# PostgreSQL in the Cloud: DBaaS vs Kubernetes

**Michał Nosek**
Enterprise Architect @ Percona

# Agenda

1. Cloud-native Applications and Kubernetes
2. Options to run PostgreSQL
3. Comparison: Google CloudSQL PostgreSQL vs K8s with Percona Operator

# Cloud-native Applications and Kubernetes

“Cloud native is an approach to building and running applications that fully exploit the advantages of the cloud computing model.”

Source: Pivotal (VMWare)

“Cloud native computing uses an open source software stack to be:

– **Containerized**. Each part (applications, processes, etc.) is packaged in its own container. This facilitates reproducibility, transparency, and resource isolation.

– **Dynamically orchestrated.** Containers are actively scheduled and managed to optimize resource utilization.

– **Microservices-oriented.** Applications are segmented into microservices. This significantly increases the overall agility and maintainability of applications.”

Source: CNCF

# Cloud-native application

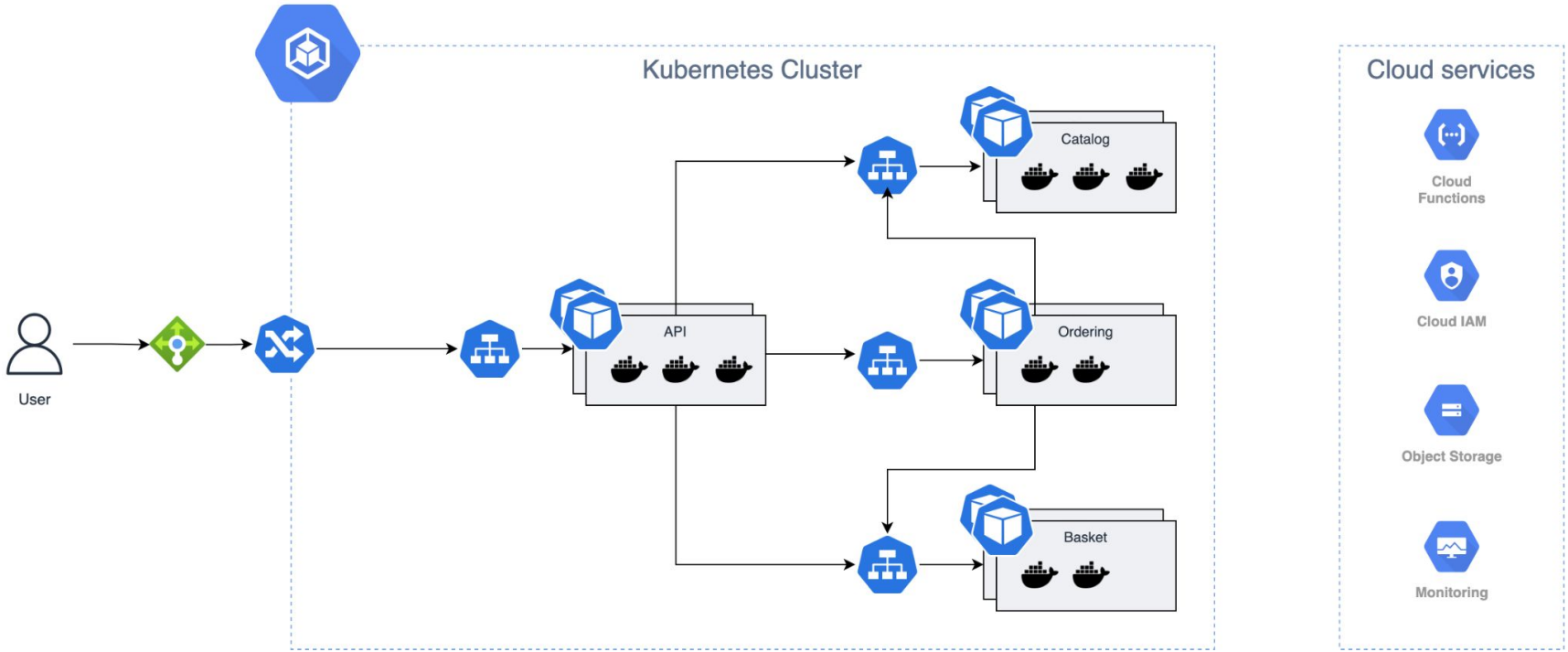
Microservices

Cloud-based

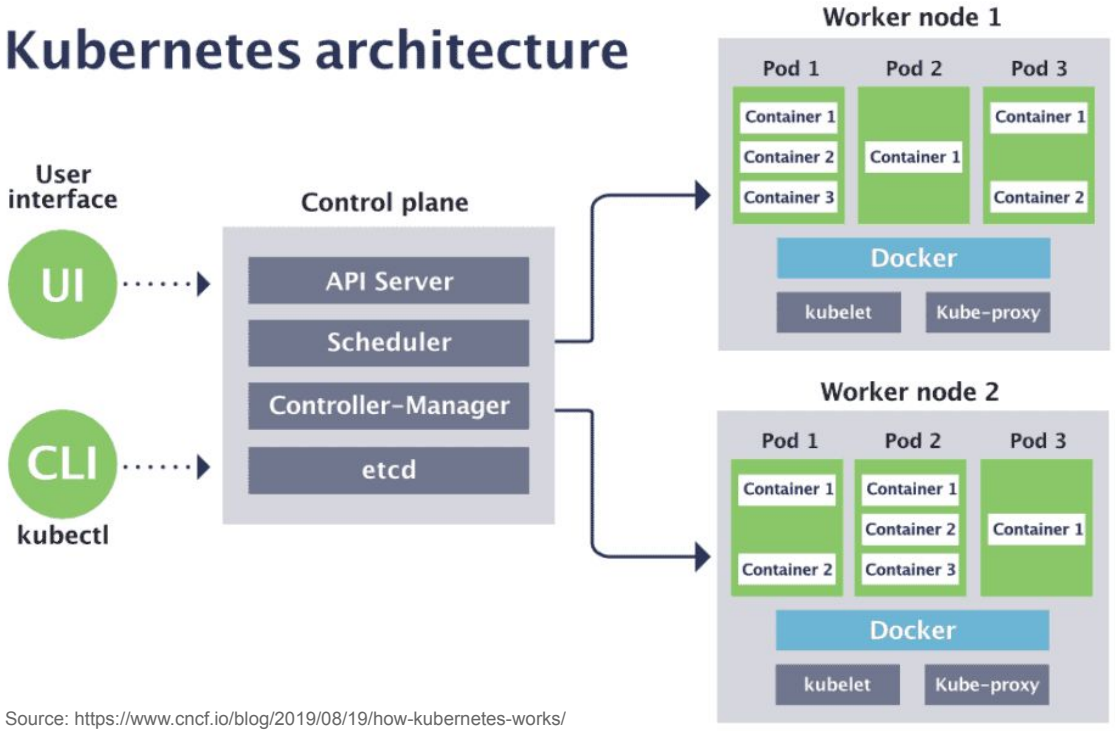
Kubernetes

Data store per service

# Cloud-native application

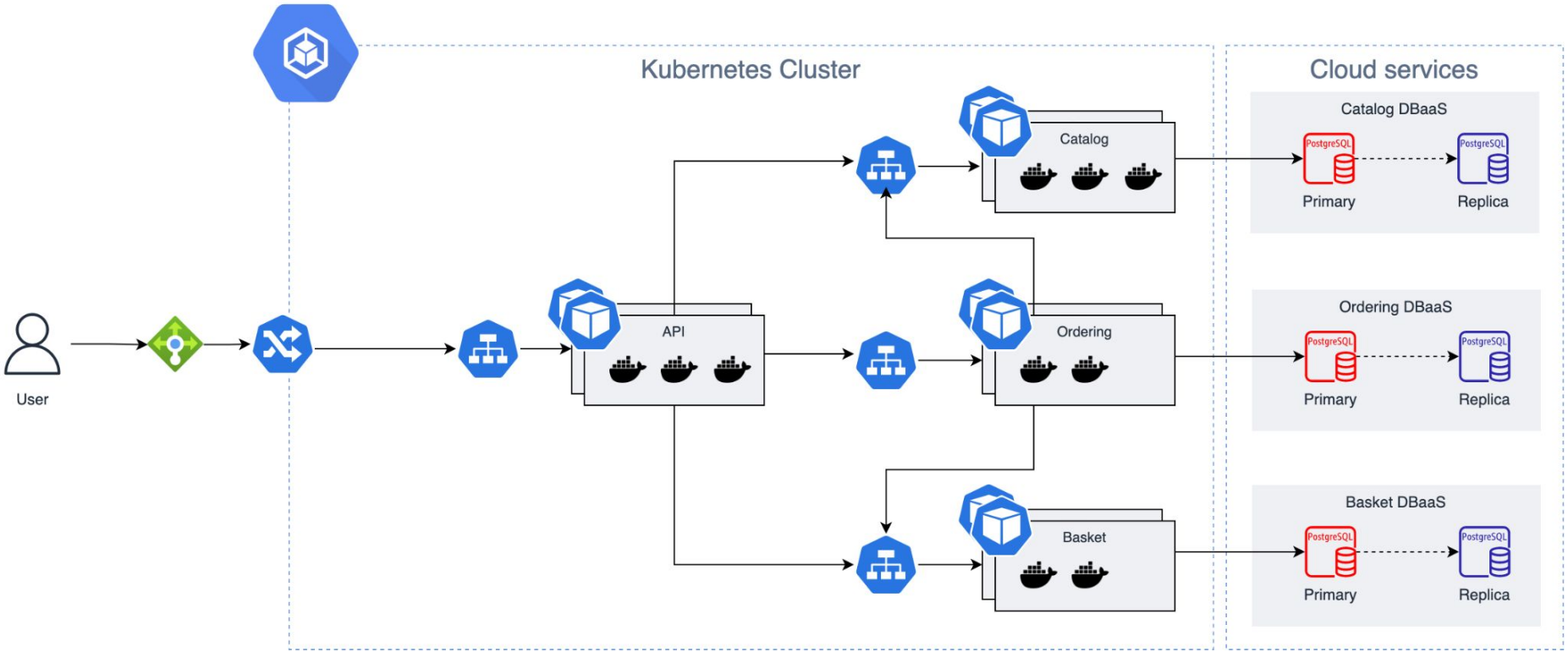


# How Kubernetes works?

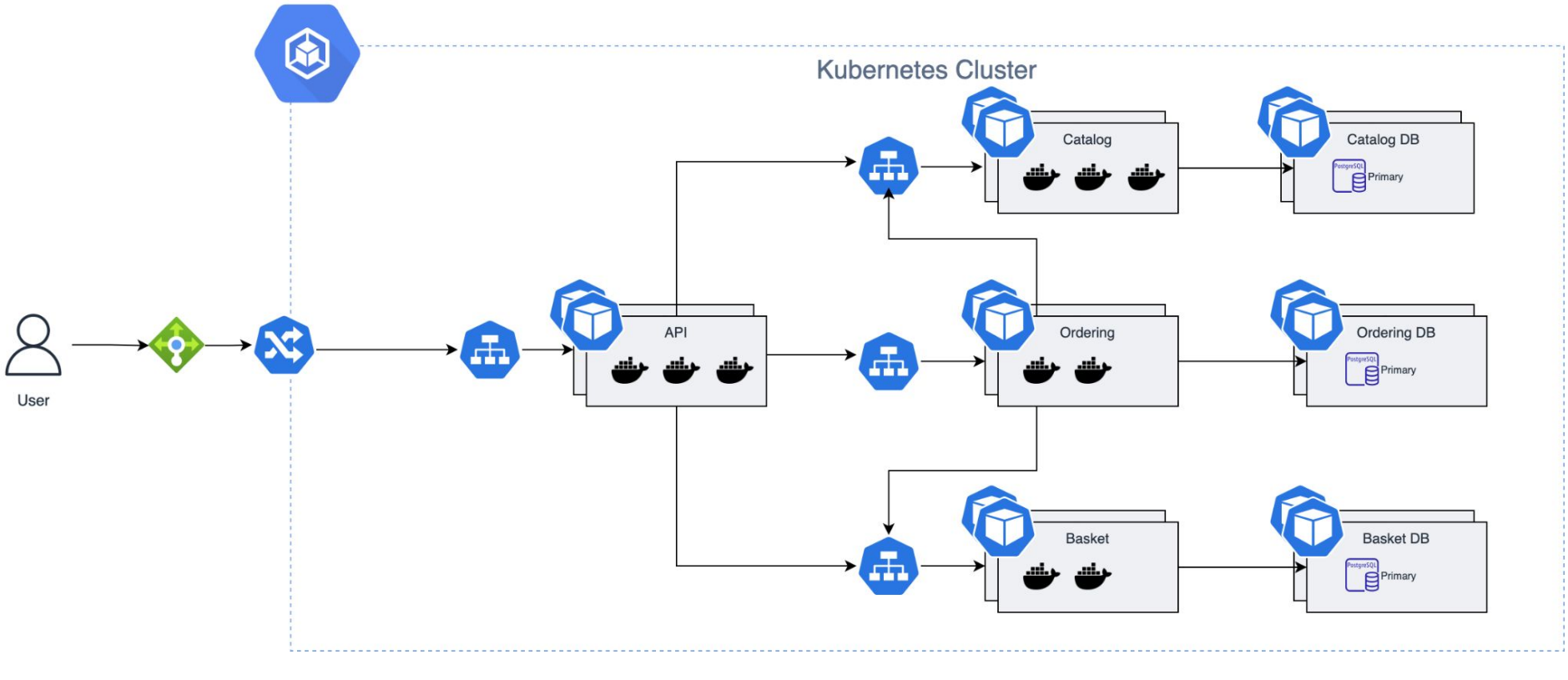



Source: https://www.cncf.io/blog/2019/08/19/how-kubernetes-works/

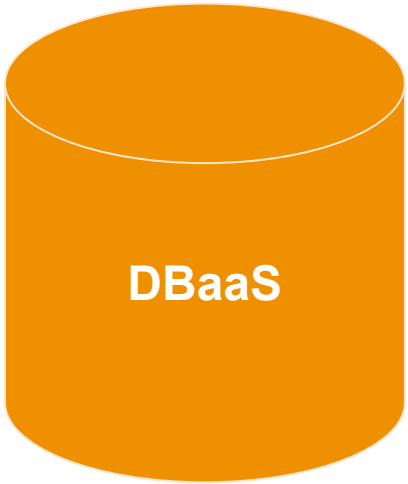
# Cloud-native application



# Cloud-native application



# Two options




# DBaaS Options

1. Cloud-vendor provided
    a. Google CloudSQL PostgreSQL
    b. AWS RDS (Aurora) PostgreSQL
    c. Azure Database for PostgreSQL

2. Cloud independent
    a. Crunchy Bridge
    b. EDB BigAnimal
    c. Aiven

# DBaaS value prop

- Infrastructure is managed for you
