
- Very popular
- Open source (TPL)
- Multi-purpose database
- One primary, multiple replicas
  - Streaming replication
  - Physical and logical
  - Sync and async
  - Cascading
  - Also file based
- Online Continuous Backup
- Point In Time Recovery
- Declarative Partitioning
- Parallel queries
- Extensible
  - PostGIS, Timescale, …
- JSON support
- ACID transactions
- SQL standard

CloudNativePG in one slide

- Kubernetes operator
- Open source
  - Apache License 2.0
  - Vendor neutral openly governed
  - Originally created by EDB
  - Applied for CNCF sandbox
- Extends the K8s controller
  - Status of the cluster
  - No statefulsets, no Patroni
- Fully declarative
  - Convention over configuration
- Production ready

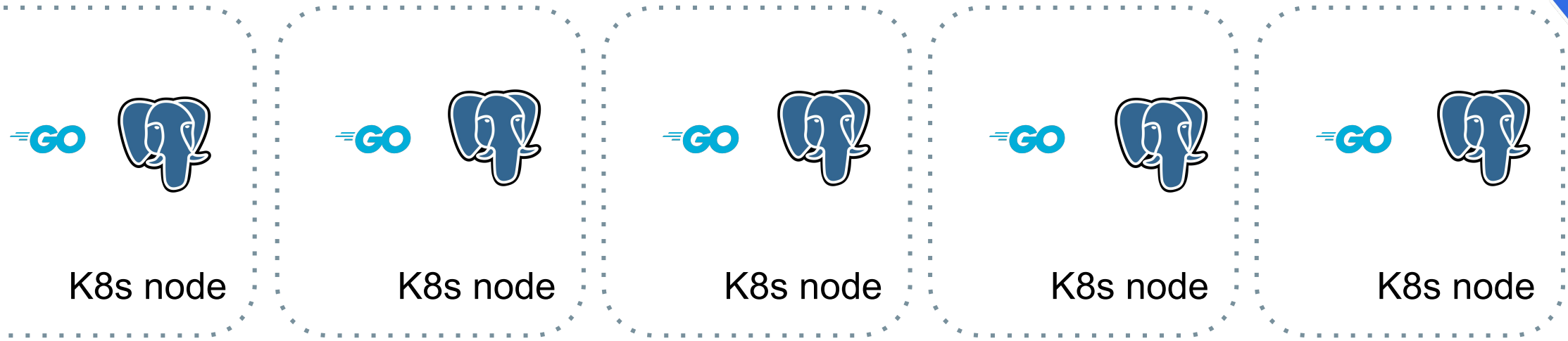
- Automated failover
- Services for RW and RO workloads
- Backup and recovery
- mTLS
- Scale up/down of read replicas
- Rolling updates
- Affinity control
- Native Prometheus exporters
- Log in JSON format to stdout
- … and much more

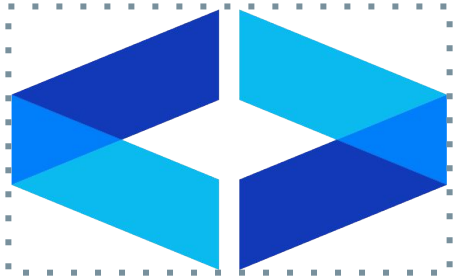
URL: github.com/cloudnative-pg

Storage management

- **Storage is the most critical component for a database**
- The PVC storing the PGDATA is central to CloudNativePG
  - Our motto is: *"The PGDATA PVC is worth a 1000 pods"*
- Freedom of choice
  - Local storage
  - Network storage
- Direct support for Persistent Volume Claims (PVC)
- Automated generation of PVC
  - Support for PVC templates
  - Storage classes

Architecture example #1





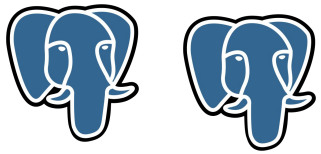
Storage

Architecture example #2

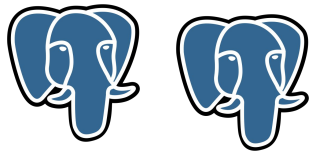
Fine scheduling control: taints on nodes, selectors, …

