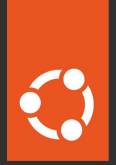


MySQL on K8s

From pitfalls to best practices

Jan-2025

Agenda

1. **Pitfalls of running databases on K8s**
2. **Best practices for running MySQL on K8s**
3. **Going further**

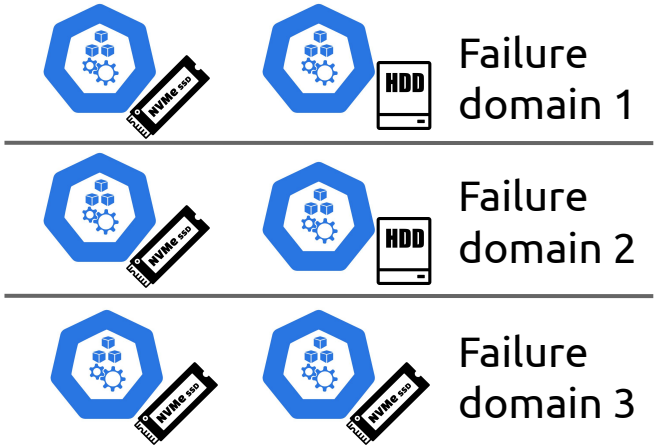
Agenda

1. **Pitfalls of running databases on K8s**
2. Best practices for running MySQL on K8s
3. Going further

Pitfalls of running databases on K8s

Placement constraints

How to dispatch the pods over the worker nodes ?

