PERCONA Live 26
BAY AREA

MongoDB on Kubernetes 101: A Hands-On Guide

Michal Nosek
Principal Field Technologist @ Percona

PERCONA

Housekeeping

- 3 long hours with me!
- Hands on! Console and SSH required!
- 20 min break in the middle
- Let me know if you're stuck so you're not behind
- Order is important
- **Get your credentials and namespace right**
- I will share a guide for you to copy/paste commands if that's your preference

- Can I do it in my private K8s cluster (e.g. local minikube)? Yes, but…

PERCONA

This tutorial is not for you if:

- You run databases on K8s since 2005
- You submitted >10 bugs to Percona Operator projects
- You're responsible for 20 000 K8s pods in production

Important Information

Property	Value
Student VM Public IP	See next slide!
Student VM User Name	percona
Student VM Password	unbiased_open_source_experts
Kubernetes Namespace	ns-X
PMM Public IP	34.56.66.233
PMM user/pass	admin/admin
Slides (you can copy/paste)	**https://tinyurl.com/2ddt9n7t**

Directory Location	Description
/home/percona/percona-server-mongodb-operator	Percona K8s Operator repository files
/home/percona/percona-server-mongodb-operator/deploy	**Percona K8s Operator .yaml files**

PERCONA

Student VM IPs

namespace	IP
ns-2	136.113.71.68
ns-3	34.69.117.73
ns-4	34.122.201.192
ns-5	136.112.208.233
ns-6	34.27.200.214
ns-7	34.71.204.5
ns-8	34.69.223.45
ns-9	34.58.126.27

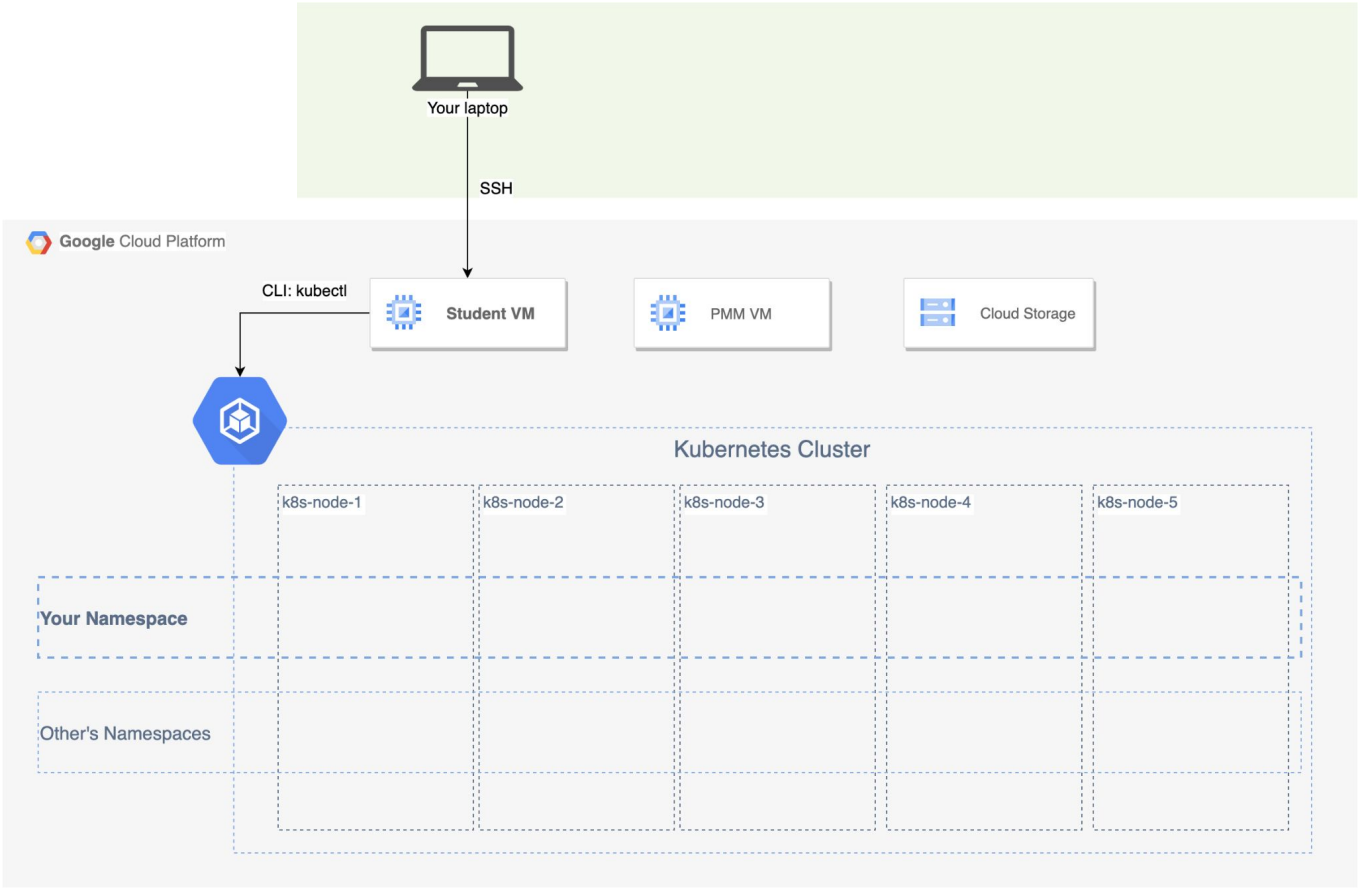
PERCONA

PERCONA®

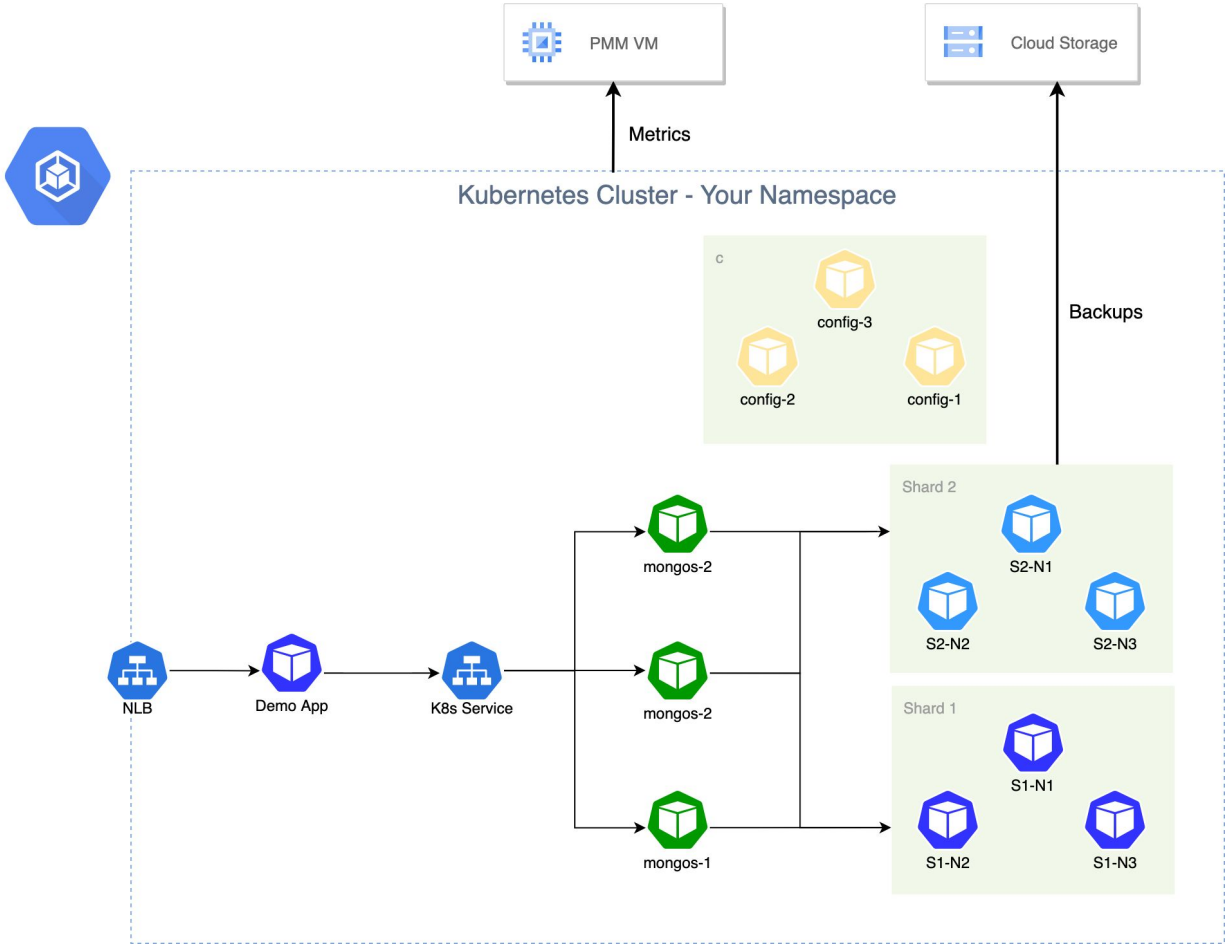
0. **Percona Server for MongoDB and K8s Operators**