K8s node

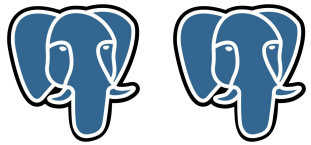
K8s node



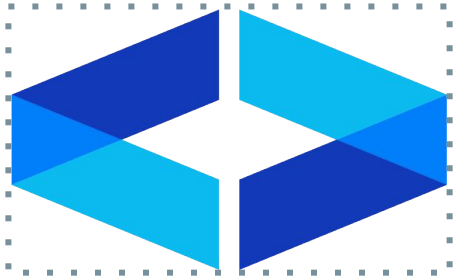
K8s node



K8s node

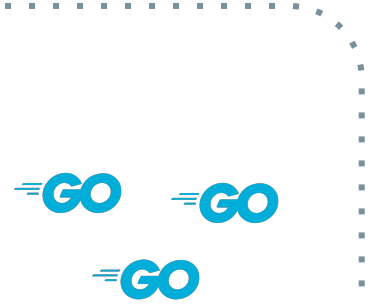


K8s node

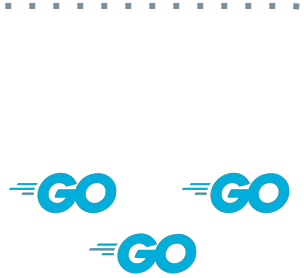


Storage

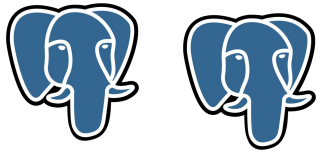
Architecture example #3



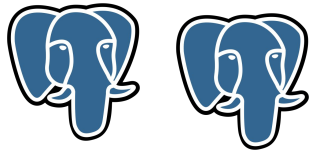
K8s node



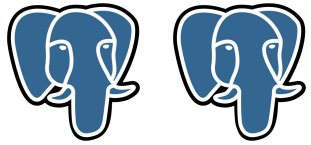