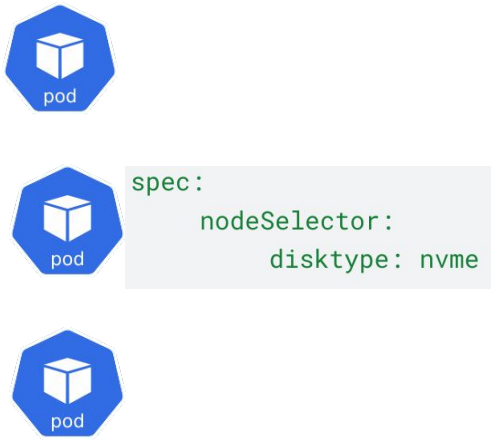


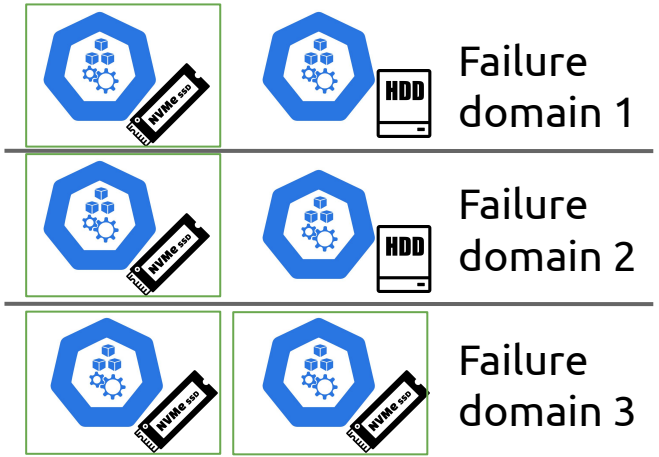
Includes MySQL pods

Worker nodes

Pitfalls of running databases on K8s

Placement constraints using **node selector**



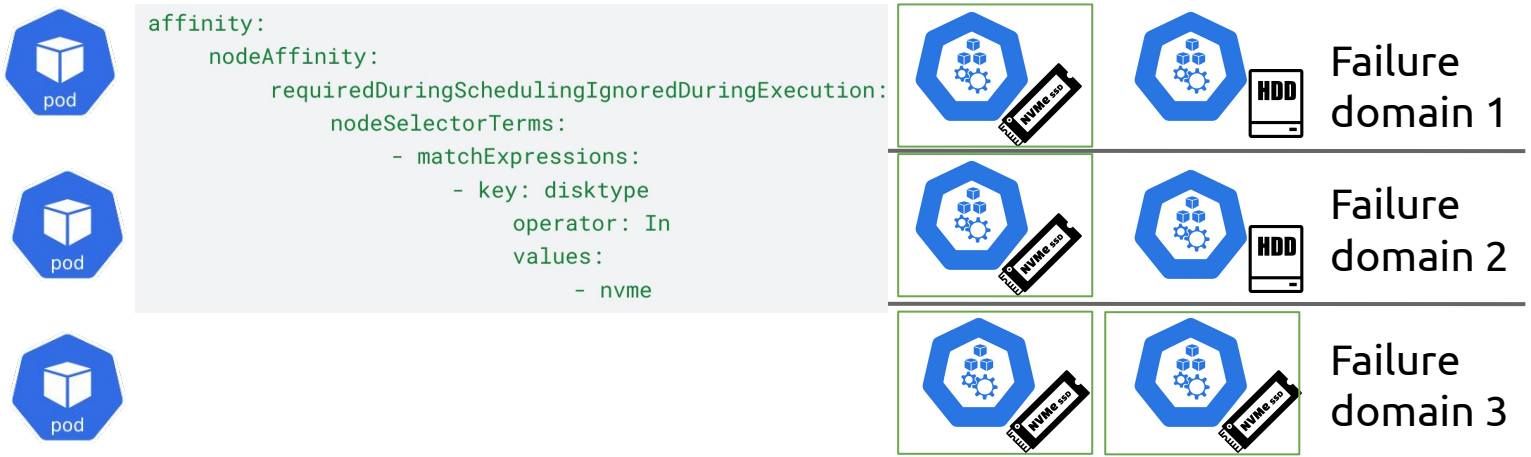


Includes MySQL pods

Worker nodes

Pitfalls of running databases on K8s

Placement constraints using **node affinity**



Includes MySQL pods

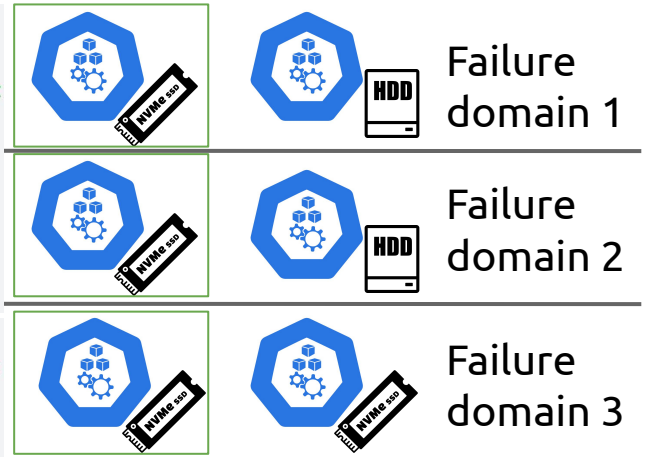
Worker nodes

Pitfalls of running databases on K8s

Placement constraints using **node affinity and pod anti-affinity**


```
affinity:
     nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
               nodeSelectorTerms:
                    - matchExpressions:
                         - key: disktype
                              operator: In
                              values:
                                   - nvme
```

```
podAntiAffinity:
  {required,preferred}DuringSchedulingIgnoredDuringExecution:
    - weight: 100
      podAffinityTerm:
        labelSelector:
          matchExpressions:
          - key: cluster
            operator: In
            values:
            - M1
        topologyKey: topology.kubernetes.io/zone
```



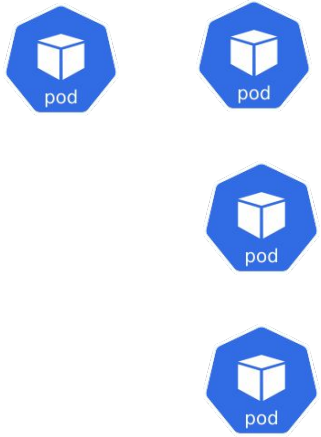
Includes MySQL pods

Worker nodes

Pitfalls of running databases on K8s

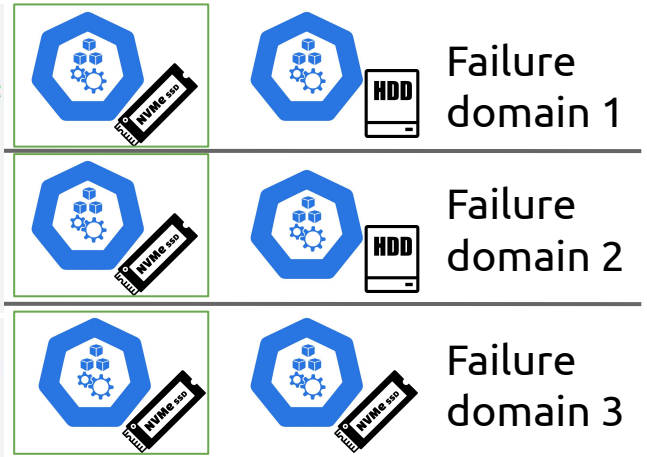
Placement constraints using **node affinity and pod anti-affinity**