1. Connectivity test and files structure
2. Deploy the Operator
3. Deploy a replica set (single shard)
4. Deploy a test app and manage users
5. Config changes
6. Scale up and out
7. Major version upgrade
8. Backup and Restore
9. Monitoring
10. High Availability and self-healing

PERCONA®

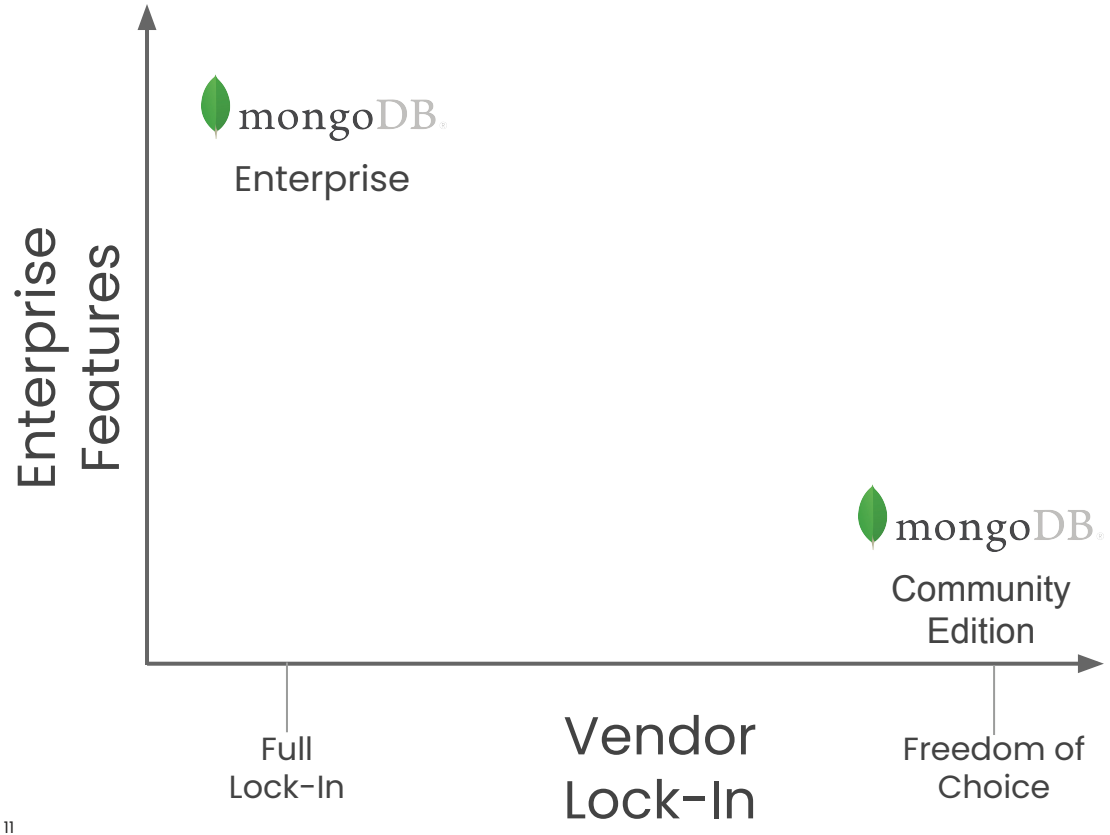


PERCONA



PERCONA

PERCONA



Critical features

Encryption | LDAP | Monitoring | Auditing | Support & SLAs | etc.

Come with strings attached!

PERCONA

Percona Server for MongoDB

Binary compatible, drop-in replacement

100% open source, free to download and use

Enterprise features, without the restrictions

Works everywhere, on-premises, cloud, hybrid

Enterprise Level QA

Test and package for everyone!

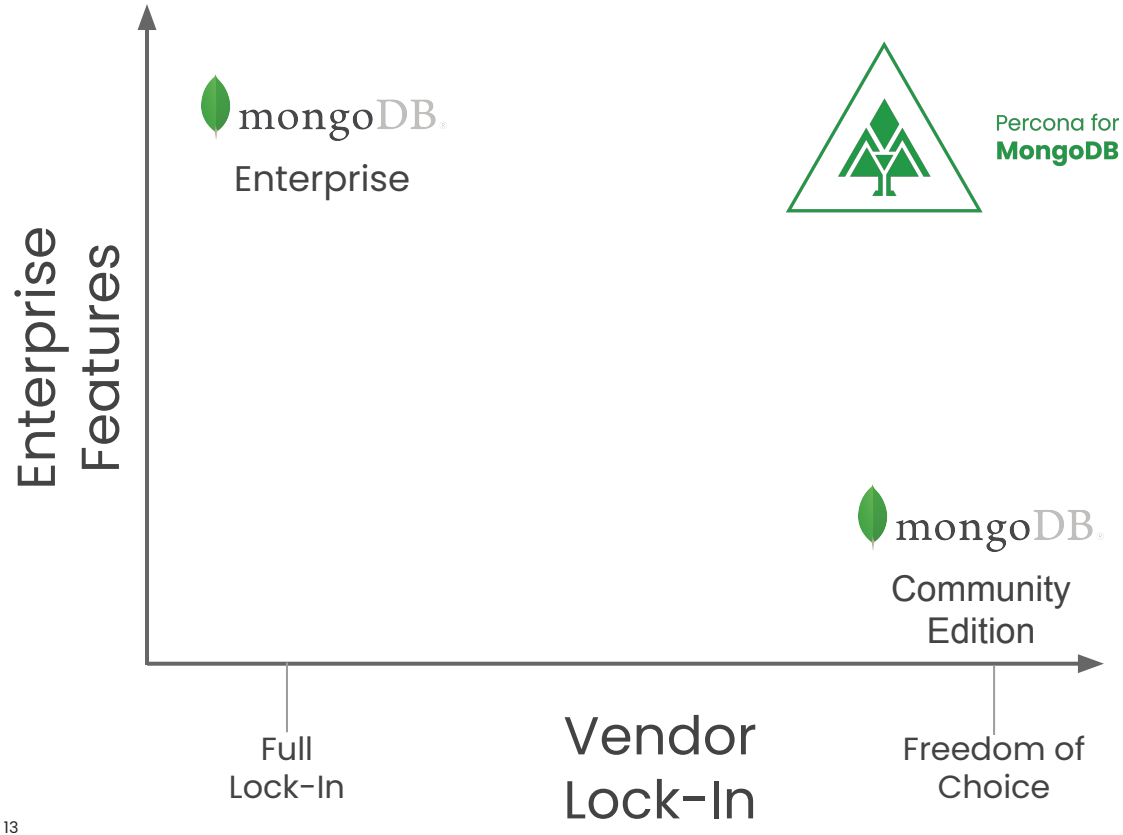
Enterprise Features

Bring in the enterprise features companies need.

Critical Bug Fixes

Fix the bugs and make it super perform.

PERCONA®



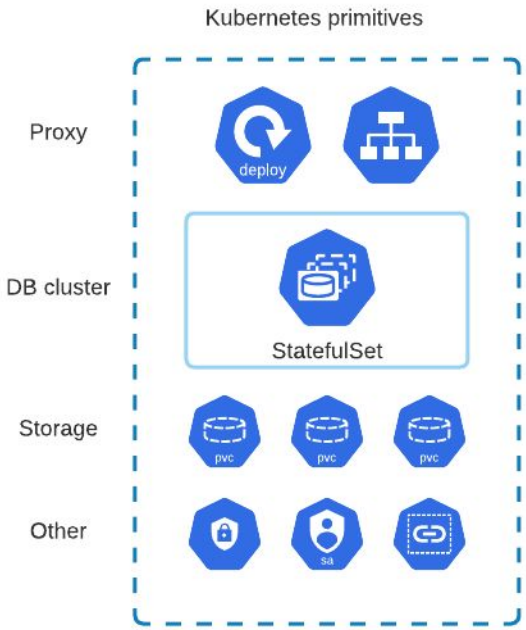
Critical features

Encryption | LDAP | Monitoring | Auditing | Support & SLAs | etc.