K8s node



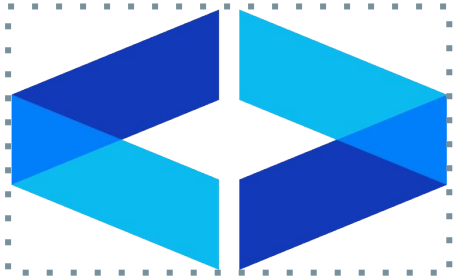
K8s node



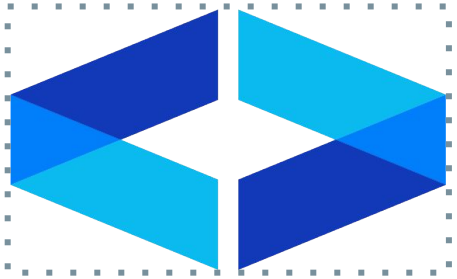
K8s node



K8s node

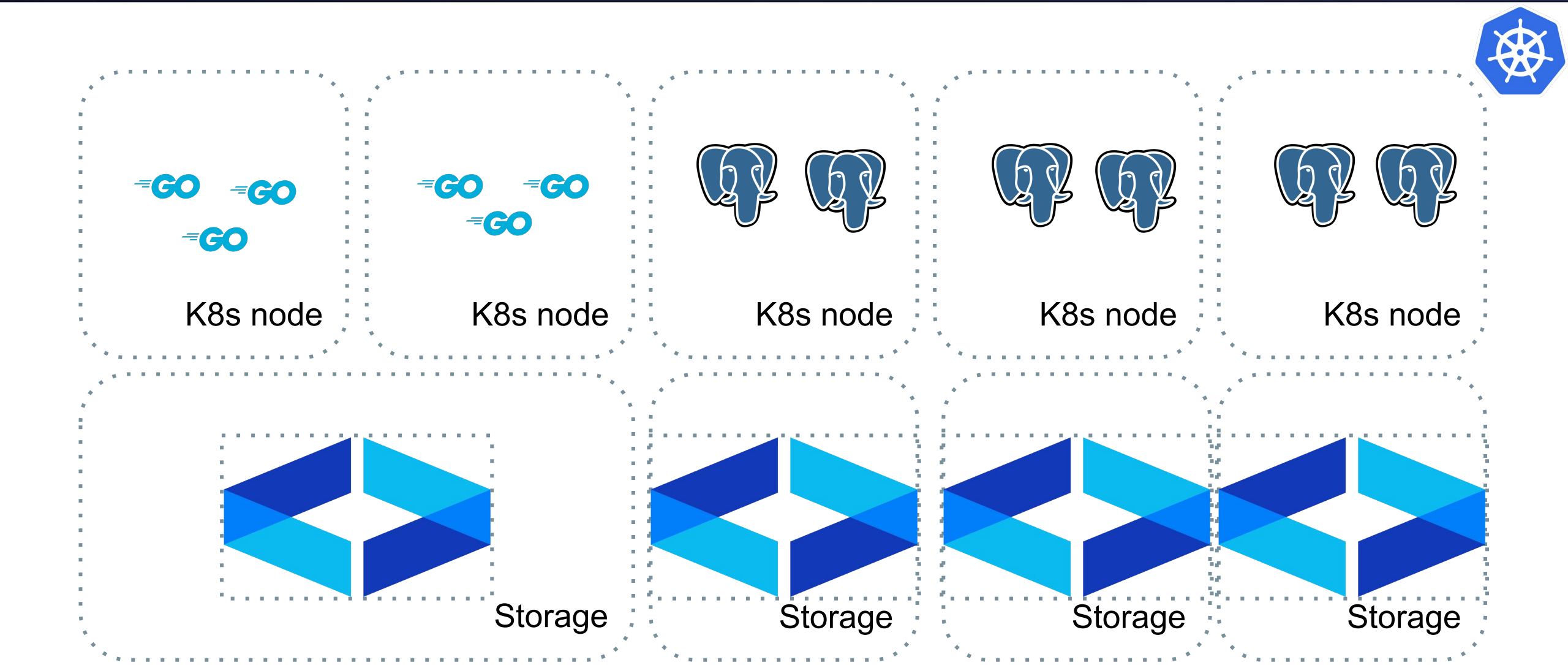


Storage



Storage

Architecture example #4



Architecture example #5

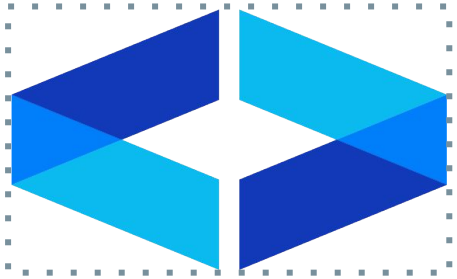
K8s node

K8s node

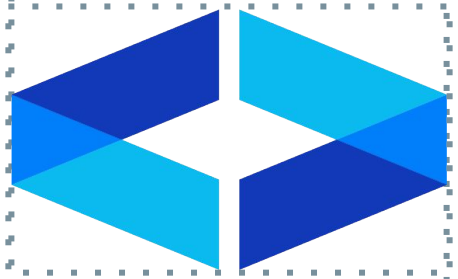
K8s node

K8s node

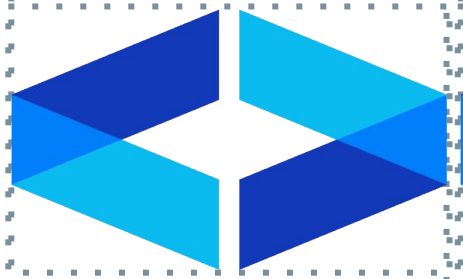