?




```
affinity:
    nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
                - matchExpressions:
                    - key: disktype
                      operator: In
                      values:
                          - nvme
```

```
podAntiAffinity:
  {required,preferred}DuringSchedulingIgnoredDuringExecution:
    - weight: 100
      podAffinityTerm:
        labelSelector:
          matchExpressions:
          - key: cluster
            operator: In
            values:
            - M1
        topologyKey: topology.kubernetes.io/zone
```

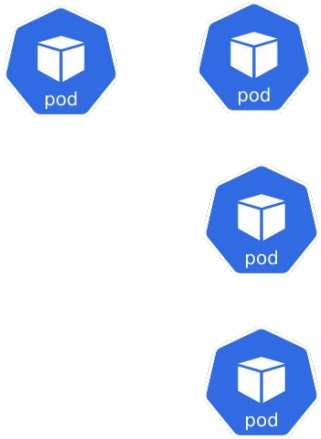


Includes MySQL® pods

Worker nodes

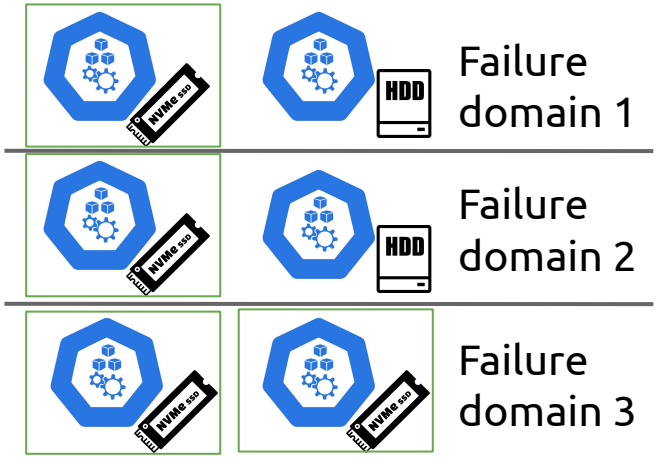
Pitfalls of running databases on K8s

Placement constraints using **TopologySpreadConstraints** and **affinity**



Includes MySQL pods

```yaml
affinity:
    nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
                - matchExpressions:
                    - key: disktype
                      operator: In
                      values:
                          - nvme
topologySpreadConstraints:
    - maxSkew:
        topologyKey: zone
        minDomains: 3
        whenUnsatisfiable: DoNotSchedule
        nodeAffinityPolicy: Honor
        labelSelector:
            matchLabels:
                cluster: m1
    -  maxSkew: 0
        topologyKey: node
        minDomains: 4
        whenUnsatisfiable: DoNotSchedule
        nodeAffinityPolicy: Honor
        labelSelector:
            matchLabels:
                cluster: m1
```



Worker nodes

Pitfalls of running databases on K8s

Placement constraints using **TopologySpreadConstraints**

Known limitations

- There's no guarantee that the constraints remain satisfied when Pods are removed. For example, scaling down a Deployment may result in imbalanced Pods distribution.

  You can use a tool such as the Descheduler to rebalance the Pods distribution.

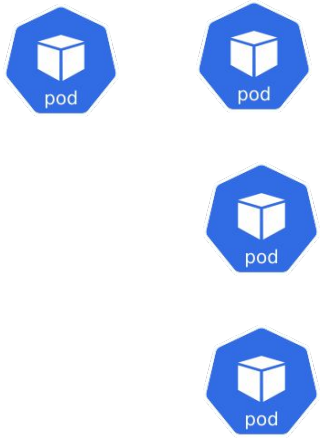
- Pods matched on tainted nodes are respected. See Issue 80921.

- The scheduler doesn't have prior knowledge of all the zones or other topology domains that a cluster has. They are determined from the existing nodes in the cluster. This could lead to a problem in autoscaled clusters, when a node pool (or node group) is scaled to zero nodes, and you're expecting the cluster to scale up, because, in this case, those topology domains won't be considered until there is at least one node in them.

  You can work around this by using a cluster autoscaling tool that is aware of Pod topology spread constraints and is also aware of the overall set of topology domains.

Pitfalls of running databases on K8s

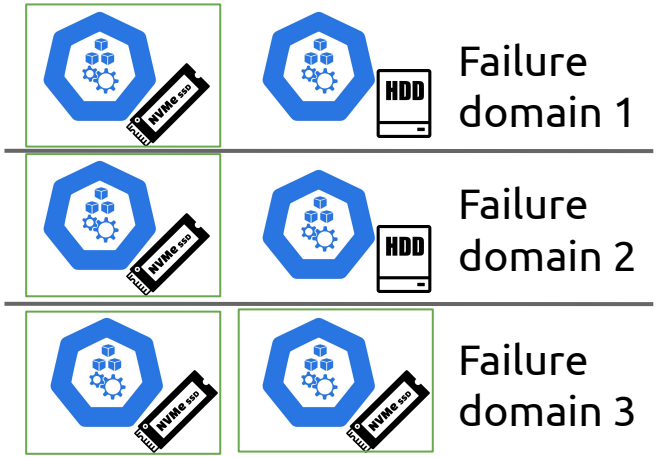
Placement constraints using **TopologySpreadConstraints** and **Taints**