With no strings attached!

PERCONA

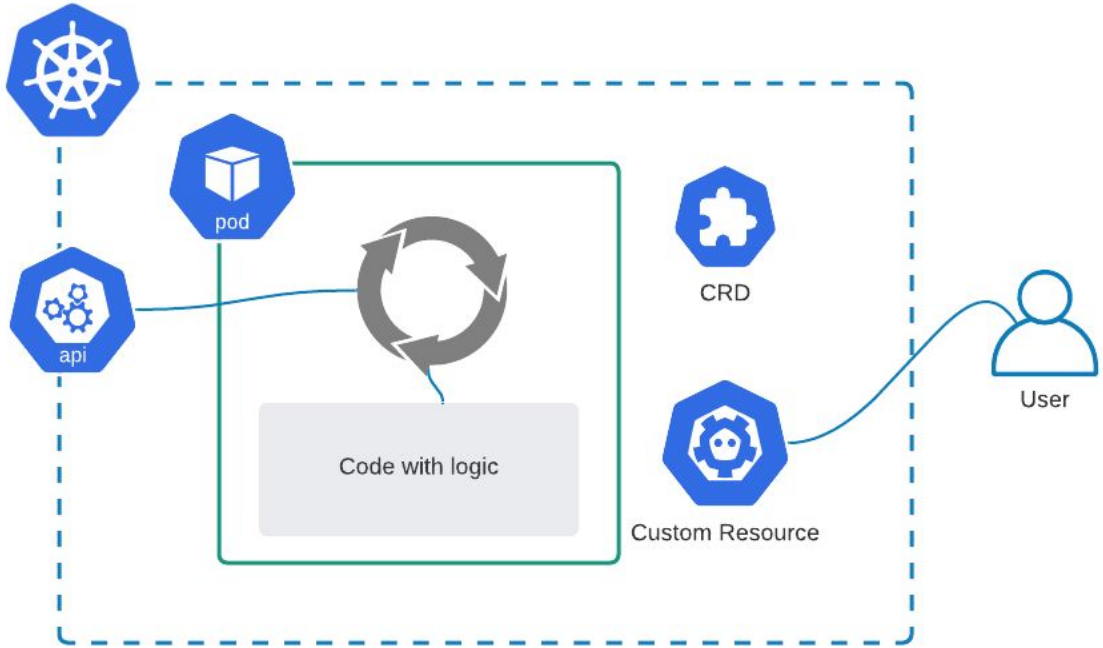
Databases on Kubernetes



1. Underlying Kubernetes
2. StatefulSet for Primary/Replicas
3. High Availability and failover
4. DR
5. Storage (PVC/Hostpath)
6. Services
   a. Internal
   b. External
7. Configuration
8. Monitoring
9. Backups
10. Upgrades
11. Encryption
12. .....

PERCONA

Operators

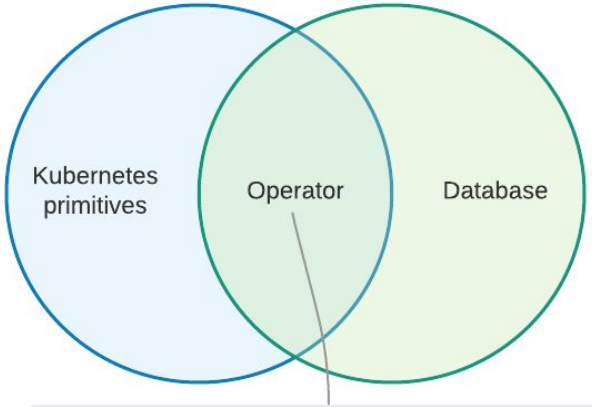


PERCONA

Operators

- Operator controls database and k8s primitives

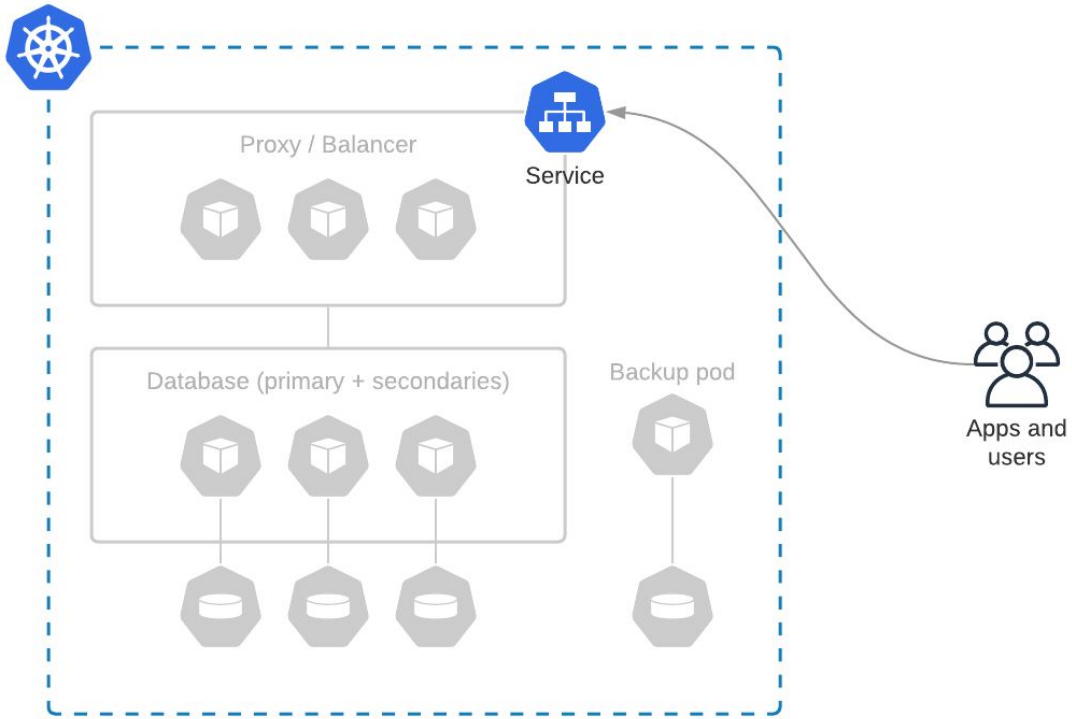
- Day-1 simplified to one step
- Day-2 operations automated




```
spec:
  image: percona/percona-server-mongodb:4.4.6-8
  replsets:
   - name: rs0
      size: 3
    sharding:
       mongos:
          size: 3
        configsvrReplSet:
           size: 3
      backups:
         ...
```


PERCONA

Kubernetes Operators



PERCONA

PERCONA

Instructions explained (example only)

 # cr.yaml - yaml file name that we expect you to edit/create

```
backup:
  enabled: true
  restartOnFailure: true
  name: YOUR_BACKUP_NAME_HERE
```

```
# What the command does
$ kubectl apply -f cr.yaml - command to execute
```

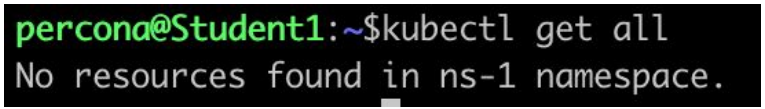
```
percona@Student0:~/percona-server-mongodb-operator$kubectl get pods
NAME                                               READY   STATUS              RESTARTS   AGE
my-cluster-name-rs0-0                              2/2     Running             0          118s
my-cluster-name-rs0-1                              2/2     Running             0          81s
my-cluster-name-rs0-2                              0/2     ContainerCreating   0          39s
percona-server-mongodb-operator-568f85969c-pvhr8   1/1     Running             0          7m56s
```

1.1 Connect

```
# 1. Connect to your student VM public IP
$ ssh percona@<student_instance_server_pub_ip>

# 2. Check if kubectl works
$ kubectl get all

# 3. Check if k9s works (I recommend new tab)
$ k9s
```