- It’s easy:
  - Automated deployment (day “1” operations)
  - Automated DBA tasks (day”2” operations)

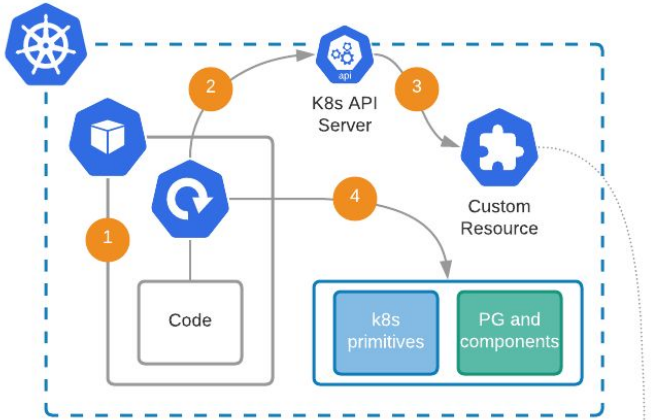
# PostgreSQL on Kubernetes

- One platform for everything
- No vendor lock-in, portability
- Cost-efficient
- ~~It's easy~~

1. Kubernetes
2. StatefulSet for Primary/Replicas
3. DB Replication
4. High Availability and failover
5. Storage (PVC/Hostpath)
6. Services
   a. Read/Write
   b. Read Only
7. pgBouncer (with HA)
8. Configuration
9. Monitoring agents
10. Backups
11. Updates
12. DR
13. SSL
14. …..

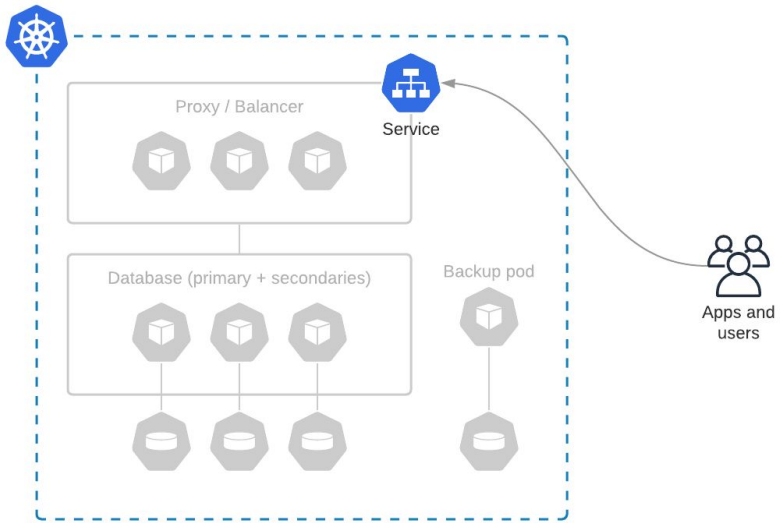
# Operators

1. Operator is a Pod + Custom Resource Definition
2. Reconcile loop checks k8s API for Custom Resource (CR)
3. k8s controllers are aware of CR creation or changes
4. Operator's code provisions and manages
    a. k8s primitives
    b. DB and components configuration




```
apiVersion: pg.percona.com/v1
kind: PerconaPGCluster
metadata:
  name: my-cluster
spec:
  pgPrimary:
    image: percona/...:1.0.0
    ...
  pgReplicas:
    hotStandby:
      size: 2
      resources:
        requests:
          cpu: "4"
          memory: "8Gi"
  backup:
    ...
```

- One platform for everything
- No vendor lock-in, portability
- Cost-efficient
- **It's easy**
  - **Automated deployment (day "1" operations)**
  - **Automated DBA tasks (day"2" operations)**



# Kubernetes Operators

1. Zalando
2. KubeDB
3. StackGres
4. EDB Cloud Native PostgreSQL
5. Crunchy Data PostgreSQL Operator
6. Percona Operator for PostgreSQL


VS

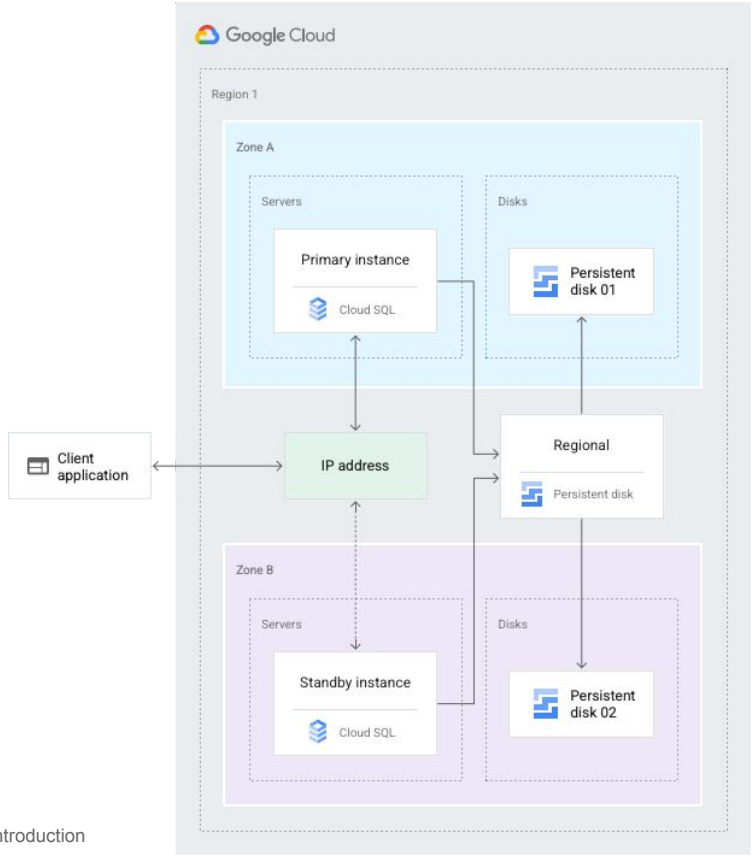
PERCONA DISTRIBUTION
FOR POSTGRESQL OPERATOR

1. Architecture
2. How to deploy it?
3. Connectivity
4. High Availability
5. Disaster Recovery
6. Updates
7. Scaling
8. Costs