Includes MySQL pods

```yaml
topologySpreadConstraints:
    - maxSkew:
        topologyKey: zone
        minDomains: 3
        whenUnsatisfiable: DoNotSchedule
        nodeAffinityPolicy: Honor
        labelSelector:
            matchLabels:
                cluster: m1
    -  maxSkew: 0
        topologyKey: node
        minDomains: 4
        whenUnsatisfiable: DoNotSchedule
        nodeAffinityPolicy: Honor
        labelSelector:
            matchLabels:
                cluster: m1
tolerations:
  - key: "disktype"
    operator: "Equal"
    value: "nvme"
    effect: "NoSchedule"


  - key: "node.kubernetes.io/unreachable"
    operator: "Exists"
    effect: "NoExecute"
    tolerationSeconds: 6000
```



Worker nodes

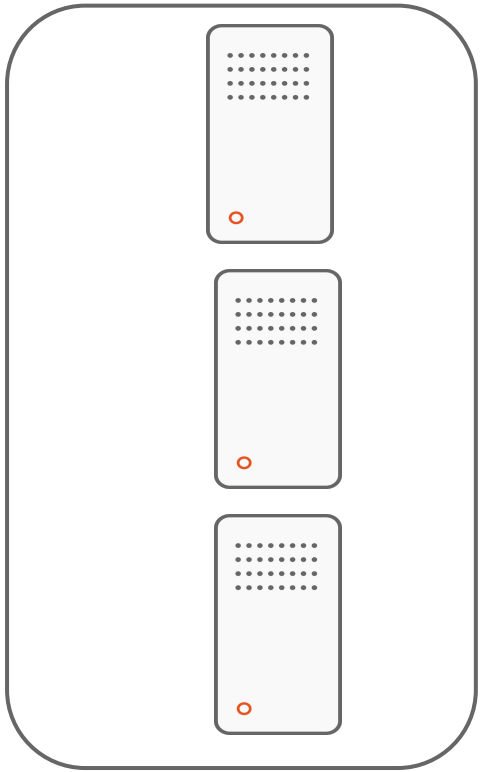
Pitfalls of running databases on K8s

Placement constraints

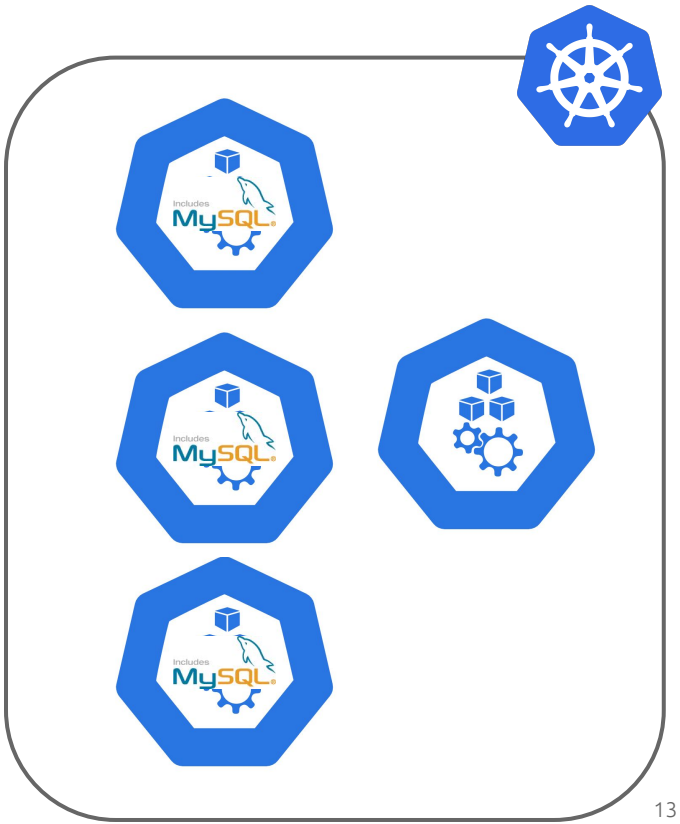
	Placement Method	Checked at Scheduling Time	Checked at Runtime
1	Node Selector	✔	✘
2	Node Affinity	✔	✘
3	Pod Affinity/Anti-Affinity	✔	✘
4	Taints & Tolerations	✔	✔ (for evictions)
5	Topology Spread Constraints	✔	✘