PERCONA

1.2 Inspect config files

```
# Inspect file structure - this is your working directory
$ cd percona-server-mongodb-operator/deploy/

# Check the files
$ more secrets.yaml
$ more cr.yaml
$ more backup-s3.yaml
$ more backup/backup.yaml
$ more backup/restore.yaml
```

Percona Operator Objects:

cr.yaml:**kind**: PerconaServerMongoDB
backup.yaml:**kind**: PerconaServerMongoDBBackup
backup/restore.yaml:**kind**: PerconaServerMongoDBRestore

PERCONA

1.3 Useful kubectl commands

Command	Description
`kubectl apply -f file.yaml`	Deploys an object from yaml file
`kubectl get all`	Lists all objects
`kubectl get pods`	Lists all pods
`kubectl get services`	Lists all services
`kubectl describe pod|service|other_resource`	Shows details of the resource
`kubectl logs pod_name`	Shows logs related to a pod
`kubectl delete pod pod_name`	Deletes a pod

PERCONA

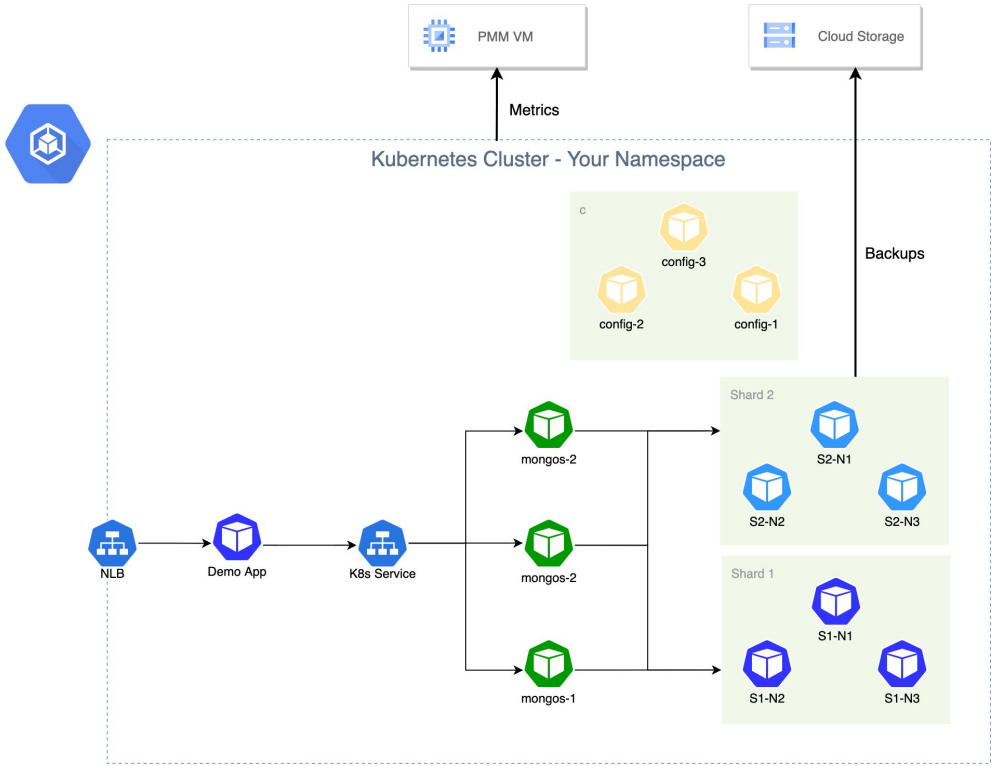
PERCONA®

2.0 Deploy the Operator

1. CRDs are cluster-wide, I already deployed them for you

2. RBAC is namespace-wide, so we deploy it

3. Operator will be independent in each namespace. **Why?**



PERCONA

2.1 Deploy Operator

```
# Deploy the role-based access control configuration
$ kubectl apply -f rbac.yaml

# Deploy Percona Operator
$ kubectl apply -f operator.yaml

# Check if your operator is successfully deployed
$ kubectl get pods
```

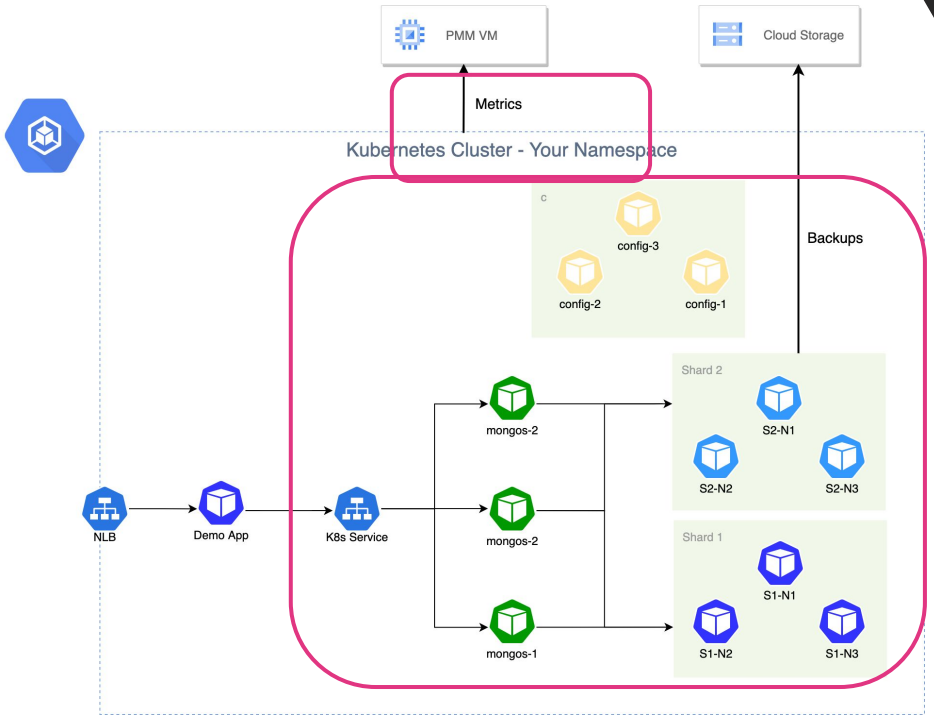
```
percona@Student0:~/percona-server-mongodb-operator/deploy$kubectl get pods
NAME                                               READY   STATUS    RESTARTS   AGE
percona-server-mongodb-operator-5dd88ff7f7-v5ckq   1/1     Running   0          9s
```


PERCONA

PERCONA

3.0 Deploy a replica set

1. Deploy system user secret
2. Edit cluster parameters
   a. Resources assigned
   b. MongoDB version
   c. Storage size
3. Deploy the cluster
4. Connect to the cluster



PERCONA

3.1 Deploy system user secret

```
# secrets.yaml
apiVersion: v1
kind: Secret
metadata:
  name: YOUR_SECRET_NAME
type: Opaque
stringData:
  MONGODB_BACKUP_USER: backup
  MONGODB_BACKUP_PASSWORD: backup123456
  MONGODB_CLUSTER_ADMIN_USER: clusterAdmin
  MONGODB_CLUSTER_ADMIN_PASSWORD: clusterAdmin123456
  MONGODB_CLUSTER_MONITOR_USER: clusterMonitor
  MONGODB_CLUSTER_MONITOR_PASSWORD: clusterMonitor123456
  MONGODB_USER_ADMIN_USER: userAdmin
  MONGODB_USER_ADMIN_PASSWORD: userAdmin123456
  PMM_SERVER_TOKEN: PMM_AUTH_TOKEN
```

```
# Deploy secrets
$ kubectl apply -f secrets.yaml
```


PERCONA

3.2 Edit cluster parameters

cr.yaml

`metadata.name = **YOUR_CLUSTER_NAME**` (your invention)
`spec.secrets.users = **YOUR_SECRET_NAME**` (from the previous step 3.1)
`spec.replsets.[0].name = **shard1**`
`spec.replsets.[0].volumeSpec.persistentVolumeClaim.resources.requests.storage = **4Gi**`