# 1 - Architecture

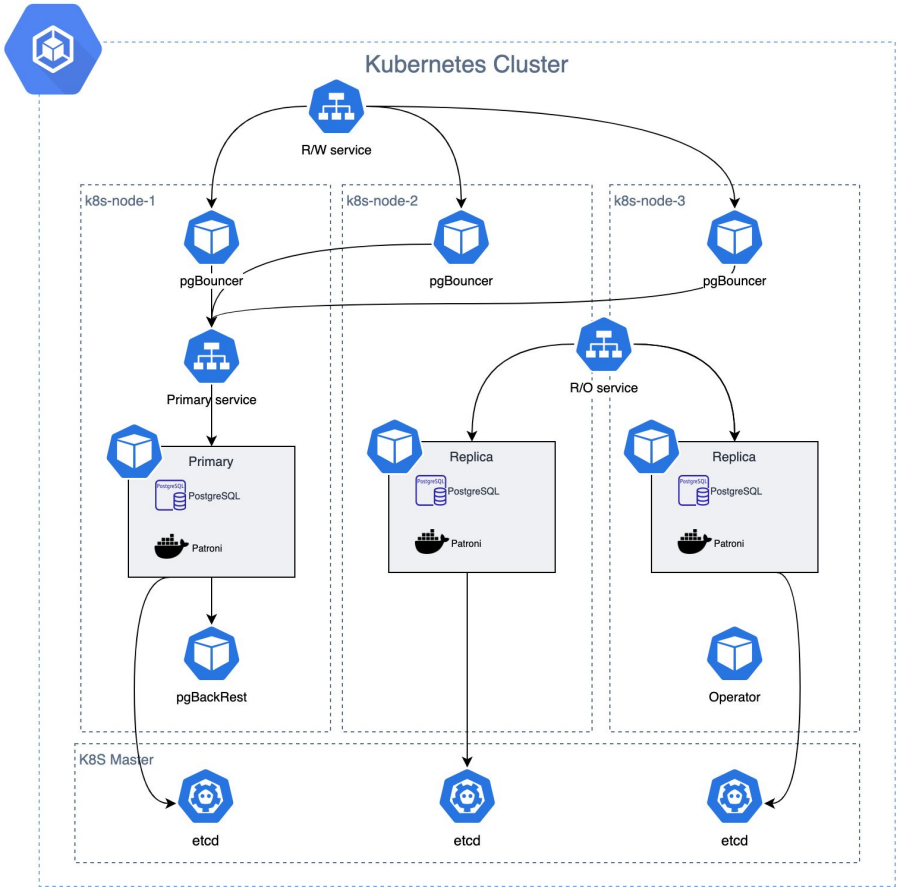
Google Cloud SQL PostgreSQL

# 1. Architecture




Source: https://cloud.google.com/sql/docs/postgres/introduction

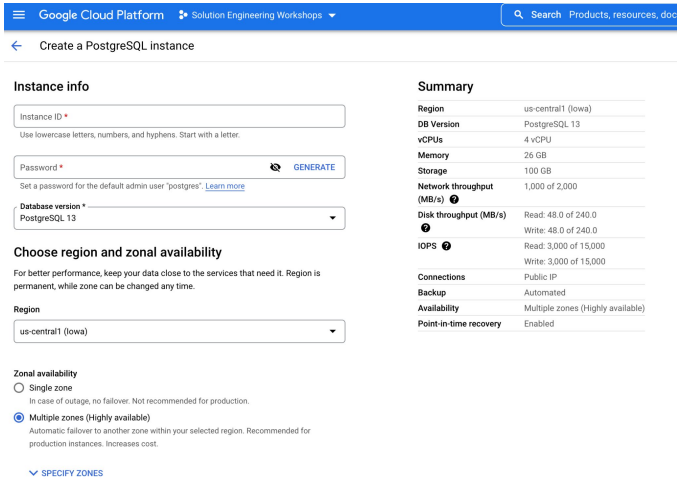
# 1. Architecture



# 2 - How to deploy it

Google Cloud SQL PostgreSQL

# 2. How to deploy it




## gcloud API

```
gcloud sql instances create INSTANCE
    [--activation-policy=ACTIVATION_POLICY] [--[no-]assign-ip] [--async]
    [--authorized-networks=NETWORK,[NETWORK,...]]
    [--availability-type=AVAILABILITY_TYPE] [--no-backup]
    [--backup-location=BACKUP_LOCATION]
    [--backup-start-time=BACKUP_START_TIME] [--cpu=CPU]
    [--database-flags=FLAG=VALUE,[FLAG=VALUE,...]]
    [--database-version=DATABASE_VERSION] [--enable-bin-log]
    [--failover-replica-name=FAILOVER_REPLICA_NAME]
    [--maintenance-release-channel=MAINTENANCE_RELEASE_CHANNEL]
    [--maintenance-window-day=MAINTENANCE_WINDOW_DAY]
    [--maintenance-window-hour=MAINTENANCE_WINDOW_HOUR]
    [--master-instance-name=MASTER_INSTANCE_NAME] [--memory=MEMORY]
    [--replica-type=REPLICA_TYPE] [--replication=REPLICATION]
    [--require-ssl] [--root-password=ROOT_PASSWORD]
    [--storage-auto-increase] [--storage-size=STORAGE_SIZE]
    [--storage-type=STORAGE_TYPE] [--tier=TIER, -t TIER]
    [--disk-encryption-key=DISK_ENCRYPTION_KEY
      : --disk-encryption-key-keyring=DISK_ENCRYPTION_KEY_KEYRING
      --disk-encryption-key-location=DISK_ENCRYPTION_KEY_LOCATION
      --disk-encryption-key-project=DISK_ENCRYPTION_KEY_PROJECT]
    [--region=REGION; default="us-central" | --gce-zone=GCE_ZONE
      | --zone=ZONE] [GCLOUD_WIDE_FLAG ...]
```

# 2. How to deploy it

Prerequisites:

- GKE Cluster
- kubectl configured
- Cloud IAM configured
- Namespace created

Step 1: Deploy the Operator

```
$ git clone -b v1.1.0
https://github.com/percona/percona-postgresql-operator

$ kubectl apply -f deploy/operator.yaml
```

Result:

```
percona@Demo:~$kubectl get pods

NAME                                  READY   STATUS      RESTARTS   AGE
postgres-operator-9b46c97fb-5bgr9     4/4     Running     1          121m
```

# 2. How to deploy it

Step 2: Edit cr.yaml - your cluster configuration

```
apiVersion: pg.percona.com/v1
kind: PerconaPGCluster
metadata:
  name: cluster1
spec:
  database: pgdb
  port: "5432"
  user: pguser

  pgPrimary:
    image: percona/postgresql-ppg14-ha
    resources:
      requests:
        memory: "128Mi"
    volumeSpec:
      size: 1G
```

```
pgReplicas:
  hotStandby:
    size: 2
    resources:
      requests:
        memory: "128Mi"
    volumeSpec:
      size: 1G
    enableSyncStandby: false

pgBouncer:
  Image: percona/postgresql-ppg14-pgbouncer
  size: 3
  resources:
    requests:
      cpu: "1"
      memory: "128Mi"
```

# 2. How to deploy it

## Step 3: Deploy the Cluster

```
$ kubectl apply -f deploy/cr.yaml
```

Result:

```
percona@Demo:~$kubectl get all

NAME                                               READY   STATUS      RESTARTS   AGE
pod/cluster1-dccb948b6-jr6m2                       1/1     Running     0          19m
pod/cluster1-repl1-788bb496c7-fw9xc                1/1     Running     0          18m
pod/cluster1-repl2-5b7f64b979-7pb84                1/1     Running     0          18m
pod/cluster1-pgbouncer-6df59cdd8c-5646b            1/1     Running     0          18m
pod/cluster1-pgbouncer-6df59cdd8c-cst64            1/1     Running     0          18m
pod/cluster1-pgbouncer-6df59cdd8c-zjqb4            1/1     Running     0          18m
pod/cluster1-backrest-shared-repo-fb7db8979-8q4d5  1/1     Running     0          19m

NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
service/cluster1                    ClusterIP   10.27.251.206   <none>        2022/TCP,5432/TCP
service/cluster1-pgbouncer          ClusterIP   10.27.254.8     <none>        5432/TCP
service/cluster1-replica            ClusterIP   10.27.240.212   <none>        2022/TCP,5432/TCP
```

# 3 - Connectivity

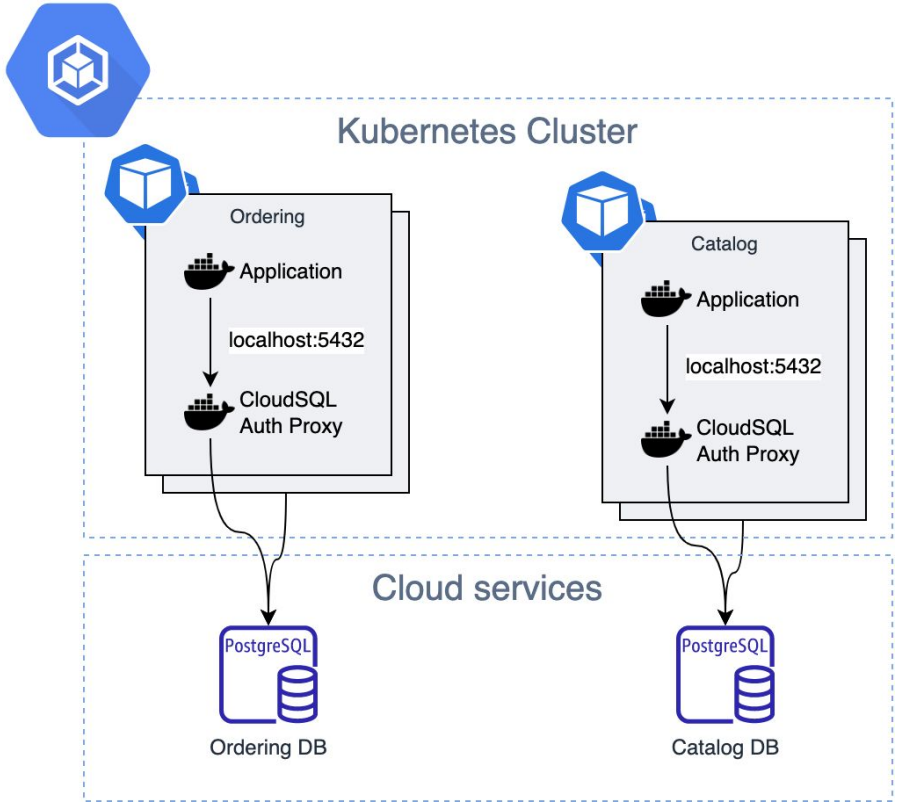
# 3. Connectivity

```
postgresql://[user[:password]@][host][:port][/dbname]
```

