Automated Disaster Recovery for Multi-Region GenAI applications using vector databases on Kubernetes
Agenda

1. Intro - “AI is magic”
2. Problem space
   a. pgvector on Kubernetes 101
   b. Multi-region deployments
   c. Disaster Recovery (DR) problems
3. Automating DR
4. Demo
5. Q&A
AI is Magic
Creativity and imagination

Pretty kitten working behind the laptop

Unicorn plays hockey

Dad became a monster, cause we broke his Lego
AI is magic, but for Muggles

Various technologies power AI:

- Large Language Models - heart of modern Generative AI
  - Powered by Transformers (Attention is All You Need)
- GPUs for training
- Vector databases for semantic search, visual or multimodal search
  - Through vector embeddings
Everything is a vector

This is how DALL-E sees vector embedding

Example of RAG - Retrieval Augmented Generation

company data

SEMANTIC SEARCH

pgvector

response

completion with LLM

contextually enhanced prompt

question
Vector databases

Various options

- Pinecone
- Milvus
- Chroma
- FAISS
- pgvector (extension for PostgreSQL)
Problem space
pgvector on Kubernetes

YAML manifest

```yaml
apiVersion: pgv2.percona.com/v2
kind: PerconaPGCluster
metadata:
  name: my-pg
spec:
  instances:
    - name: pgvector1
      replicas: 3
  extensions:
    custom:
      - name: pgvector
        version: 0.5.1
```

Kubernetes cluster

Percona Operator

Custom Resource

Pod

postgresql + pgvector
Why multi-region

- Not only multi-region:
  - multi-cloud
  - hybrid-cloud
- Main reasons
  - Disaster recovery
  - Migration
  - No-vendor lock-in / cloud-ready
Disaster recovery

Recovery Time Objective (RTO)

- How fast do you recover from failure?

Recovery Point Objective (RPO)

- What is your last transaction?
DR through Backups - high RTO

k8s cluster A

Percona Operator

Custom Resource

pgvector

postgresql

pgbackrest

k8s cluster B

Percona Operator

Custom Resource

Object storage

Azure, GCS, S3

upload backups

restore when needed
DR through Replication - low RTO
Looks like problem solved?

1. Recover fast from failure
2. Recovery Point Objective is solved on DB level
   a. For streaming replication it can be down to almost 0
What about the app and failover?

1. Failover is not automated
   ○ Operators can detect failure, but 3rd agent needed (quorum)

2. Point application to correct database
   ○ Which endpoint should the app use when failover happens?
   ○ Who is reconfiguring it and how?

3. Failback
   ○ What should happen when main site goes back up?
Automating DR
groups:
  - name: example
    rules:
      - alert: HighRequestLatency
        expr: percona_pg_cluster_failure_total{cluster="cluster-1"} > 0
        for: 10m
        labels:
          app: example-app

https://prometheus.io/docs/alerting/latest/configuration/#webhook_config
apiVersion: recovery.eloql.co/v1alpha1
kind: RecoveryPlan
metadata:
  name: psql-primary-failover-plan
spec:
  alertLabels:
    app: example-app
  steps:
  - type: patch
    patch:
      apiVersion: "pgv2.percona.com/v2"
      resource: "perconapgclusters"
      namespace: "psql-operator"
      name: "cluster1"
      override:
        fieldPath: "spec.standby.enabled"
        value:
          raw: true
          patchType: "application/merge-patch+json"

  - type: patch
    patch:
      apiVersion: "pgv2.percona.com/v2"
      resource: "perconapgclusters"
      namespace: "psql-operator"
      name: "cluster2"
      override:
        fieldPath: "spec.standby.enabled"
        value:
          raw: false
          patchType: "application/merge-patch+json"

  - type: readField

Puts PerconaPGCluster primary into standby
apiVersion: recovery.elotl.co/v1alpha1
kind: RecoveryPlan
metadata:
  name: psql-primary-failover-plan
spec:
  alertLabels:
    app: example-app
steps:

- type: patch
  patch:
    apiVersion: "pgv2.percona.com/v2"
    resource: "percona_pgclusters"
    namespace: "psql-operator"
    name: "cluster1"
    override:
      fieldPath: "spec.standby.enabled"
      value:
        raw: true
      patchType: "application/merge-patch+json"

- type: patch
  patch:
    apiVersion: "pgv2.percona.com/v2"
    resource: "percona_pgclusters"
    namespace: "psql-operator"
    name: "cluster2"
    override:
      fieldPath: "spec.standby.enabled"
      value:
        raw: false
      patchType: "application/merge-patch+json"

Turns PerconaPGCluster standby into primary
- type: patch
  patch:
    apiVersion: "pgv2.percona.com/v2"
    resource: "perconapgcclusters"
    namespace: "psql-operator"
    name: "cluster1"
    override:
      fieldPath: "spec.standby.enabled"
      value:
        raw: true
      patchType: "application/merge-patch+json"

- type: patch
  patch:
    apiVersion: "pgv2.percona.com/v2"
    resource: "perconapgcclusters"
    namespace: "psql-operator"
    name: "cluster2"
    override:
      fieldPath: "spec.standby.enabled"
      value:
        raw: false
      patchType: "application/merge-patch+json"

- type: readField
  readField:
    apiVersion: "pgv2.percona.com/v2"
    resource: "perconapgcclusters"
    namespace: "psql-operator"
    name: "cluster2"
    fieldPath: "status.host"
    outputKey: "Cluster2IP"

- patch:
  apiVersion: "v1"
  resource: "configmaps"
  namespace: "default"
  name: "planconnect-conf"
Updates HAProxy configuration with an address of a new primary
HAPROXY_HOST = 'a8f74e18d50d145f2baf78876061c21d-1459685038.us-east-2.elb.amazonaws.com:5432/zoo?sslrootcert=/tmp/tls/ca.crt'

print(HAPROXY_HOST)

conn = psycopg2.connect(
    f'{CONN_CREDS}@{HAPROXY_HOST}"
)
cur = conn.cursor()

while True:
    cur.execute(''
        select * from test.percona;
    '"
    print(cur.fetchall(2))
    time.sleep(3)
    conn.close()

[[5, 'This framework generates embeddings for each input sentence', 'dummy1'], array([[0.0137173, 0.04285153, 0.01502861, 0.02504697, 0.05138371, 0.04365526], 0.01502861, 0.01502861, 0.02504697, 0.05138371, 0.04365526], dtype=float32)], [6, 'Sentences are passed as a list of string', 'dummy2'], array([0.0545249, 0.05508243, 0.013796, 0.01458188, 0.00743343, 0.03071218], dtype=float32)])
[[7, 'The quick brown fox jumps over the lazy dog', 'dummy3'], array([0.04393358, 0.0583442, 0.04817839, 0.01647367, 0.03835254, 0.05880814], dtype=float32)]]
Nova’s secret sauce: Schedule Policies
apiVersion: pgv2.percona.com/v2
kind: PerconaPGCluster
metadata:
  name: cluster1
namespace: psql-operator
spec:
  ...

apiVersion: policy.elotl.co/v1alpha1
kind: SchedulePolicy
metadata:
  name: psql-cluster-1
spec:
  namespaceSelector:
    matchLabels:
      kubernetes.io/metadata.name: psql-operator
  clusterSelector:
    matchLabels:
      kubernetes.io/metadata.name: us-east-1-wlc
resourceSelectors:
  labelSelectors:
    - matchLabels:
      psql-cluster: cluster-1
Thank you!
Learn more:

Percona Operators

Elotl Nova