PERCONA®

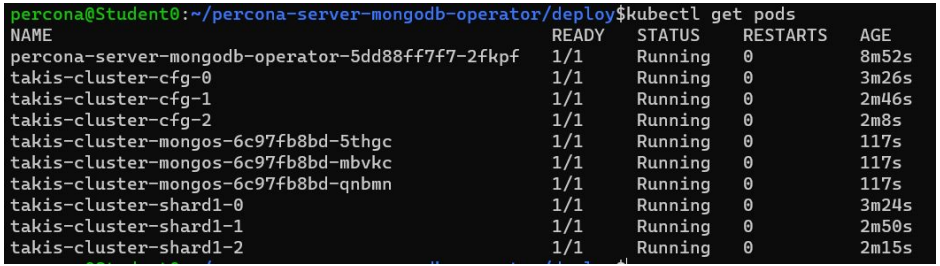
3.3 Deploy the cluster

```
# Deploy the cluster
$ kubectl apply -f cr.yaml

# wait for the  *magic* to happen!
$ k9s

OR

$ kubectl get pods
$ kubectl get services
$ kubectl get PerconaServerMongoDB
$ kubectl get pvc
```



PERCONA

3.4 Connect to the cluster

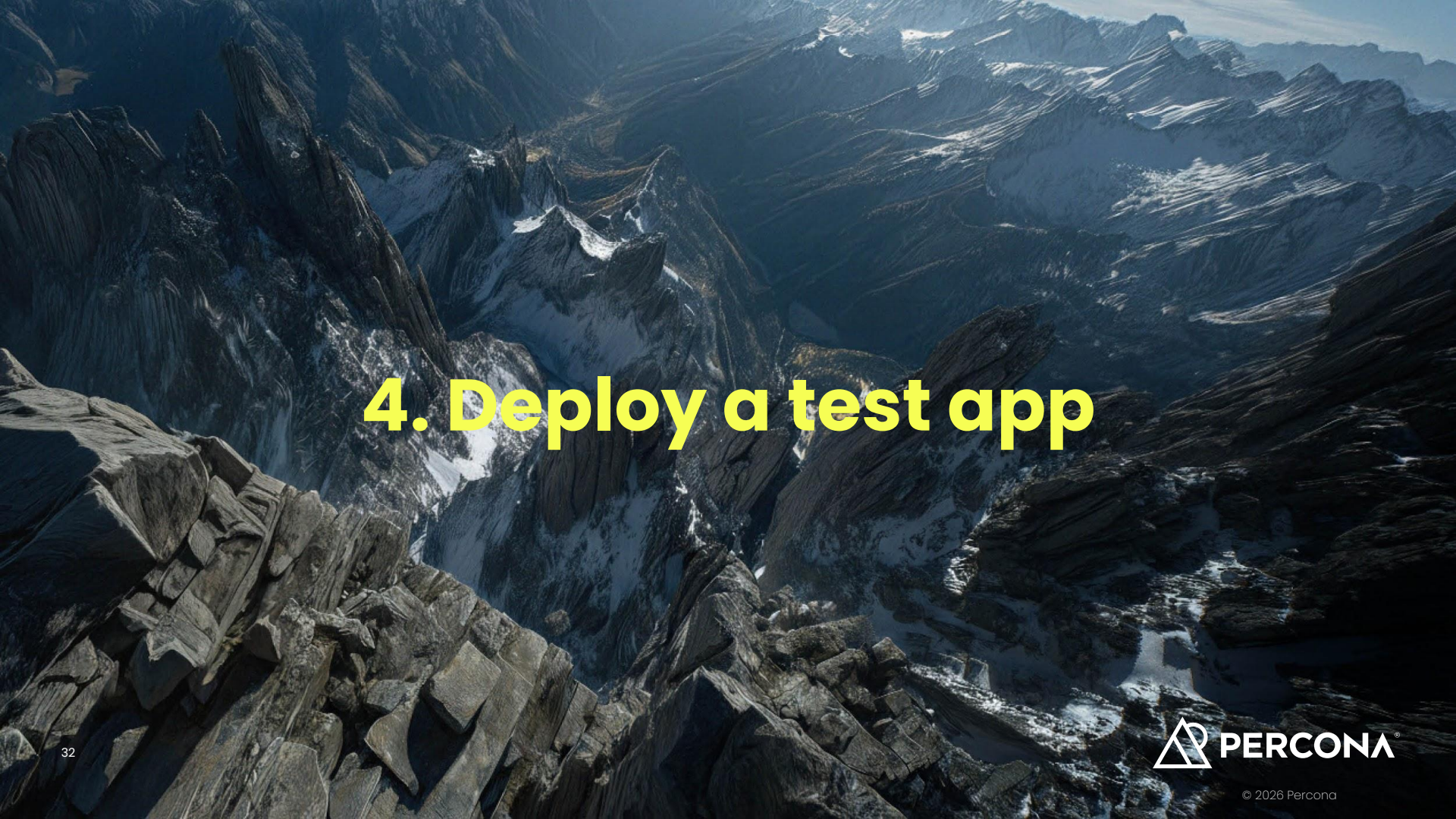
```
# Launch docker container of PSMDB 7.0 client to connect to our PSMDB
# You may use a separate session for this
# This should take few seconds

$ kubectl run -i --rm --tty percona-client \
  --image=percona/percona-server-mongodb:7.0 --restart=Never -- bash -il


# use mongo client to connect to the database
mongodb@percona-client /]$
mongosh "mongodb://userAdmin:userAdmin123456@CLUSTER_NAME-mongos/admin?ssl=false"

# check DBs
$ show dbs;
```

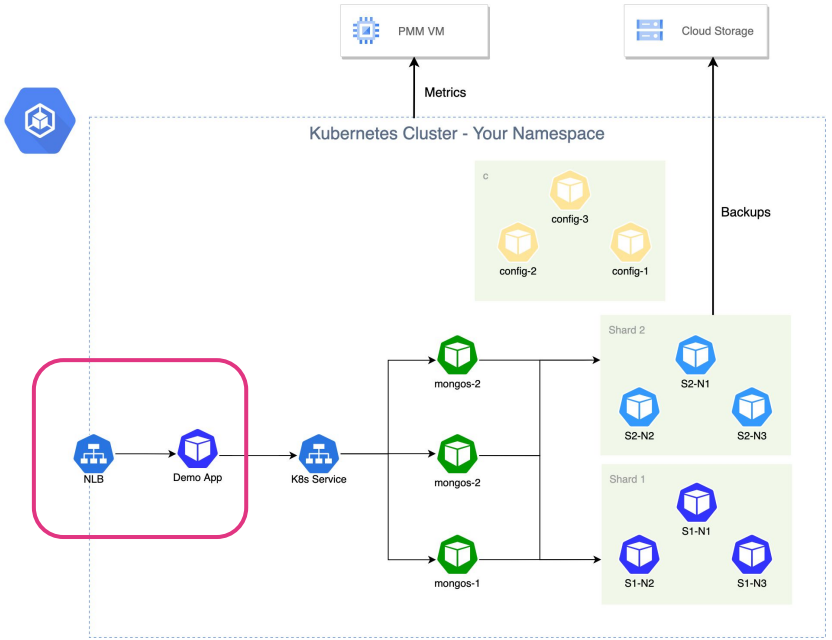

PERCONA



PERCONA®

4.0 Deploy a test app

1. Create app db and user
2. Create a K8s deployment of the container image that we prepared before (todo app)
3. Expose the app to the public internet through NLB
4. Play around with the app (verify it works)



PERCONA

4.1 Create DB and users

```
# cr.yaml
users:
  - name: demoApp
    db: test
    roles:
      - name: readWrite
        db: test
```

```
# Apply changes and check
$ kubectl apply -f cr.yaml
$ kubectl get secrets
```


PERCONA

4.2 Deploy the containerized app

```
# demoapp-deployment.yaml
#Map MongoDB password and service here
             env:
           - name: MONGOS_SERVICE
             value: "my-cluster-name-mongos:27017"  # MONGOS ENDPOINT
           - name: MONGODB_PASSWORD
             valueFrom:
               secretKeyRef:
                 name: my-cluster-name-secret # YOUR SECRET NAME
                 key: demoApp
```

```
# Apply changes and check when it's ready
$ kubectl apply -f demoapp-deployment.yaml
$ kubectl get pods
```