# 3. Connectivity

CloudSQL Auth Proxy running as a sidecar container:

- Simple connectivity
- Security
- Automatic IAM Authentication
- Prevents SPOF
- Better resource utilization
- Application-specific identity



Google Cloud SQL PostgreSQL

# 3. Connectivity

Prerequisites:

- Instance connection name
- Google Service Account
- CloudSQL Admin API
- Same VPC

## Step 1: Create KSA

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: <YOUR-KSA-NAME>
```

## Step 2: Bind GSA with KSA

```
gcloud iam service-accounts add-iam-policy-binding \
--role="roles/iam.workloadIdentityUser" \
--member="serviceAccount:YOUR-GCP-PROJECT.svc.id[YOUR-K8S-NAMESPACE/YOUR-KSA-NAME]" \
YOUR-GSA-NAME@YOUR-GCP-PROJECT.iam.gserviceaccount.com

kubectl annotate serviceaccount YOUR-KSA-NAME \
iam.gke.io/gcp-service-account=YOUR-GSA-NAME@YOUR-GCP-PROJECT.iam.gserviceaccount.com
```

# 3. Connectivity

Step 3: Add the sidecar to your application pod

```
...
spec:
  template:
    spec:
      serviceAccountName: <YOUR-KSA-NAME>
      containers:
      - name: <YOUR-APPLICATION-NAME>
        ...
      - name: cloud-sql-proxy
        image: gcr.io/cloudsql-docker/gce-proxy:1.28.0
        command:
          - "/cloud_sql_proxy"
          - "-ip_address_types=PRIVATE"
          - "-instances=<INSTANCE_CONNECTION_NAME>=tcp:5432"
```

# 3. Connectivity

| Role name | Attributes |
|---|---|
| `postgres` | Superuser, Create role, Create DB, Replication, Bypass RLS |
| `primaryuser` | Replication |
| `pguser` | Non-privileged user |
| `pgbouncer` | Administrative user for the pgBouncer connection pooler |

Step 1: Get postgres user password from a secret

```
$ kubectl get secret cluster1-users --template={{.data.postgres}} | base64 --decode
```

Step 2: Create a user for an application by connecting to cluster1 service

```
$ kubectl run -i --rm --tty pg-client \
--image=perconalab/percona-distribution-postgresql:13.2 --restart=Never -- bash -il

[postgres@pg-client /]$ psql -h cluster1 -p 5432 -U postgres
```

# 3. Connectivity

Step 3: Create a secret to store credentials

```
apiVersion: v1
kind: Secret
metadata:
  name: postgres-credentials-secret
type: Opaque
data:
  USER_NAME: your_user
  PASSWORD: your_password
  RW_HOST: cluster1-pgbouncer
  RO_HOST: cluster1-replica
```

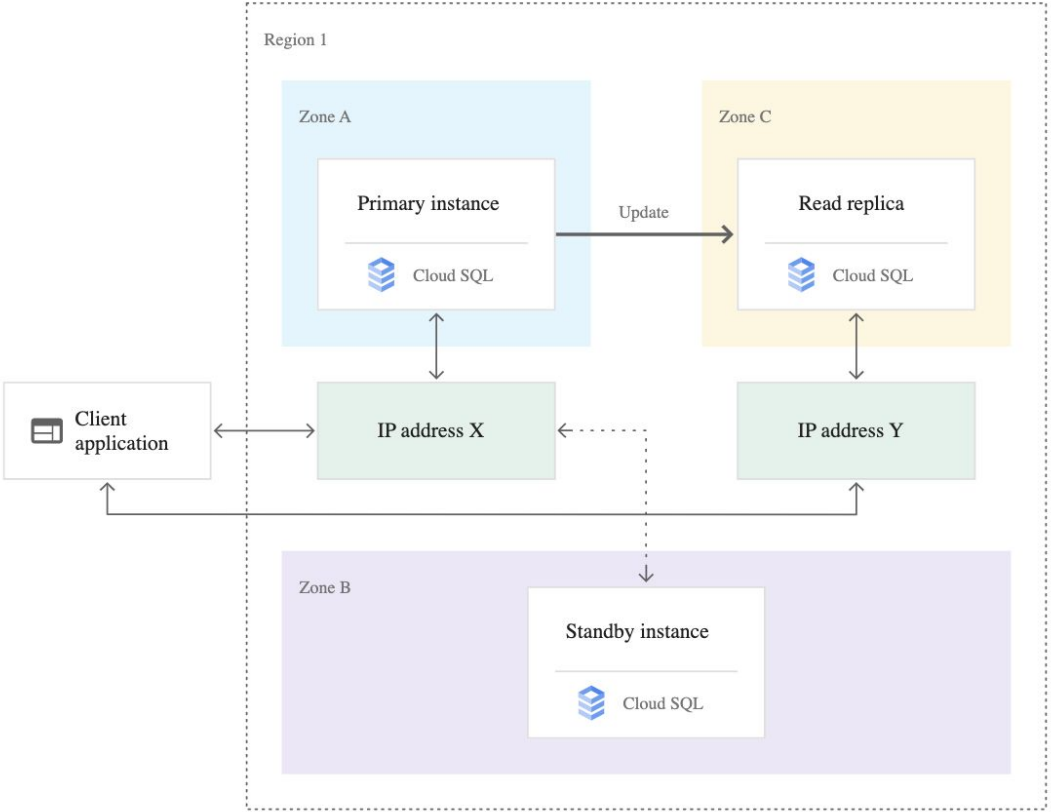
Step 4: Mount the secret to your application pod to get credentials as env variables