Pitfalls of running databases on K8s

Ingress

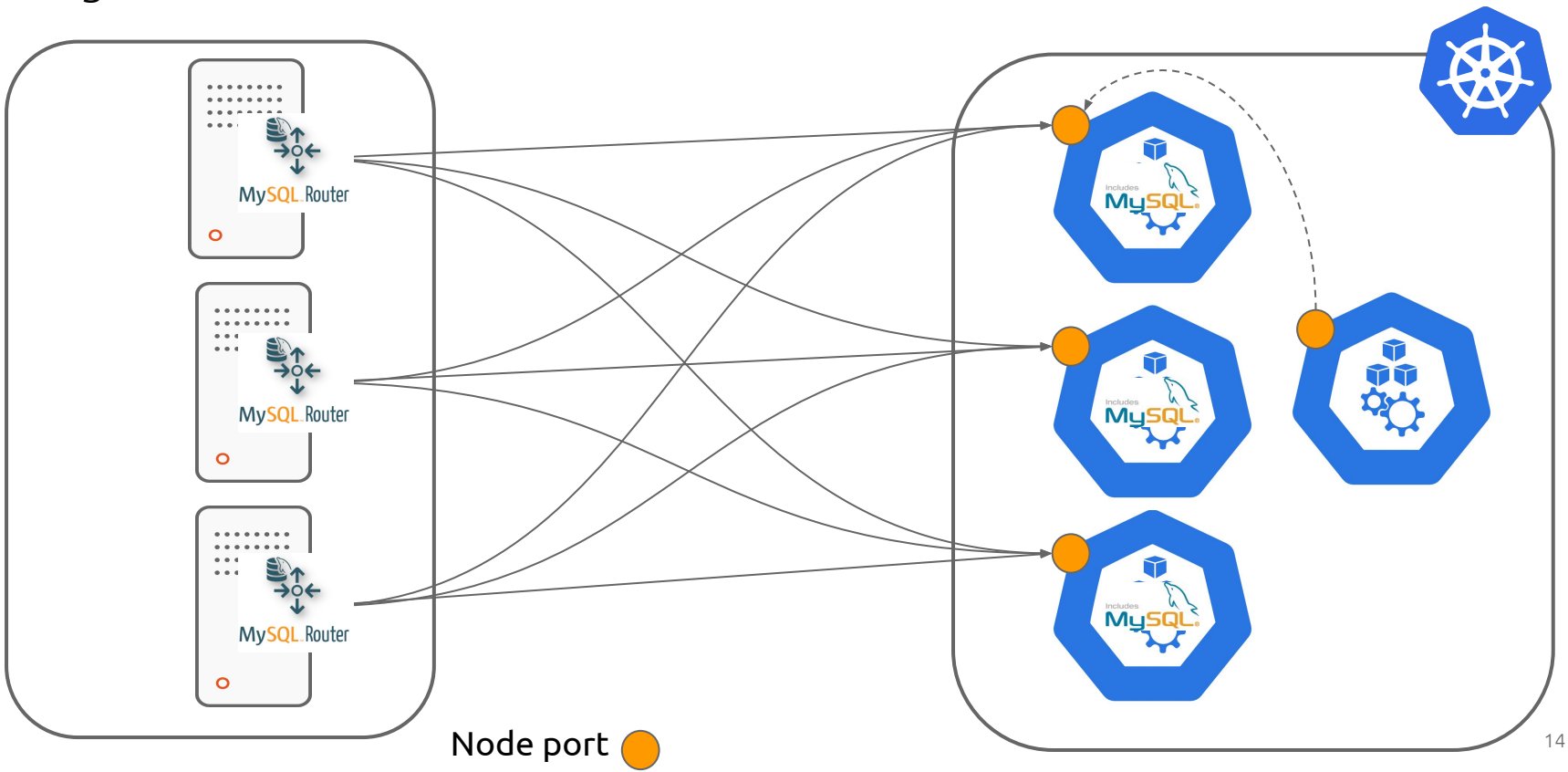


How to expose DB endpoints outside K8s?