PERCONA

4.3 Expose the demo app via a NLB

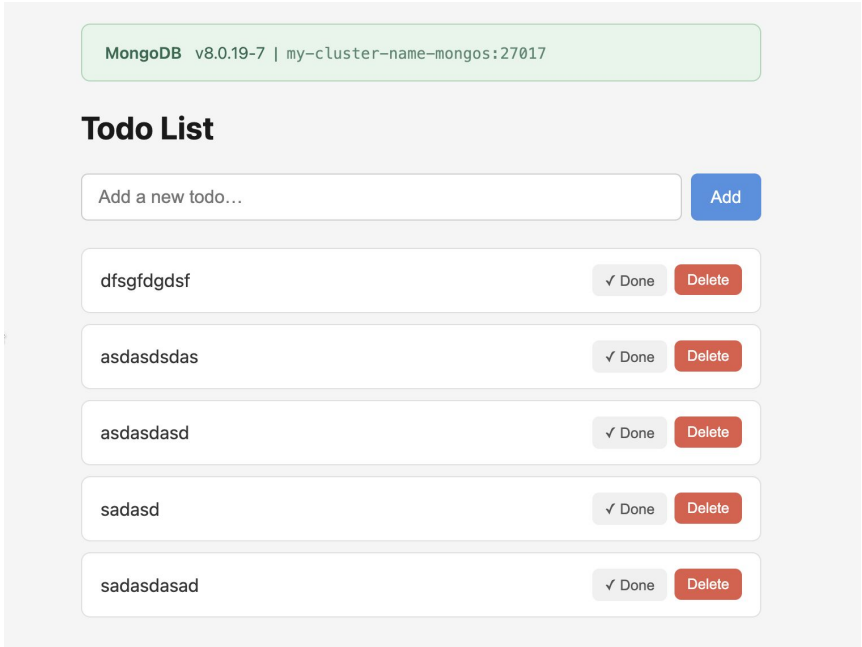
```
# demoapp-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: todo-demoapp
spec:
  type: LoadBalancer
  selector:
    app: todo-demoapp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
```

```
# Apply changes and check when the public IP is ready
$ kubectl apply -f demoapp/demoapp-service.yaml
$ kubectl get services
```

```
percona@Student0:~/percona-server-mongodb-operator/percona-server-mongodb-operator/deploy/
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
demoapp-service     LoadBalancer   10.116.3.151    <pending>     80:31326/TCP   12s
```

4.4 Play arround with the demo app

1. Go to http://APP_EXTERNAL_IP
2. Add some data and explore the app



PERCONA

4.5 Fix the issue

```
# cr.yaml
users:
  - name: demoApp
    db: test
    roles:
      - name: readWrite
        db: test
      - name: clusterMonitor
        db: admin
```

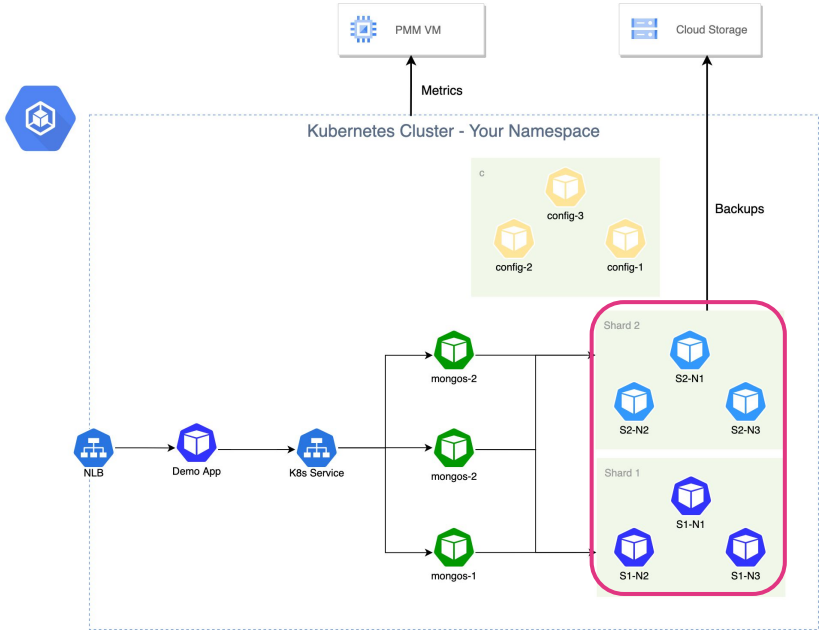
```
# Apply changes and check
$ kubectl apply -f cr.yaml
$ kubectl get secrets
```


PERCONA

PERCONA

5.0 Modify MongoDB Parameters

1. ~~Direct edit of mongodb.conf~~
2. Options:
   a. Use K8s ConfigMap
      e.g. my-cluster-name-rs0-mongod
   b. Use K8s Secret
   c. Specific section of cr.yaml (replicaset, mongos, configsvrReplSet)



PERCONA

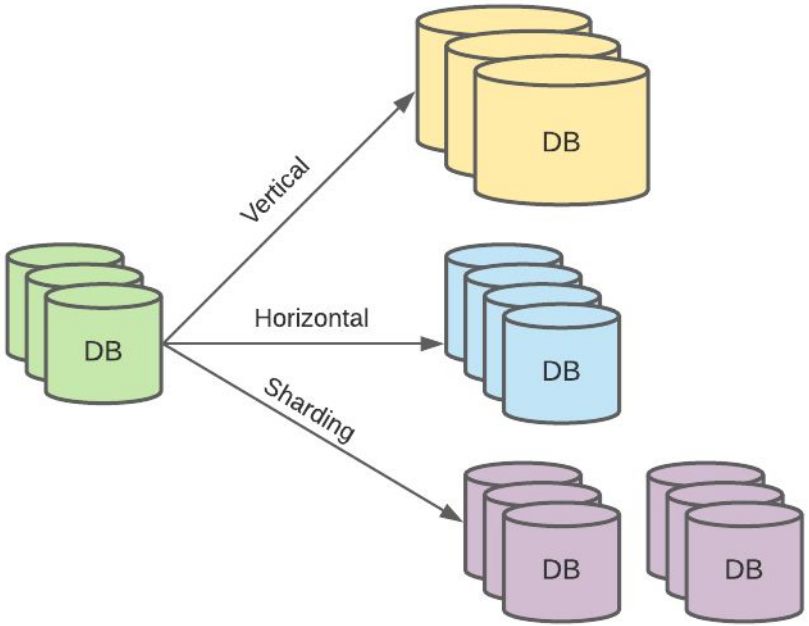
PERCONA

6.0 Scale up and out

- Vertical: Add more CPU, RAM
- Horizontal: Add more replicas
- Sharding: Divide the workload into multiple replica sets

- Storage scalability depends on CSI

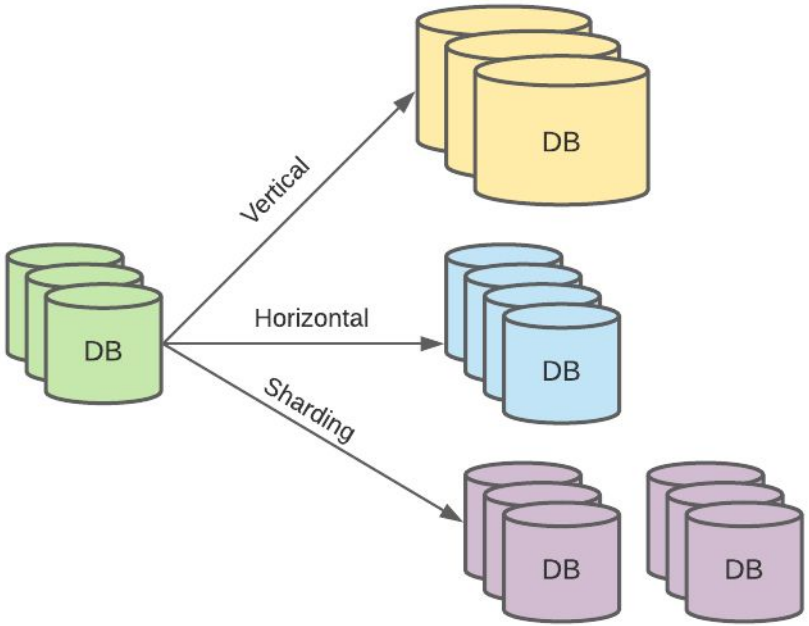
- Vertical autoscaler supported



PERCONA

6.0 Scale up and out

1. Add a shard
2. Add nodes to each shard
3. Debug issues
4. Add more resources



PERCONA

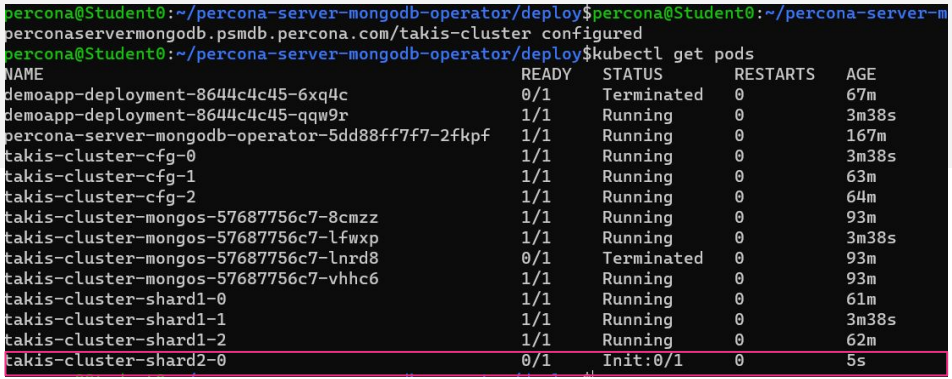
6.1 Add shard

```
# Copy entire shard1 definition and rename it
```

```
# cr.yaml
replsets:
- name: shard1
  (...)

- name: shard2
  (...)
```

```
# Apply changes and check
$ kubectl apply -f cr.yaml
$ kubectl get pods
```