```
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
       envFrom:
      - secretRef:
          name: postgres-credentials-secret
```

# 4 - High Availability

Google Cloud SQL PostgreSQL

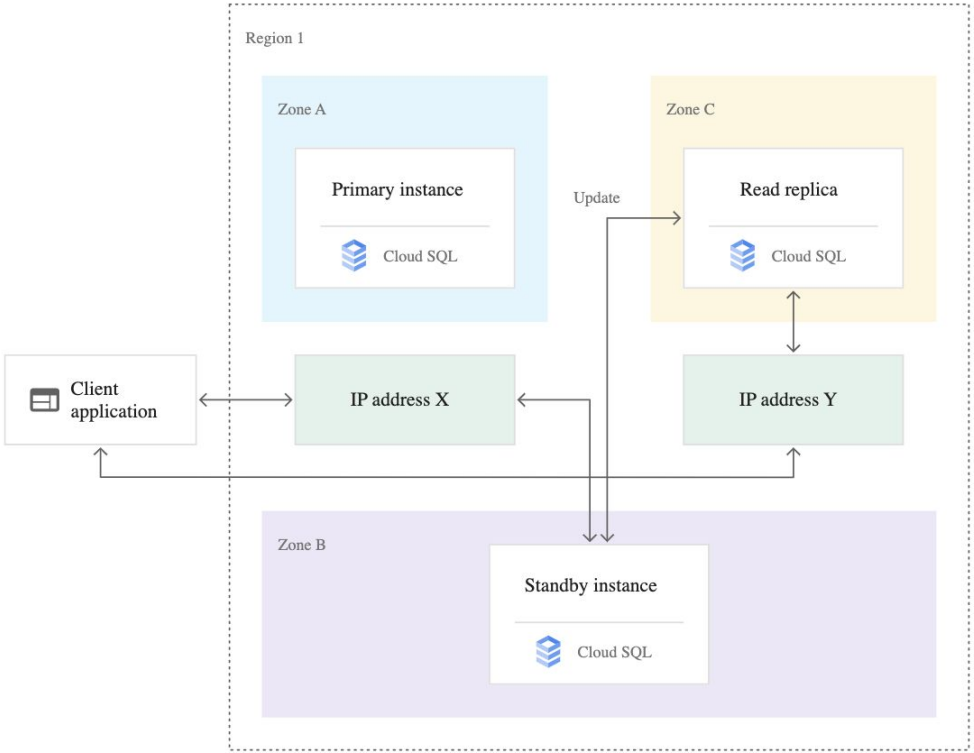
# 4. High Availability




https://cloud.google.com/sql/docs/postgres/high-availability#failover

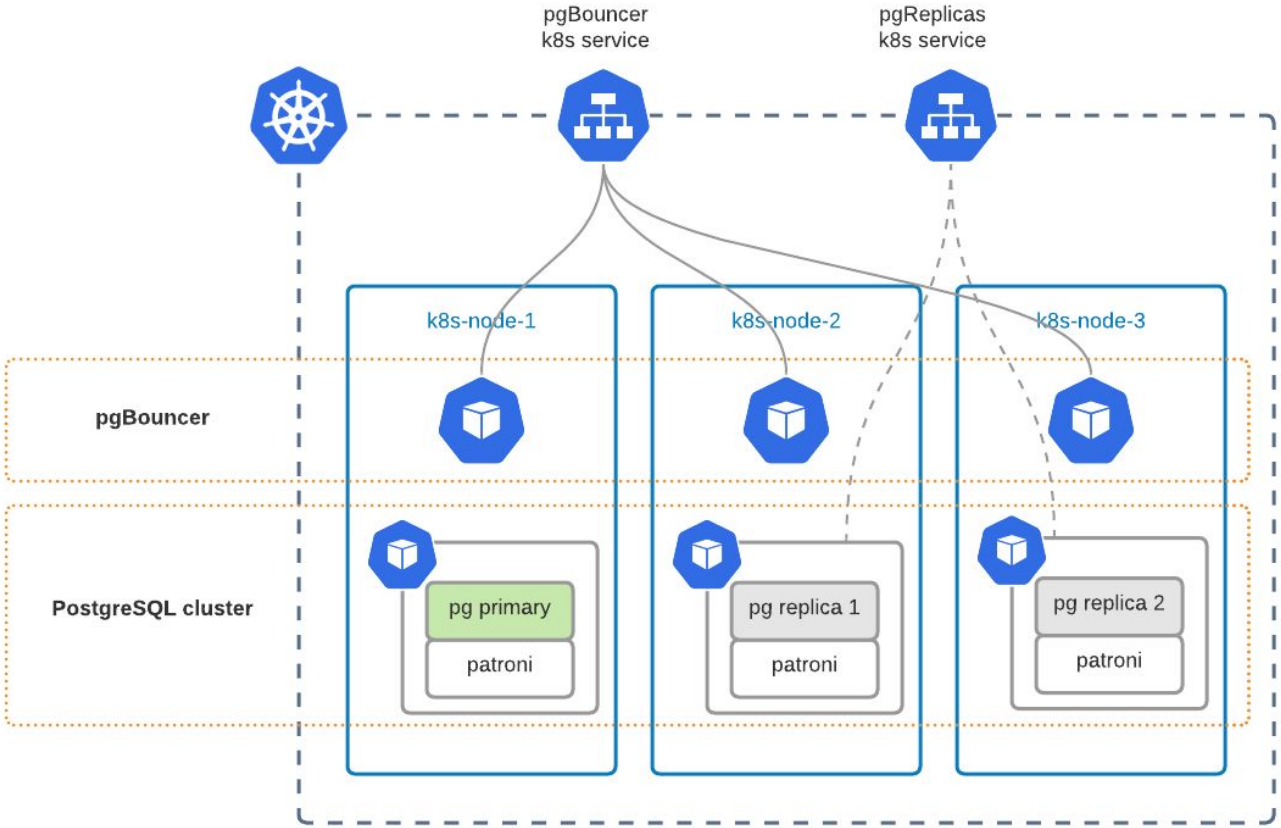
Google Cloud SQL PostgreSQL

# 4. High Availability

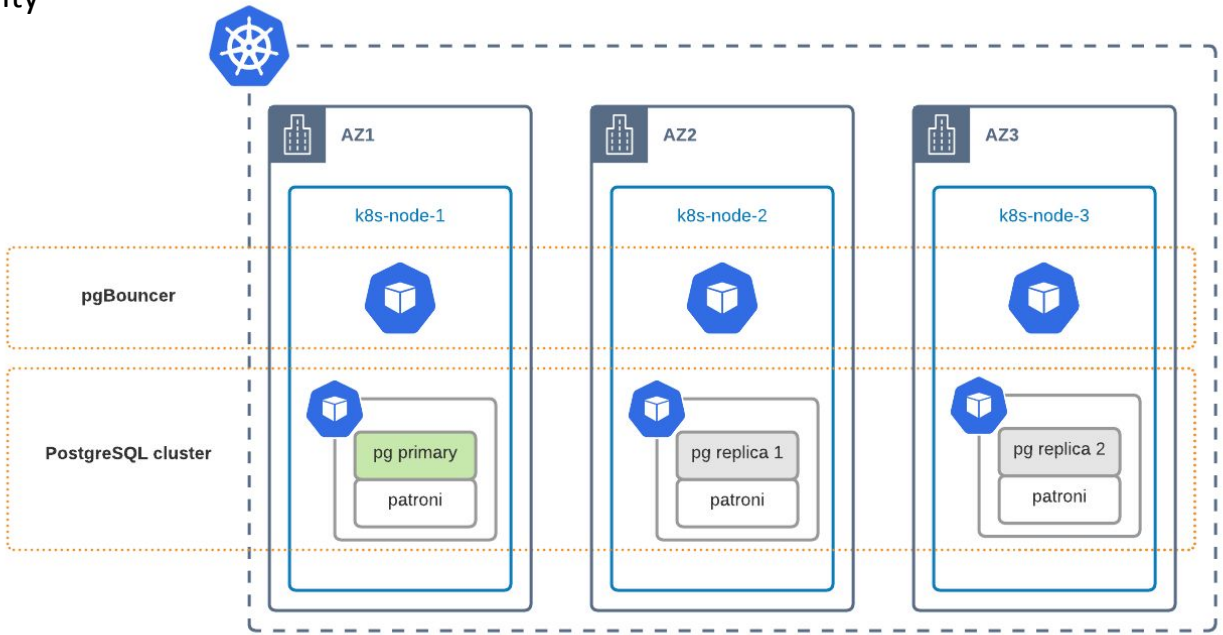



https://cloud.google.com/sql/docs/postgres/high-availability#failover

# 4. High Availability



# 4. High Availability

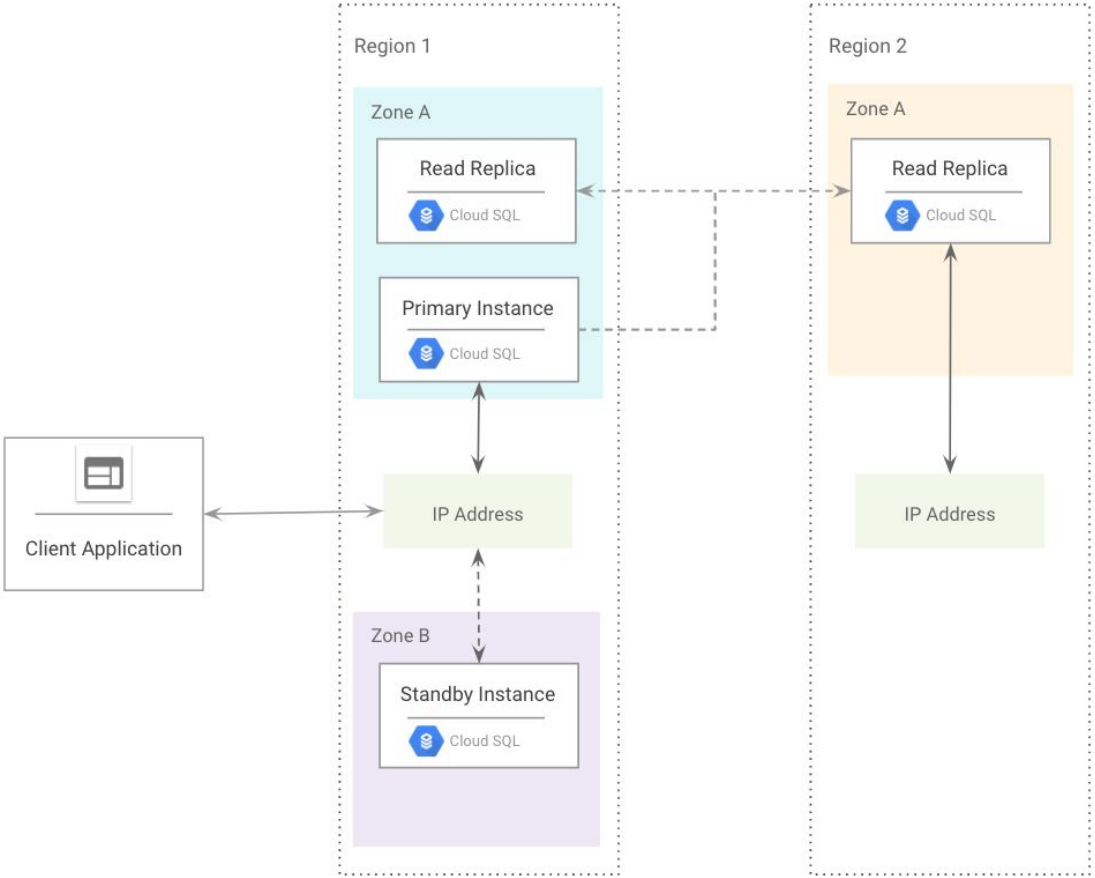