K8s node



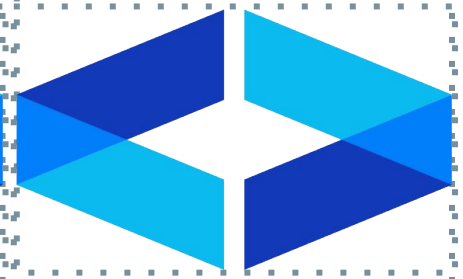
Storage



Storage

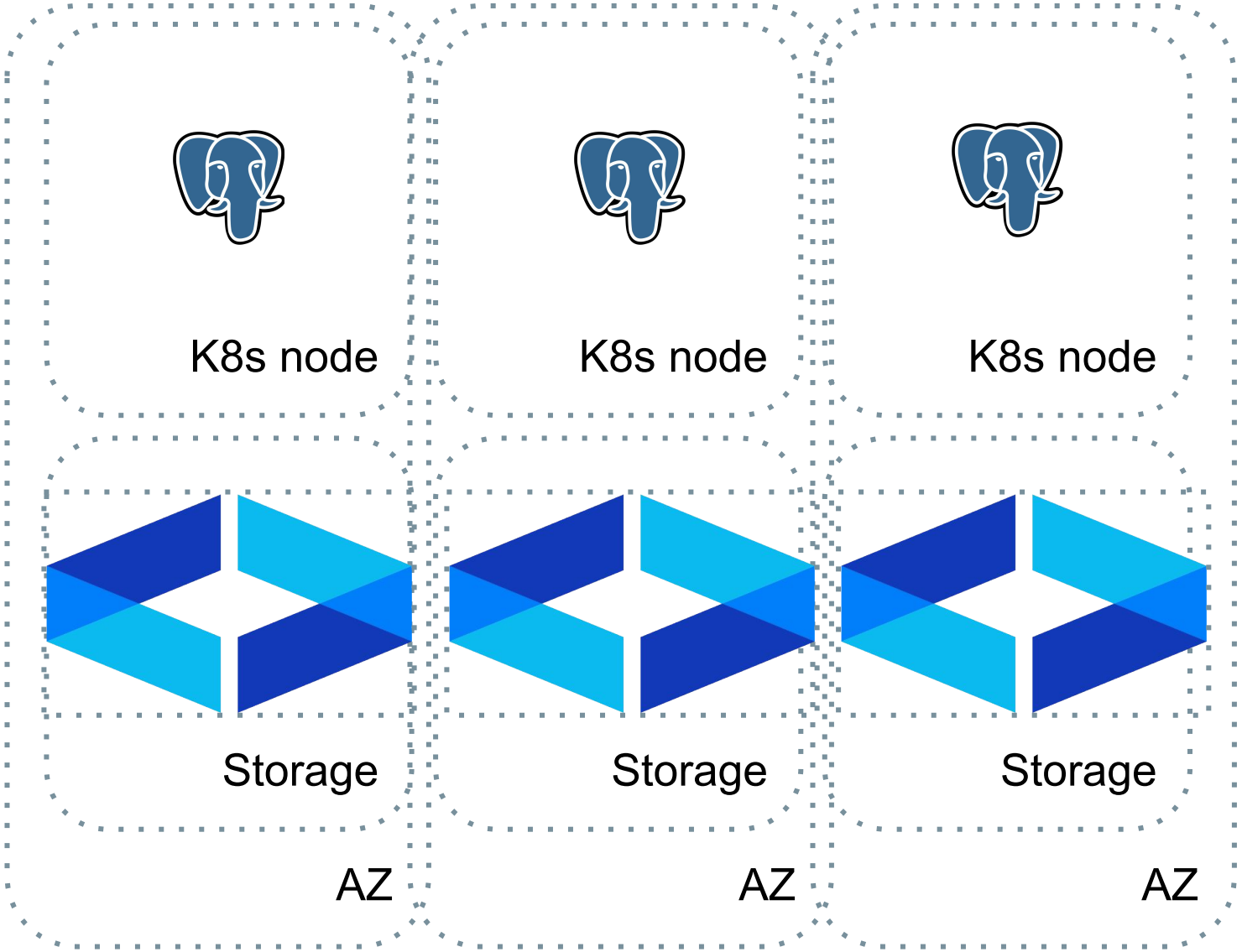


Storage



Storage

Architecture example #6



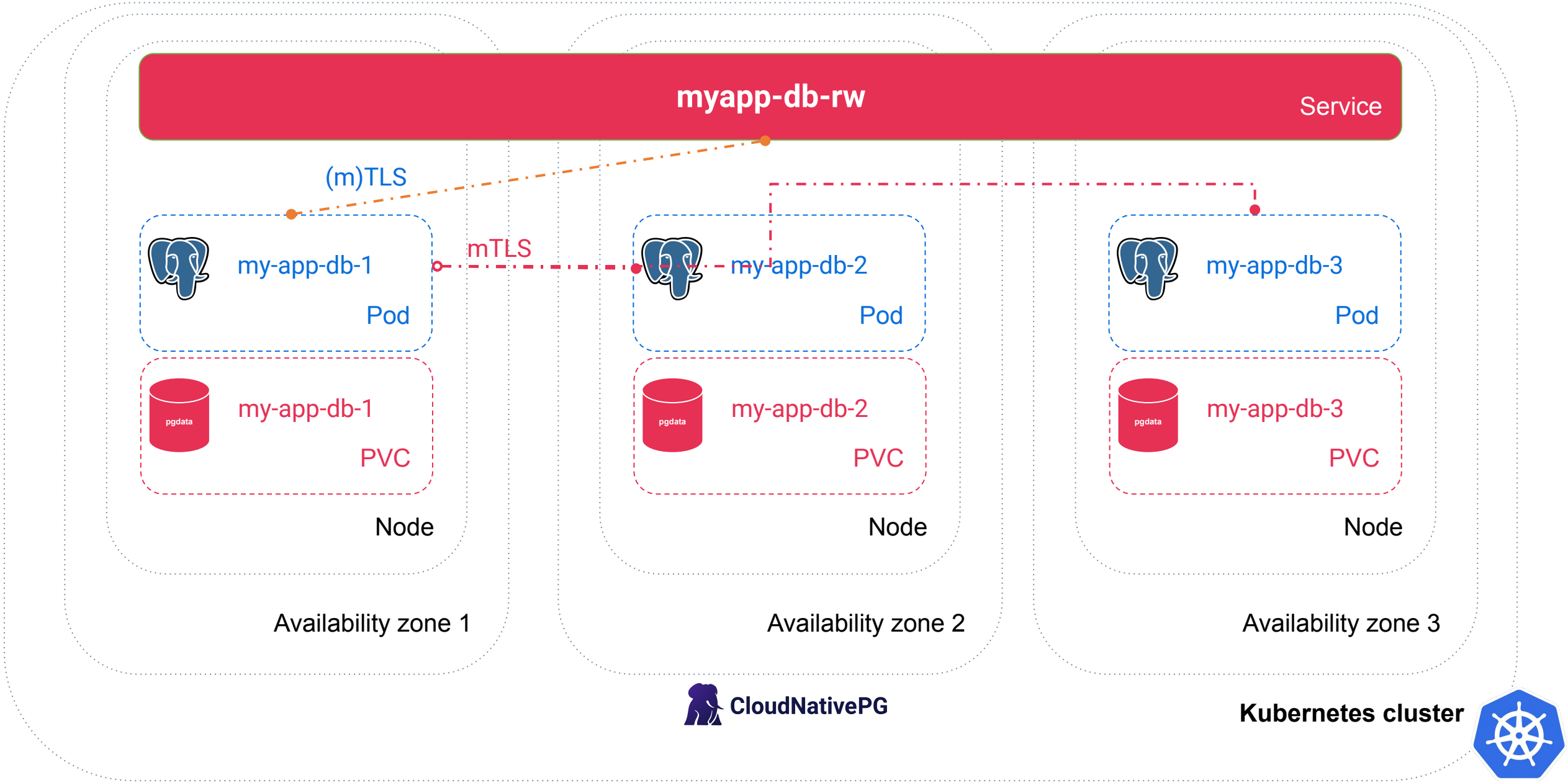
An example of configuration

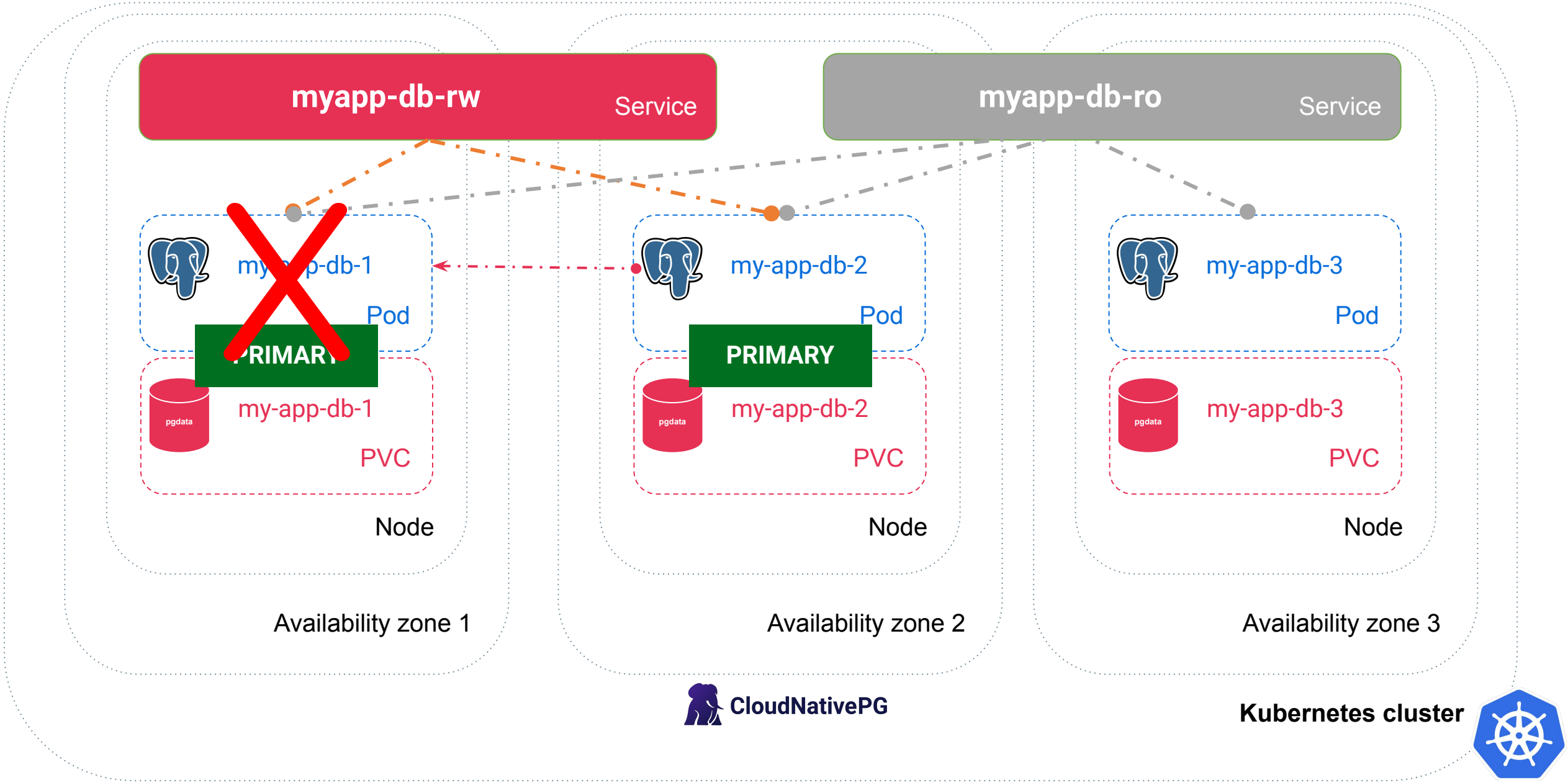

```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: myapp-db
spec:
  instances: 3

  affinity:
    topologyKey: topology.kubernetes.io/zone

  storage:
    size: 50Gi

  walStorage:
    size: 5Gi
```


myapp-db-rw
Service
(m)TLS
mTLS
my-app-db-1
Pod
my-app-db-1
PVC
pgdata
Node
my-app-db-2
Pod
my-app-db-2
PVC
pgdata
Node
my-app-db-3
Pod
my-app-db-3
PVC
pgdata
Node
Availability zone 1
Availability zone 2
Availability zone 3
CloudNativePG
Kubernetes cluster


myapp-db-rw
Service
myapp-db-ro
Service
my-app-db-1
Pod
PRIMARY
pgdata
my-app-db-1
PVC
Node
my-app-db-2
Pod
PRIMARY
pgdata
my-app-db-2
PVC
Node
my-app-db-3
Pod
pgdata
my-app-db-3
PVC
Node
Availability zone 1
Availability zone 2
Availability zone 3
CloudNativePG
Kubernetes cluster

Part 3 - The importance of storage

CSI and Database

How many copies of the data are you writing at the CNPG layer and the CSI layer and the Block storage layer.