PERCONA

6.2 Scale each shard to 5 nodes

```
# cr.yaml

 replsets:
  - name: shard1
    size: 5
    (...)
  - name: shard2
    size: 5
    (...)
```

```
# Apply changes and check
$ kubectl apply -f cr.yaml
$ kubectl get pods
```


PERCONA

6.3 Scale each shard to 5 nodes - debug

```
# Check what happened
$ kubectl describe pod PENDING_POD_NAME

# This is pod anti-affinity. Let us fix it!

# Check results
$ kubectl get pods
```


PERCONA

6.4 Add more resources

cr.yaml

```
replsets:
 - name: shard1
   (...)
   resources:
     limits:
       cpu: "350m"
       memory: "0.6G"
     requests:
       cpu: "350m"
       memory: "0.6G"
```

```
# Apply changes and check changed resources
$ kubectl apply -f cr.yaml
$ kubectl describe pod ONE_OF_THE_PODS
```


PERCONA

PERCONA

7.0 Major version upgrade

- Operator picks up a new image version and starts upgrade procedure
- SmartUpdate available (for non-prod)
- Be aware of Operator version and CRD version (cluster-wide!)
- We simply upgrade from 7 to 8, minor upgrades look similarly

PERCONA®

7.1 Apply new major version

```
# cr.yaml
  spec.image = percona/percona-server-mongodb:8.0.19-7
```

```
# Check Mongo image version before upgrade (or in the app)
$ kubectl describe psmdb my-cluster-name | grep Image

# Apply changes
$ kubectl apply -f cr.yaml

# Observe cluster restart
$ kubectl get pods

# Check Mongo image version
$ kubectl describe psmdb my-cluster-name | grep Image
```

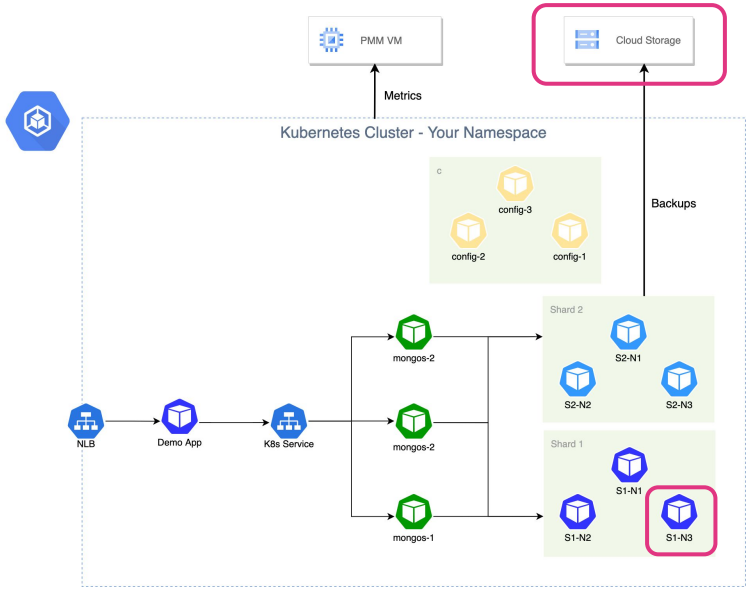

PERCONA

PERCONA

8.0 Backup and restore

1. Create secret to store Cloud Storage API keys
2. Enable backups and configure storage details
   - Google Cloud Storage through S3 API
3. Execute an on-demand full backup
4. Configure scheduled backups

5. Make changes to the data (through the app)
6. Deploy restore job



8.1 S3 secret keys

```
# backup-gcs.yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-cluster-name-backup-gcs
type: Opaque
data:
  GCS_CLIENT_EMAIL: BASE64_ENCODED_CLIENT_EMAIL
  GCS_PRIVATE_KEY: BASE64_ENCODED_SECRET
```

```
# Apply changes
$ kubectl apply -f backup-gcs.yaml
```


PERCONA

8.2 Enable backups

```
# cr.yaml
backup:
    enabled: true
    image: percona/percona-backup-mongodb:2.12.0
    storages:
      gcs:
        type: gcs
        gcs:
          bucket: k8s_workshop_backups
          prefix: "YOUR-NAMESPACE-HERE"
          credentialsSecret: CREDENTIALS_SECRET_HERE
          chunkSize: 10485760
          retryer:
            backoffInitial: 1
            backoffMax: 30
            backoffMultiplier: 2
    pitr:
      enabled: true
      oplogOnly: false
      oplogSpanMin: 10
```

```
# Apply changes
$ kubectl apply -f cr.yaml
```


PERCONA

8.3 On-demand full backup

```
# backup/backup.yaml
apiVersion: psmdb.percona.com/v1
kind: PerconaServerMongoDBBackup
metadata:
  finalizers:
  - percona.com/delete-backup
  name: NAME_OF_YOUR_BACKUP
spec:
  clusterName: NAME_OF_YOUR_CLLUSTER
  storageName: STORAGE_NAME
  type: BACKUP_TYPE #(physical/logical)
```

```
# Check ready status
$ kubectl get psmdb
# Run backup
$ kubectl apply -f backup/backup.yaml
# Check its status
$ kubectl get psmdb-backup
```

```
percona@Student0:~/percona-server-mongodb-operator/deploy$kubectl get psmdb-backup
NAME         CLUSTER           STORAGE   DESTINATION                                          TYPE       SIZE      STATUS
testoneoff   my-cluster-name   gcs       gs://k8s_workshop_backups/ns-0/2026-05-29T00:35:04Z   physical   29.75MB   ready
```

8.4 Configure Schedule backups

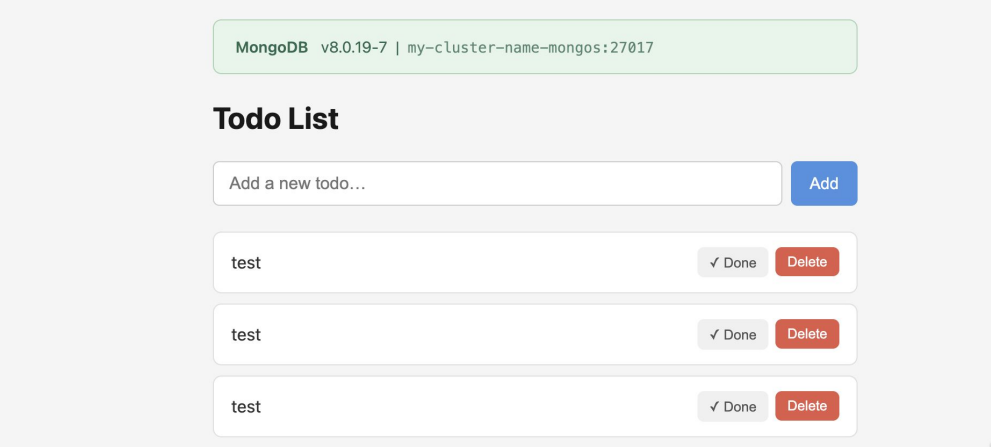
```
# cr.yaml
 tasks:
    - name: BACKUP_NAME
      enabled: true
      schedule: "0 5 * * 0"
      retention:
        count: 3
        type: count
        deleteFromStorage: true
      type: physical
      storageName: gcs
```

```
# Apply changes to schedule backup
$ kubectl apply -f cr.yaml
```