```
$ kubectl edit deploy cluster1-repl
…
-            topologyKey: kubernetes.io/hostname
+            topologyKey: topology.kubernetes.io/zone
```

# 5 - Disaster Recovery

Google Cloud SQL PostgreSQL

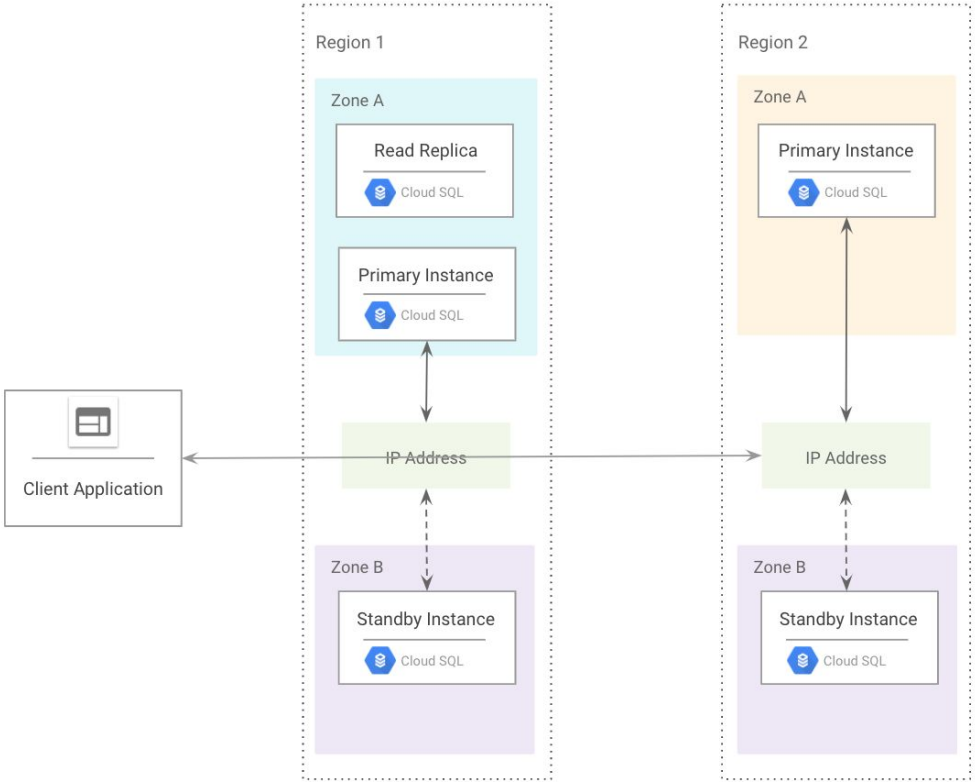
# 5. Disaster Recovery




Source: https://medium.com/google-cloud/cloud-sql-recovering-from-regional-failure-in-10-minutes-or-less-mysql-fc055540a8f0

Google Cloud SQL PostgreSQL

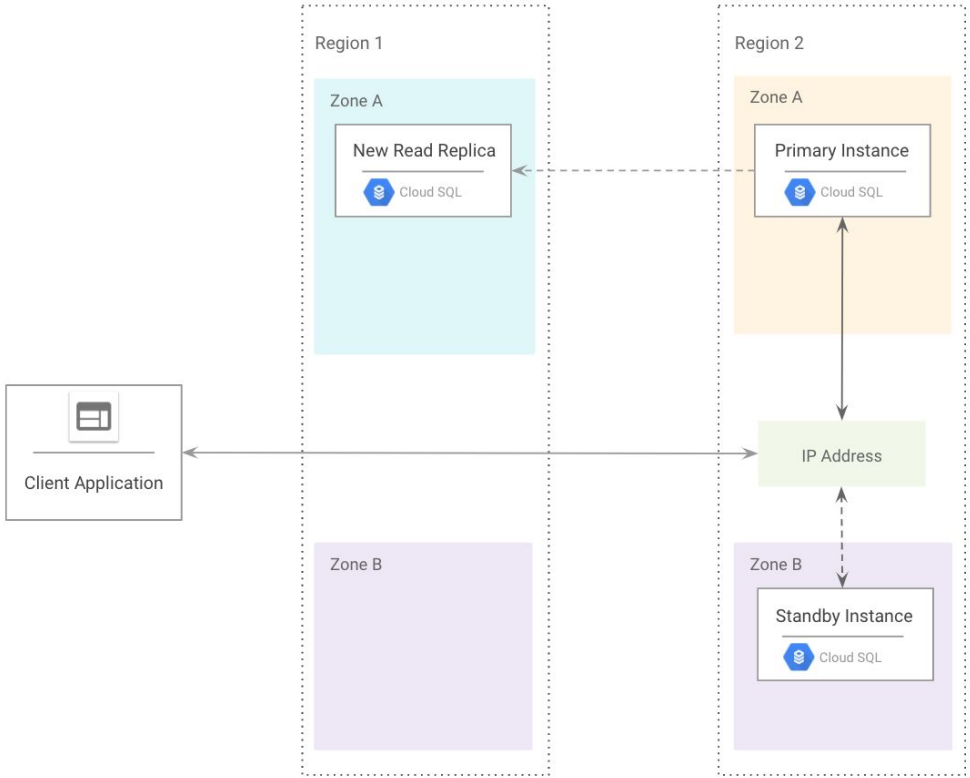
# 5. Disaster Recovery




Source: https://medium.com/google-cloud/cloud-sql-recovering-from-regional-failure-in-10-minutes-or-less-mysql-fc055540a8f0

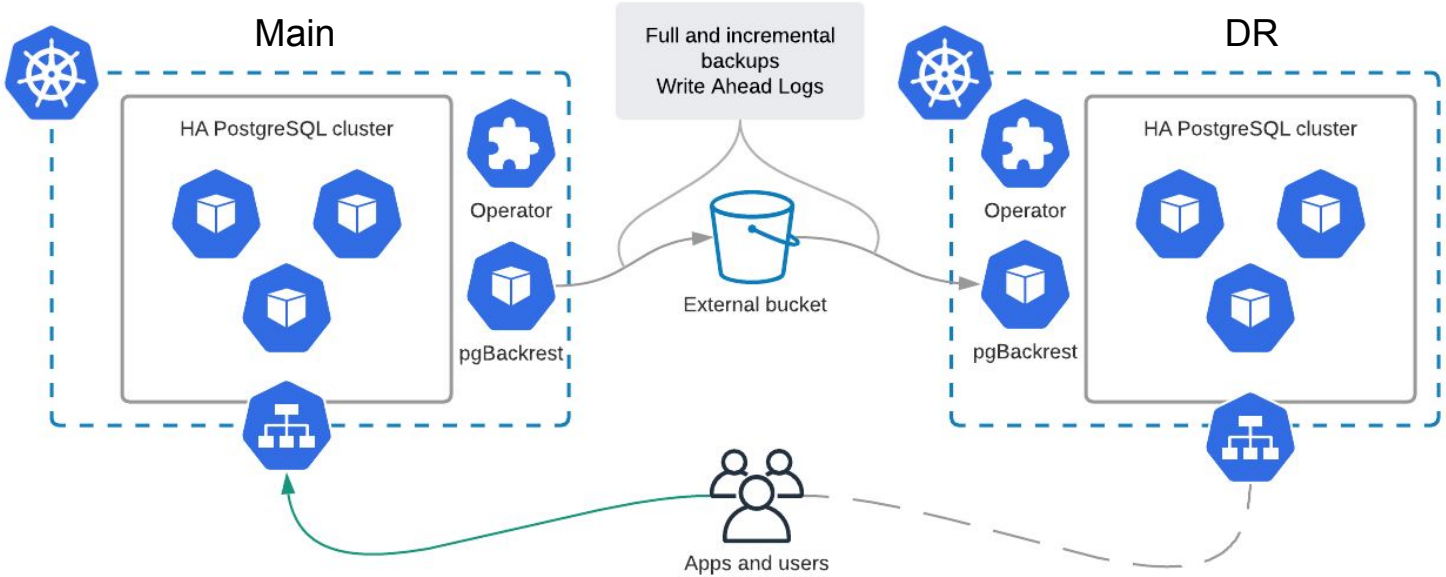
Google Cloud SQL PostgreSQL

# 5. Disaster Recovery



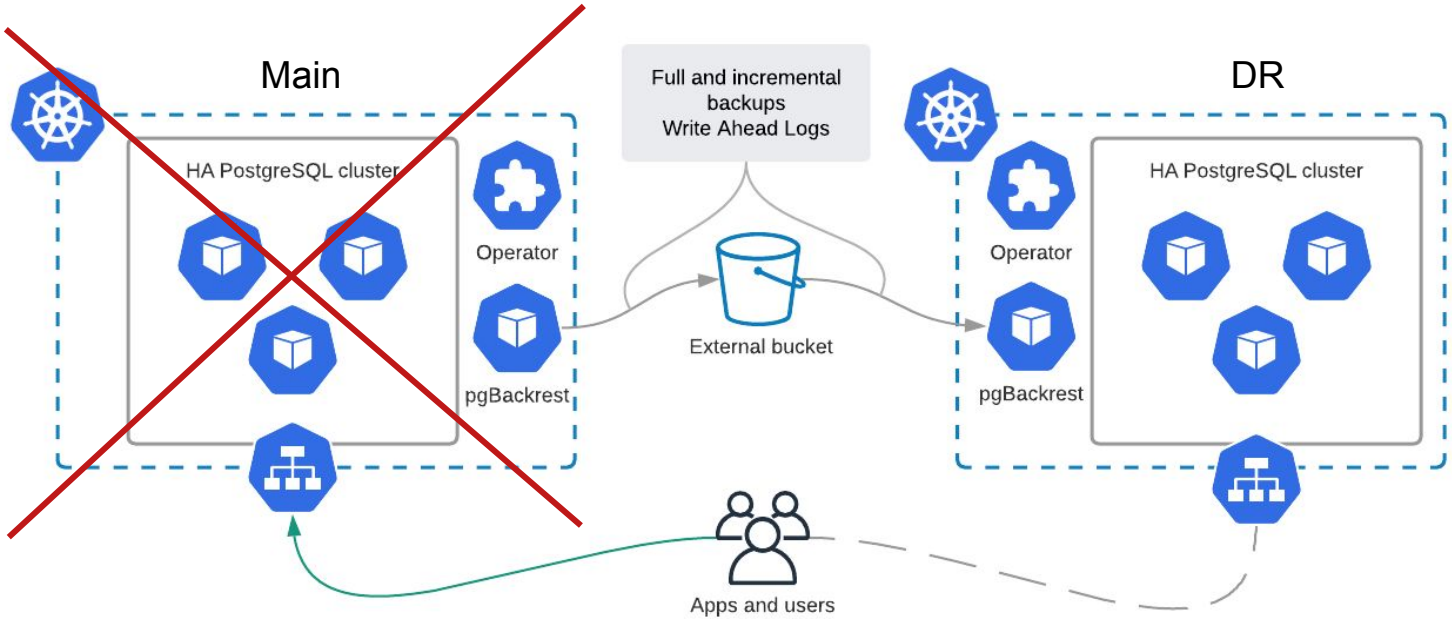

Source: https://medium.com/google-cloud/cloud-sql-recovering-from-regional-failure-in-10-minutes-or-less-mysql-fc055540a8f0

# 5. Disaster Recovery