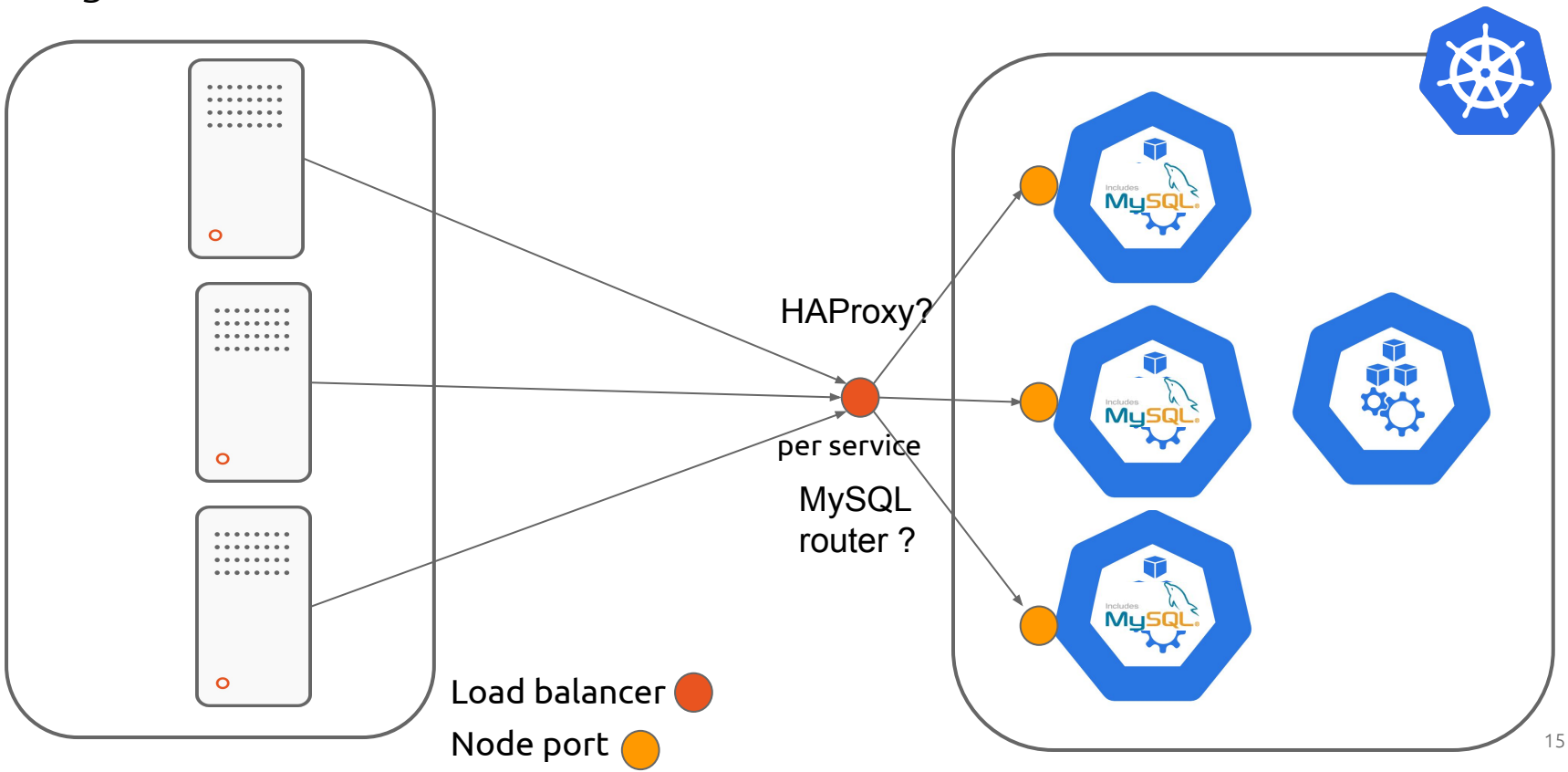
Pitfalls of running databases on K8s

Ingress



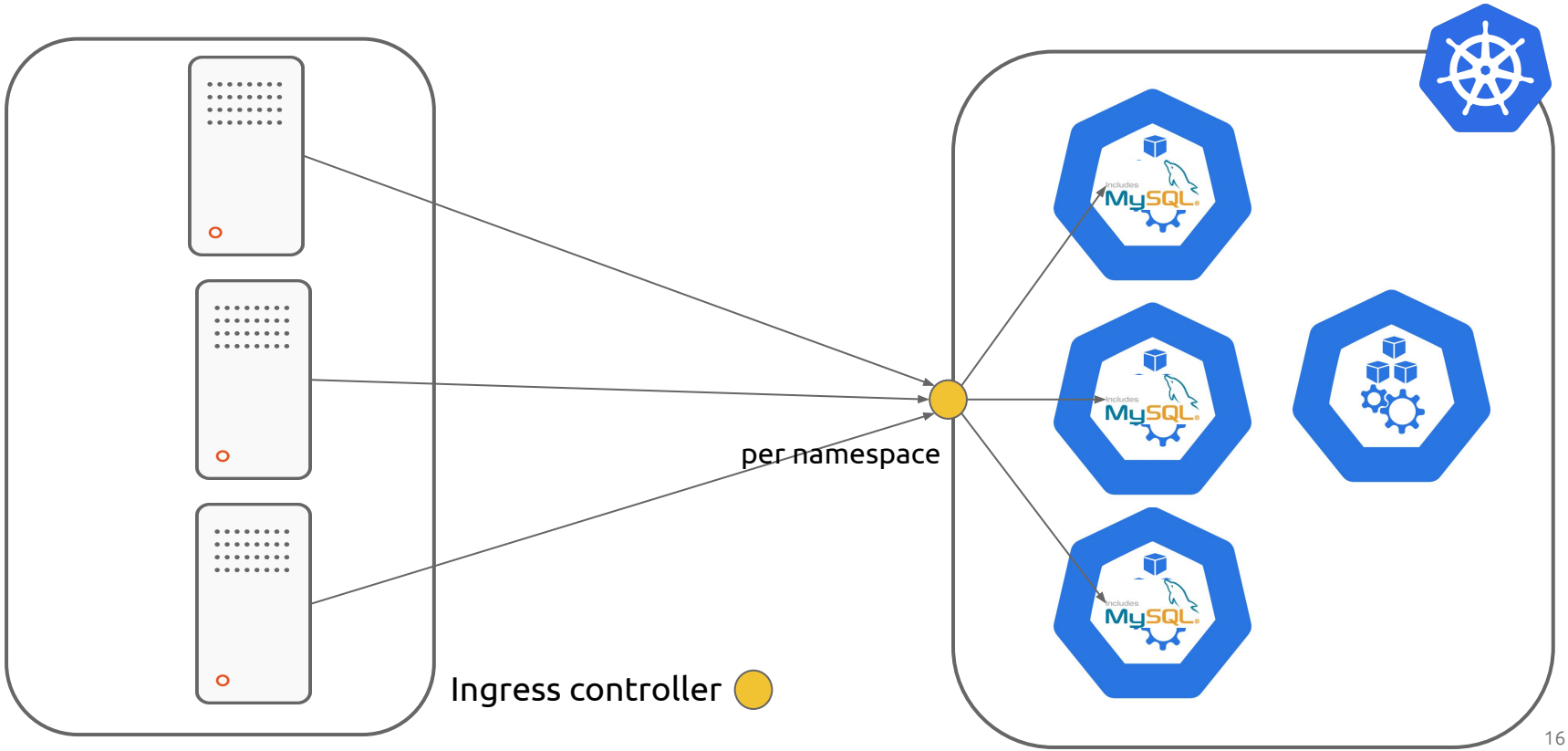
Pitfalls of running databases on K8s

Ingress



Pitfalls of running databases on K8s

Ingress



Pitfalls of running databases on K8s

Ingress

service/networking/minimal-ingress.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

Ingress controller

```yaml
apiVersion: gateway.networking.k8s.io/v1alpha2
kind: Gateway
metadata:
  name: default-namespace-gateway
  namespace: default
spec:
  gatewayClassName: haproxy-ingress-gatewayclass
  listeners:
    - allowedRoutes:
        kinds:
          - group: gateway.networking.k8s.io
            kind: TCPRoute
        namespaces:
          from: Same
      name: listener1
      port: 8000
      protocol: TCP
```

Gateway API?

Pitfalls of running databases on K8s

Upgrades

RollingUpdate

The `RollingUpdate` update strategy will update all Pods in a StatefulSet, in reverse ordinal order, while respecting the StatefulSet guarantees.

You can split updates to a StatefulSet that uses the `RollingUpdate` strategy into *partitions*, by specifying `.spec.updateStrategy.rollingUpdate.partition`. You'll practice that later in this tutorial.

Start ordinal ?

FEATURE STATE: `Kubernetes v1.31 [stable]` (enabled by default: true)

`.spec.ordinals` is an optional field that allows you to configure the integer ordinals assigned to each Pod. It defaults to nil. Within the field, you can configure the following options:

- `.spec.ordinals.start`: If the `.spec.ordinals.start` field is set, Pods will be assigned ordinals from `.spec.ordinals.start` up through `.spec.ordinals.start + .spec.replicas - 1`.