- Is there RAID or zonal/regional replication at the block storage layer?
- Do you have replication at the CSI layer?
- Are you replicating at the PostgreSQL layer?

Block replication - 2 Datacentres
CSI replication - e.g. Ondat Master + 2 Replicas
3 way PostgreSQL replication

…. 18 copies of the data persisted to disk!

Single cluster - storage replication


```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: ondat-replicated-tap-encrypted
provisioner: csi.storageos.com
allowVolumeExpansion: true
parameters:
 csi.storage.k8s.io/fstype: xfs
 storageos.com/replicas: "2"
 storageos.com/encryption: "true"
 storageos.com/topology-aware: "true"
 csi.storage.k8s.io/secret-name: storageos-api
 csi.storage.k8s.io/secret-namespace: storageos

# Create 2 replica volumes.
# Enable volume encryption - push to Hashicorp Vault or
other KMS with https://github.com/ondat/trousseau
# Enable TAP (default looks for
"topology.kubernetes.io/zone=" on nodes)
```

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: ondat-replicated-tap-encrypted
provisioner: csi.storageos.com
allowVolumeExpansion: true
parameters:
 csi.storage.k8s.io/fstype: xfs
 csi.storage.k8s.io/secret-name: storageos-api
 csi.storage.k8s.io/secret-namespace: storageos

# Simple orchestration of any storage
# https://github.com/ondat/discoblocks
```

Restore points

How are you going to RESTORE your database?
For minimum RTO/RPO and ability to restore to a Point in Time - you can stream the Write Ahead Logs to an S3 store.
For an application which is made up of many different containers (including CNPG deployment) do you need to orchestrate concurrent point in time backups and “quiesce” across multiple namespaces/deployments/pods?

- online, zero downtime, point in time restore BUT you have to manage per Postgres database - WAL archive
- application pause/freeze, can be orchestrated using a manager like Kasten or CloudCasa across many clusters/namespaces/pods - CSI snapshots.

Backups - database versus cluster


```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
…
 backup:
   barmanObjectStore:
     destinationPath: s3://cluster-example-full-backup/
     endpointURL: http://custom-endpoint:1234
     s3Credentials:
       inheritFromIAMRole: false
       accessKeyId:
         name: backup-creds
         key: ACCESS_KEY_ID
       secretAccessKey:
         name: backup-creds
         key: ACCESS_SECRET_KEY
     wal:
       compression: gzip
       encryption: AES256
     data:
       compression: gzip
       encryption: AES256
       immediateCheckpoint: false
       jobs: 2
   retentionPolicy: "30d"
```

```yaml
apiVersion: v1
items:
- apiVersion: snapshot.storage.k8s.io/v1
 deletionPolicy: Delete
 driver: csi.storageos.com
 kind: VolumeSnapshotClass
 metadata:
   annotations:
     k10.kasten.io/is-snapshot-class: "true"
   generation: 1
   labels:
     app: storageos
     app.kubernetes.io/component: volumesnapshotclass
   name: kasten-csi-storageos-com
 parameters:
   csi.storage.k8s.io/snapshotter-list-secret-name: storageos-api
   csi.storage.k8s.io/snapshotter-list-secret-namespace: storageos
   csi.storage.k8s.io/snapshotter-secret-name: storageos-api
   csi.storage.k8s.io/snapshotter-secret-namespace: storageos
kind: List
metadata:
 resourceVersion: ""
```

Part 4 - Demo

```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: replicated-db
spec:
  instances: 3

  affinity:
    topologyKey: topology.kubernetes.io/zone

  postgresql:
    parameters:
      shared_buffers: "1GB"

  resources:
    requests:
      memory: "4Gi"
      cpu: 8
    limits:
      memory: "4Gi"
      cpu: 8

  minSyncReplicas: 1
  maxSyncReplicas: 1


  storage:
    storageClass: storageos-az-enc
    size: 50Gi

  walStorage:
    storageClass: standard-az-enc
    size: 5Gi
```

Demo

Content

Part 5 - Conclusions

Conclusion

Freedom
Own your own data
Cost optimisation
Dev Ops

KubeCon | CloudNativeCon
North America 2022

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

Please scan the QR Code above to
leave feedback on this session