```
spec:
  standby: true
```

# 5. Disaster Recovery

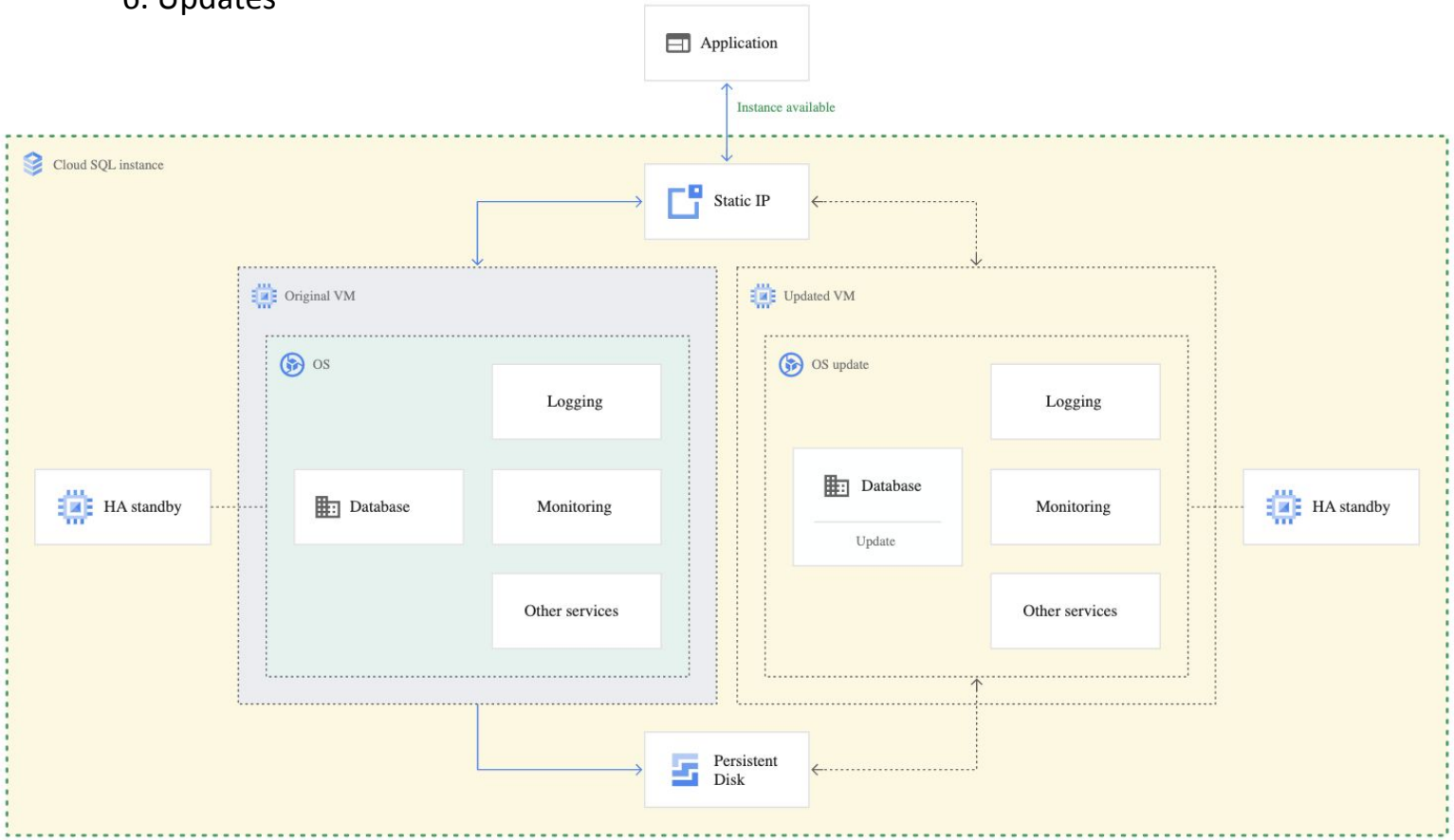



```
spec:
  standby: false
```

# 6 - Updates

Google Cloud SQL PostgreSQL

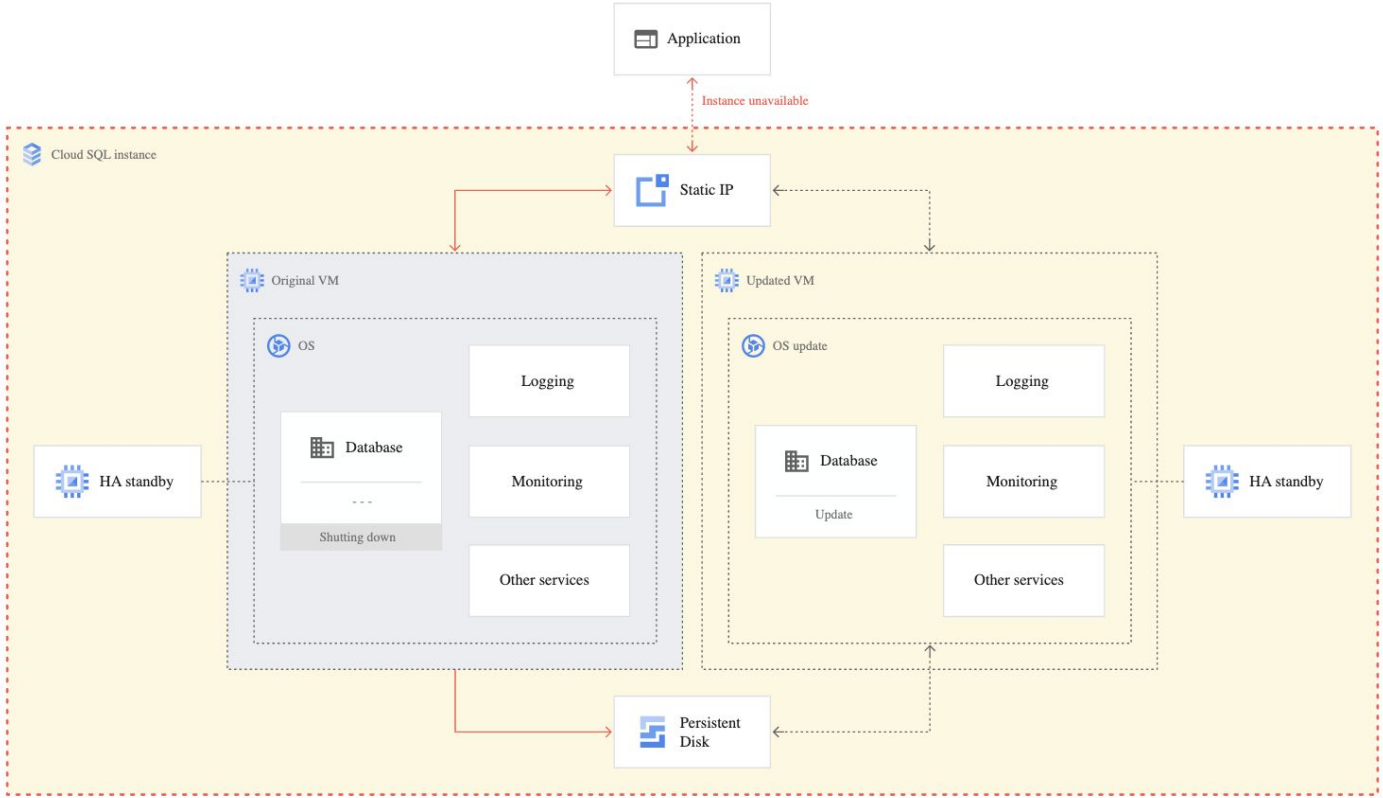
# 6. Updates




Source: https://cloud.google.com/sql/docs/postgres/maintenance

Google Cloud SQL PostgreSQL

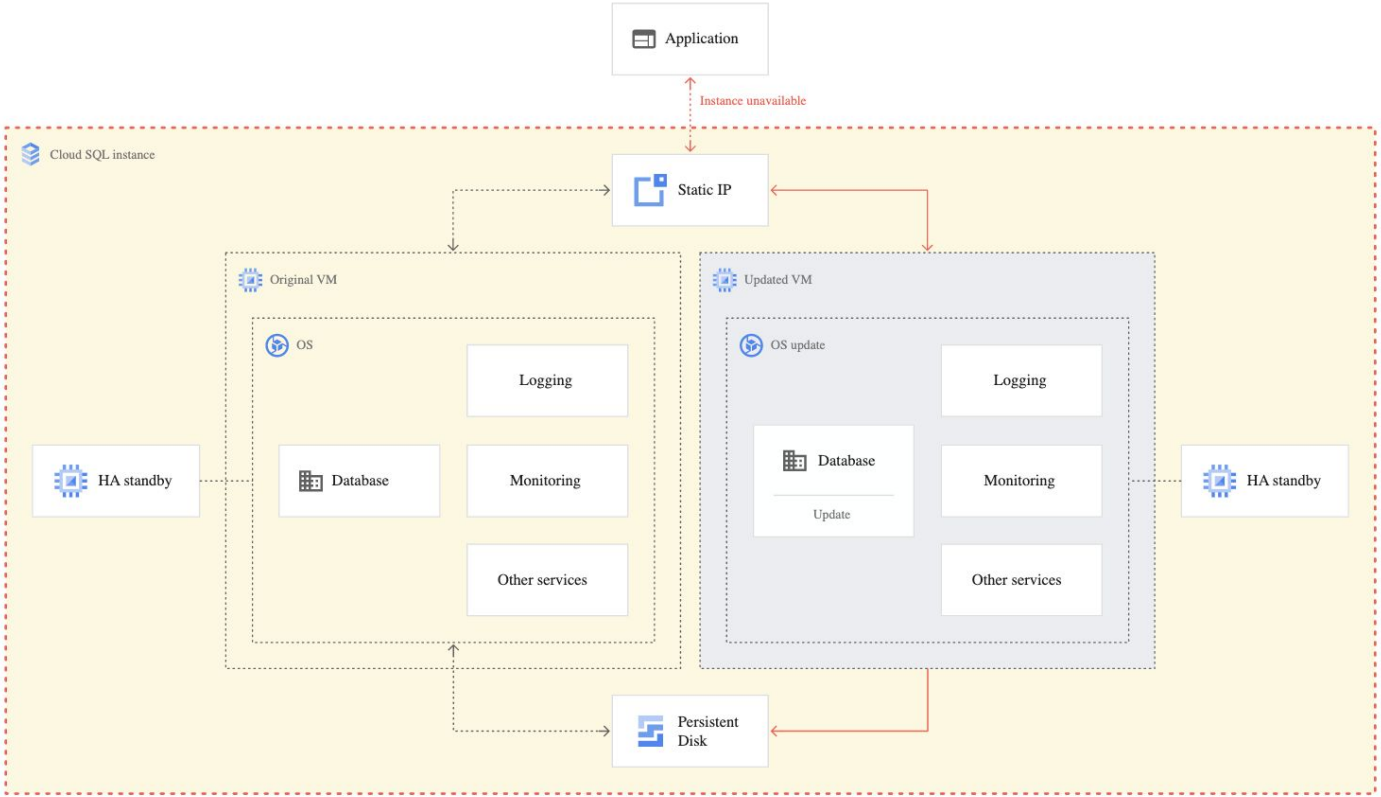
# 6. Updates




Source: https://cloud.google.com/sql/docs/postgres/maintenance

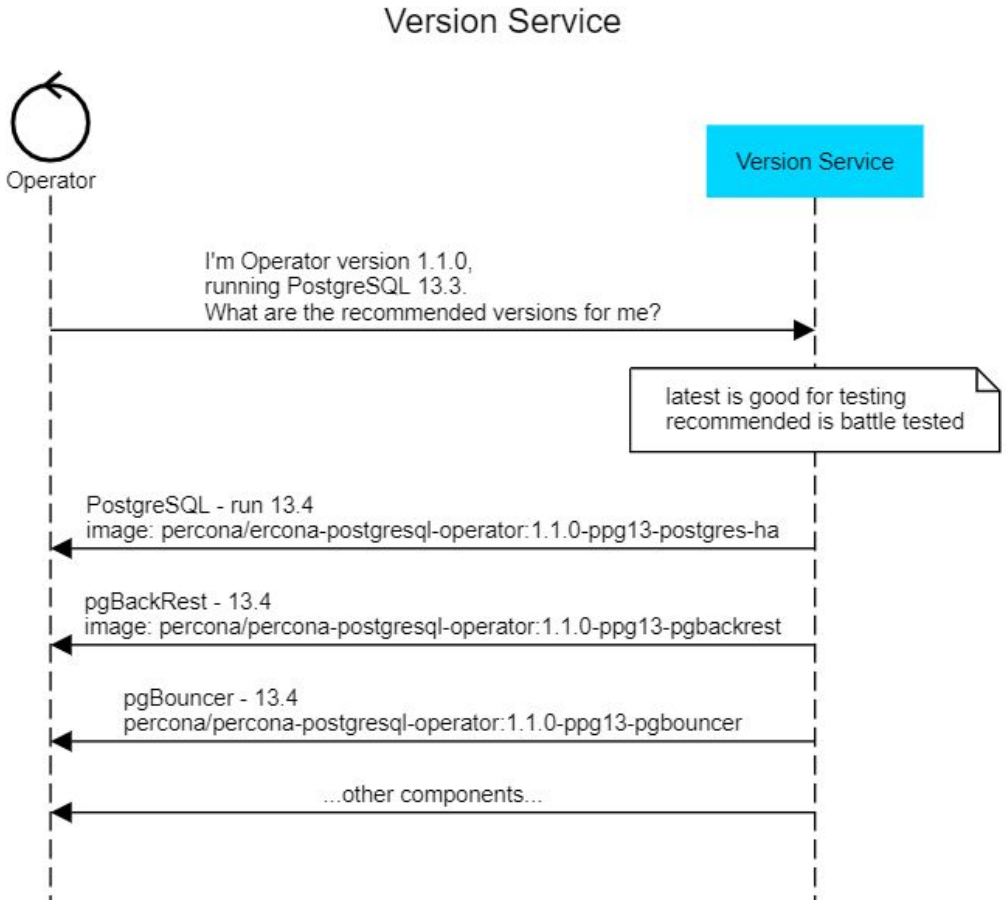
Google Cloud SQL PostgreSQL

# 6. Updates



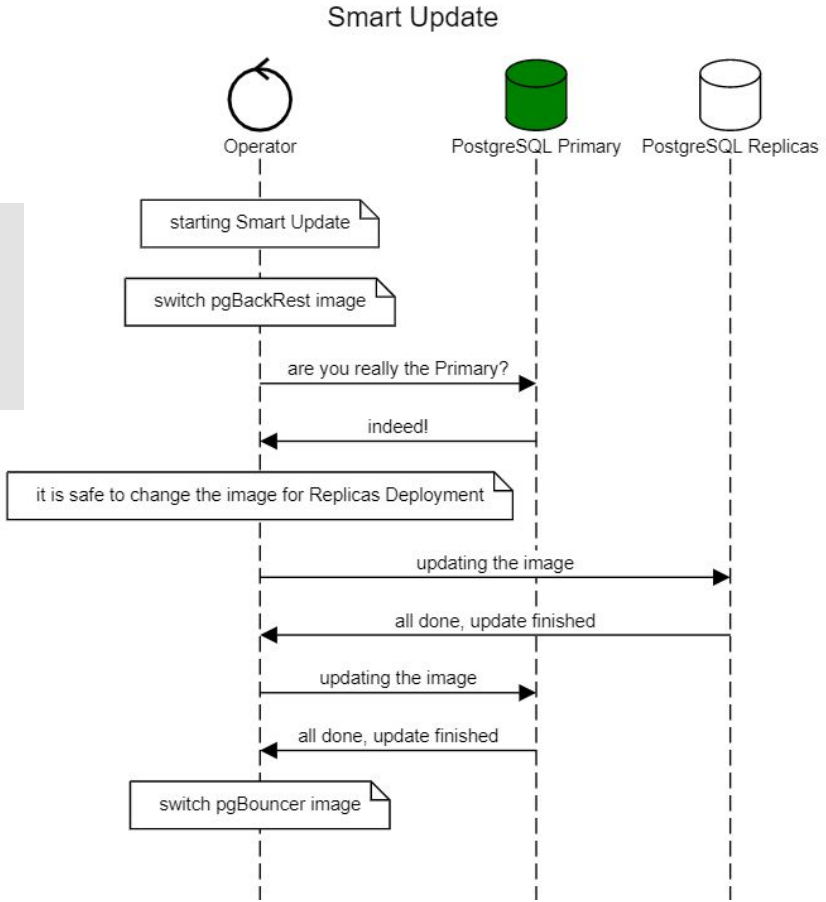

Source: https://cloud.google.com/sql/docs/postgres/maintenance

# 6. Updates


Version Service
Operator
Version Service
I'm Operator version 1.1.0,
running PostgreSQL 13.3.
What are the recommended versions for me?
latest is good for testing
recommended is battle tested
PostgreSQL - run 13.4
image: percona/ercona-postgresql-operator:1.1.0-ppg13-postgres-ha
pgBackRest - 13.4
image: percona/percona-postgresql-operator:1.1.0-ppg13-pgbackrest
pgBouncer - 13.4
percona/percona-postgresql-operator:1.1.0-ppg13-pgbouncer
...other components...

# 6. Updates