Agenda

1 Pitfalls of running databases on K8s

2 Best practices for running MySQL on K8s

3 Going further

Best practices of running MySQL on K8s

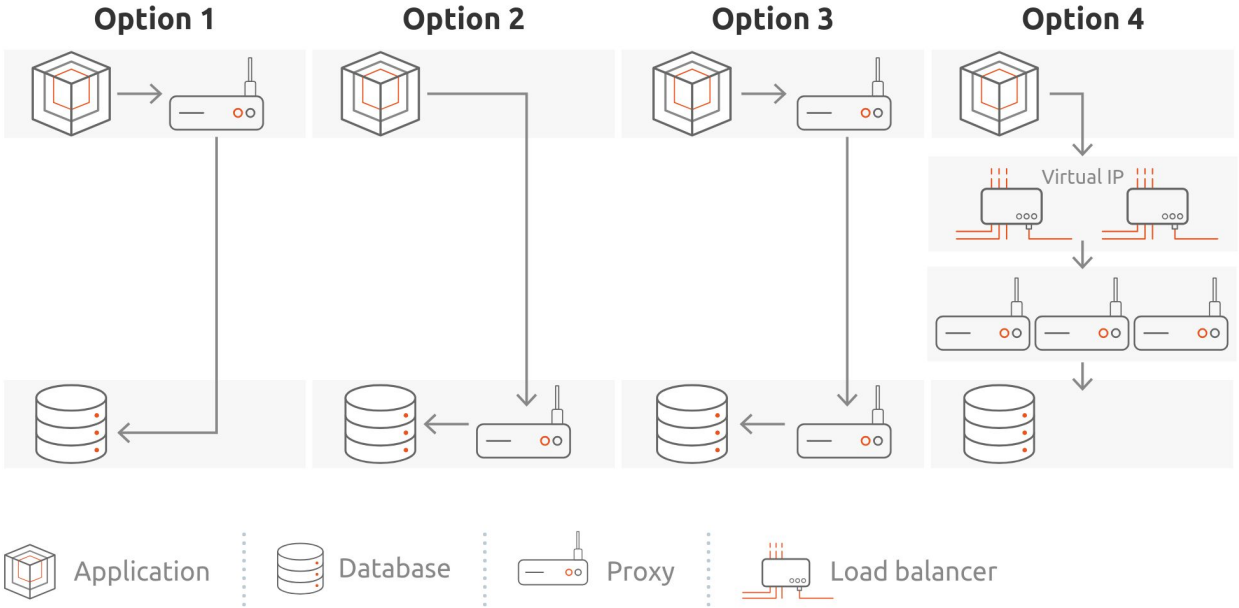
- Do you really(!) need to run your MySQL fleet on K8s ?
- Do you really(!) have to run your MySQL fleet on K8s ?
- Do you really(!) must run your MySQL fleet on K8s ?

Best practices of running MySQL on K8s

- Dedicate some worker nodes to your database workloads
  - Or even whole K8s clusters to your databases
- Consider deploying your instances across K8s clusters
- Use TopologySpreadConstraints + Taints/Tolerations to implement your placement strategy
- Consider using the host network for optimal performances and simplicity
- Consider using local storage for optimal performances

Best practices of running MySQL on K8s

- Choose wisely where you place your proxies, if you need them



Best practices of running MySQL on K8s

- Do not reinvent the wheel. Use one of the proven operators:

Charmed MySQL K8s documentation

Charmed MySQL documentation

Agenda

1 Pitfalls of running databases on K8s

2 Best practices for running MySQL on K8s

3 Going further

Going further

What we need to make running DBs on top of K8s less painful:

- More holistic approach to placement constraints:
  - mysql(i).p1 **is different from** mysql(j).p2 ⇔ Distance(p1,p2) > 0
  - mysql(i).p1 **is at least <X> far from** mysql(j).p2 ⇔ Distance(p1,p2) > X
  - mysql(i).p1 **is as far as possible from** mysql(j).p2 ⇔Maximize distance(p1,p2)

Going further

What we need to make running DBs on top of K8s less painful:

- More holistic approach to placement constraints:
  - mysql(i).p1 **is different from** mysql(j).p2 ⇔ Distance(p1,p2) > 0
  - mysql(i).p1 **is at least <X> far from** mysql(j).p2 ⇔ Distance(p1,p2) > X
  - mysql(i).p1 **is as far as possible from** mysql(j).p2 ⇔Maximize distance(p1,p2)

- Simpler ingress semantics
  - Node port but exposed only where the concerned pods are scheduled ?

Going further

What we need to make running DBs on top of K8s less painful:

- More holistic approach to placement constraints:
  - mysql(i).p1 **is different from** mysql(j).p2 ⇔ Distance(p1,p2) > 0
  - mysql(i).p1 **is at least <X> far from** mysql(j).p2 ⇔ Distance(p1,p2) > X
  - mysql(i).p1 **is as far as possible from** mysql(j).p2 ⇔Maximize distance(p1,p2)

- Simpler ingress semantics
  - Node port but exposed only where the concerned pods are scheduled ?

- An open-source and DB-aware deployment controller
  - Minimize primary switches
  - Minimize SST

Thank you! Questions?