PERCONA

8.5 Add some new data!

1. Go to http://APP_EXTERNAL_IP
2. Add some more entries!



PERCONA

8.6 Restore

```
# backup/restore.yaml
apiVersion: psmdb.percona.com/v1
kind: PerconaServerMongoDBRestore
metadata:
  name: restore1
spec:
  clusterName: NAME_OF_YOUR_CLUSTER
  backupName: NAME_OF_YOUR_BACKUP
```

```
# Run restore
$ kubectl apply -f backup/restore.yaml
# Check its status
$ kubectl get psmdb-restore
```

```
percona@Student0:~/percona-server-mongodb-operator/percona-server-mongodb-operator/deploy$kubectl get psmdb-restore
NAME       CLUSTER           STATUS   AGE
restore1   my-cluster-name   ready    5m20s
```


PERCONA

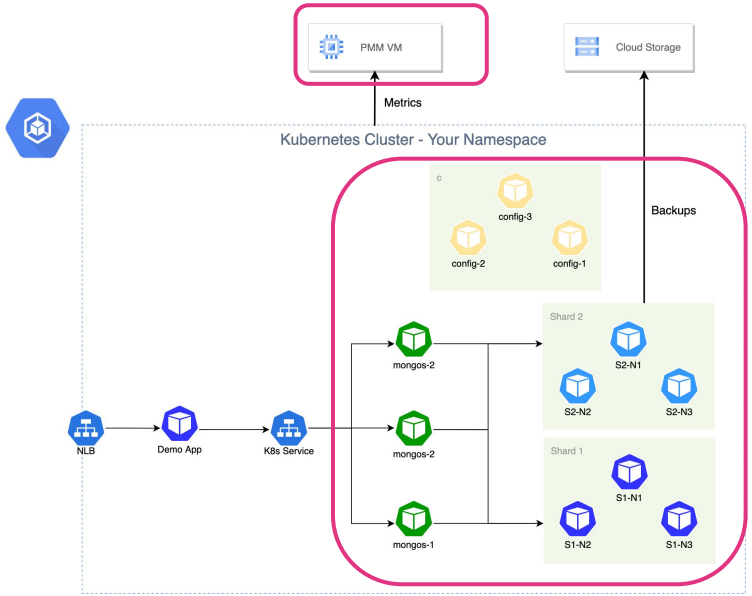
8.7 Confirm the restore

1. Go to http://APP_EXTERNAL_IP
2. Check that your entries disappeared

PERCONA

9.0 Monitoring

1. Provide configuration to connect to PMM
2. Explore PMM



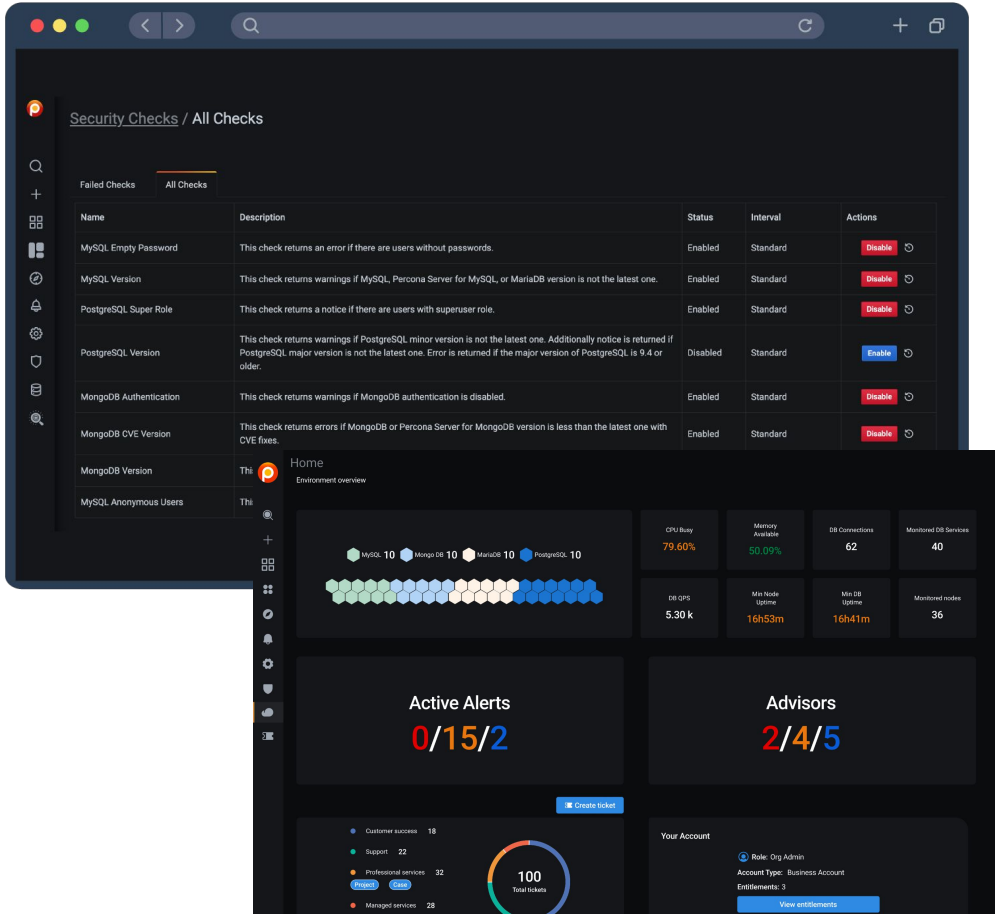
9.1 Connect to PMM

```
# cr.yaml
pmm:
    enabled: false
    image: percona/pmm-client:3.6.0
    serverHost: PMM_PUBLIC_IP
    customClusterName: NAMESPACE
    mongodParams: --environment=NAMESPACE
    mongosParams: --environment=NAMESPACE
```

```
# Apply changes
$ kubectl apply -f cr.yaml
# Observe cluster restart
$ kubectl get pods
```

9.2 Explore PMM

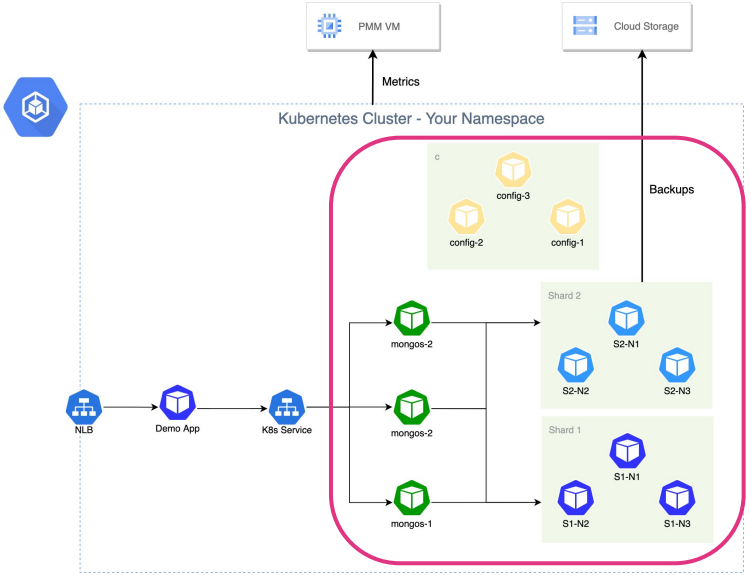
1. Go to http://PMM_PUBLIC_IP
2. Explore Dashboards
3. Explore Query Analytics



PERCONA®

10.0 High Availability and self healing

1. Open second terminal sessions to observe the situation
2. Delete random pod(s) - try to break it
3. Kill mongod process



PERCONA

10.1 HA and self-healing

Open extra ssh session:

```
# Connect
$ ssh percona@<student_instance_server_pub_ip>

# Watch pods status with k9s
$ k9s
```

Second ssh session

```
# get psmdb state
$ kubectl get psmdb
```

10.2 HA and self-healing

Second ssh session

```
# delete pod(s)
$ kubectl delete pod NAME_POD1 NAME_POD2 NAME_POD3
```

1. Watch your 1st terminal
2. Go to http://APP_EXTERNAL_IP
3. Can you still read/write data?

PERCONA

10.3 HA and self-healing

Second ssh session

```
# kill mongod
$ kubectl exec -it POD_NAME -- pkill mongod
```

1. Watch your 1st terminal - what are you getting as output?
2. Go to http://APP_EXTERNAL_IP
3. Can you still read/write data?

PERCONA

PERCONA®