```
upgradeOptions:
  versionServiceEndpoint: https://check.percona.com
  apply: recommended | latest | version_no | disabled
  schedule: "0 2 * * *"
```



# 7 - Scalability

# 7. Scalability

Vertical

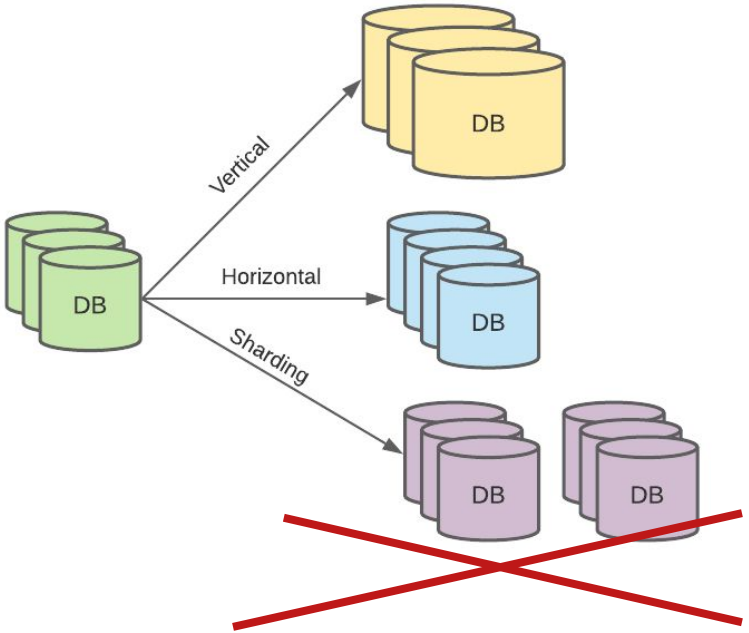
```
pgReplicas:
    hotStandby:
      resources:
        requests:
-         memory: "128Mi"
+         memory: "256Mi"
```

Horizontal - option 1

```
pgReplicas:
    hotStandby:
-       size: 2
+       size: 3
```

Horizontal - option 2

```
kubectl scale --replicas=2 perconapgcluster/cluster1
```



# 7. Scalability

## Vertical Pod Autoscaler

- Observes and recommends
- Analyses workload in long run
- Adjusts CPUs and RAM
- Limits and policies are configurable

- Pods need to be restarted
- Won’t react to short spikes

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: pxc-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind:       Deployment
    name:     cluster1-repl1
    namespace:  pgo
  updatePolicy:
    updateMode: "Auto"
```

# 8 - Costs

- 3 database nodes per database environment (microservice)
- 6 microservices requiring the same PostgreSQL
- 16 cores, 64GB of RAM and 500GB SSD storage needed per node

- us-central1 (Iowa)
- no committed usage
- no backups
- no ingress/egress
- no resources for connection pooling/backups included

| Node | Item | Unit/year | QTY | Total/year |
|---|---|---|---|---|
| Primary nodes | db-custom-16-65536 (HA)<br>500GB Regional SSD | $21,466.20 | 6 | $128,797.20 |
| Replica nodes | db-custom-16-65536<br>500GB Regional SSD | $10,733.04 | 12 | $128,796.48 |
| | | | **Total** | **$257,593.68** |

| | |
|---|---|
| Cores total: | 288 |
| Cores per node: (3 nodes) | 96 |
| RAM total (GB): | 1152 |
| RAM per node: (3 nodes) | 384 |
| Storage total (GB): | 9000 |
| Storage per node (3 nodes) | 3000 |

| Node | Item | Unit/year | QTY | Total/year |
|---|---|---|---|---|
| GKE Worker Node | n1-custom-96-393216 | $29,997.60 | 3 | $89,992.80 |
| Storage | 9000GB Regional SSD | $36,720.00 | 1 | $36,720.00 |
| | | | **Total:** | **$126,712.80** |

| CloudSQL | GKE |
| --- | --- |
| **$257,593.68** | **$126,712.80** |

1. PostgreSQL on K8s (with Operator) can be an interesting alternative
2. K8s knowledge is required, don’t start with databases
3. Evaluate different Operators - the ecosystem is rich!
4. This tech evolves very quickly!

Q&A

# Thank you!

## Any questions?

- LinkedIn: https://www.linkedin.com/in/mmnosek/
- Email: michal.nosek@percona.com

https://percona.com