Sure you can run your database in kubernetes
Successfully run your MySQL NDB Cluster in kubernetes

Bernd Ocklin
Snr Director MySQL Cluster Development
About me

Bernd Ocklin

Product Owner MySQL NDB Cluster at Oracle

with NDB and MySQL since 2005
Massively linear scale
Read- and Write Scale-Out
TBs on commodity hardware.

Always Consistent
Transactionally consistent across
distributed and partitioned dataset.

Always-On 99.9999% Availability
Designed for mission critical systems. Masterless, shared-nothing with no single point of failure.

Ease of use
Out of the box straightforward application programming. Standalone or with MySQL as a SQL front-end.

Shared Nothing
Written in C++. Can be used standalone
or with MySQL as a SQL front-end.

Open Source
Written in C++.

Shared Nothing
Written in C++. Can be used standalone
or with MySQL as a SQL front-end.
MySQL Cluster Industries

- Telecom
- Gaming & Massive Parallel Online Games
- Financials
Why Cloud Native?

**Speed**
Fast introduction of new services

**Scaling**
Fast scaling from hundred of users to millions

**Efficient Operations**
Automation
Lifecycle

**Performance / Capacity**
Improved capacity
Better resource utilisation
Fit for kubernetes?
VMs or container?

<table>
<thead>
<tr>
<th>Feature</th>
<th>VM</th>
<th>Containers / K8</th>
</tr>
</thead>
<tbody>
<tr>
<td>workload isolation</td>
<td>++</td>
<td>0 / ++ *) e.g. katacontainers</td>
</tr>
<tr>
<td>performance</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>IO</td>
<td>0</td>
<td>++</td>
</tr>
<tr>
<td>operations</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td>maturity / community / best practices</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>footprint</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>cloud native principles</td>
<td>0</td>
<td>++</td>
</tr>
</tbody>
</table>
But running databases in a container and Kubernetes?

• Yes, you can run any database in a container. Period.
• Just a matter of workload to serve and requirements.
• Milage varies with database’s suitability.
Suitable databases and cloud native principles

Resilience

Losing parts of the system should not be a big deal. It should automatically recover and heal itself.

Shared-Nothing

Cloud-native databases can operate without centralized management or any single point of failure.

Scaling, sharding

Distributed data
Scaling out, not up
Sharding.

Consistency

Distributed, cloud-native databases should present a the same view of data independent of instance queried. With the consistency guarantees of a single-machine system.

Standards

Cloud-native databases should also support query standards.
## Cloud native databases

<table>
<thead>
<tr>
<th>Feature</th>
<th>MySQL NDB</th>
<th>classic RDMS</th>
<th>InnoDB Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resilience</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Shared-nothing</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Consistent view of data</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Scaling out, sharding</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Standard query language</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Self healing</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Stateless?

- You should architect your system to be *intentional* about when, and how, you store state
- Design components to be *stateless wherever you can*
- Not stateless but smart about state, state optimized!
Kubernetes
<table>
<thead>
<tr>
<th>Analytics Reporting</th>
<th>Orchestration</th>
<th>Operation</th>
<th>Orchestration &amp; Automation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL NDB</td>
<td>State“less” Microservices</td>
<td>Data Layer</td>
<td></td>
</tr>
</tbody>
</table>

**Prometheus** | **Grafana** | **envoy** | **fluentd** | **Jaeger** | **Istio** | **Helm** | **Platform Services** |

**Highly Available Object Store**

MySQL NDB

**kubernetes** | **Docker** | **(Cloud) Infrastructure**
Kubernetes Objects running a database

```yaml
... containers:
  - image: mysql/mysql-cluster:8.0.22
    imagePullPolicy: IfNotPresent
    name: ndb
    command: ["/bin/bash"]
    args:
      - -ecx
      - /usr/sbin/ndbd -c mgmd-0.ndb-svc.default.svc.cluster.local \
        -initial --nodaemon -v
```
Kubernetes Objects running a database

Pod

Container

Container

docker hub

docker hub
Workload Resources

---

**Workload API**

- Pod
- Pod
- Container
- Container

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: StatefulSet
...
spec:
...
serviceName: ndb-svc
template:
  spec:
    containers:
    - image: mysql/mysql-cluster:8.0.22
      ...
    volumeMounts:
    - name: ndb-persistent-storage
      mountPath: /var/lib/ndb
    - name: config-volume
      mountPath: /var/lib/ndb/config
    ...
```
Running MySQL Cluster in Kubernetes with StatefulSets

- Stable, unique network identifiers.
- Stable, persistent storage.
- Ordered, graceful deployment and scaling.
- Ordered, automated rolling updates.
Headless Service providing network identity

```
apiVersion: v1
kind: Service
metadata:
  name: ndb-svc
  labels:
    app: ndb
spec:
  ports:
    - port: 1186
  selector:
    app: ndb
clusterIP: None
```
ConfigMaps to “inject” configuration into Pods/Containers

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: ndb-configmap
  namespace: default
data:
  config.ini:
    [ndbd default]
    # NDB redundancy level
    NoOfReplicas=3
    ...
```
Use Persistent Volumes for storage

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ndb-pv-claim
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```
NDB Architecture

“stateless” Microservices

optional SQL Layer

MySQL NDB Cluster

Partitioning- and distribution engine
MySQL NDB Cluster in Kubernetes

Pod

Sidecar container

Data Node container

Config

Data- and trace files

Persistent Volume Claim

Persistent Volume

Pod

Sidecar container

Management Node container

ConfigMap
MySQL NDB Cluster in Kubernetes

Deployment

StatefulSet

Pod Data Node
Pod Data Node
Pod Mgm Node
Pod Mgm Node
Pod Mgm Node
StatefulSet

kubectl
Demo - deploying manually

https://www.github.com/ocklin/ndb-k8-manually
Best practices
DNS “stability”

- Pods
  - reschedule on other Kubernetes nodes
  - change IP addresses
  - consider DNS TTL, time to resolve new host address
  - ! **GRANT** ... TO
    - ‘username’@<IP-address>
  - Use `AllowUnresolvedHostnames=1`
  - Retry
Service Mesh Istio

- Envoy is a proxy
  - connects to cluster will "look like" connects from localhost
  - cluster expects connects from remote host
  - use TcpBind_INADDR_ANY = 1
Sidecars

- Always use a most minimal maintenance container
  - idle, low resource
  - but allows parallel access to volumes and stored data
  - easier debugging if things go wrong
Kubernetes is complex

- Many layers and teams responsible
- Lots of people or resources to blame if something goes wrong
- Observability is key!
PodDisruptionBudgets and Eviction API

- Eviction API considers pod disruption budgets
  - e.g. used when draining kubernetes nodes
- makes sure that you do not accidentally shutdown all your nodes of the database
- `kubectl delete` ignores PodDisruptionBudgets!
PodAffinity and AntiAffinity

- All nodes have labels, make heavy use of labels!
- PodAffinity allows to prefer k8 nodes with labels to e.g.
  - keep database nodes apart across racks or ADs
  - avoid collocation of instances sharing same data
  - prefer faster storage (e.g. SSD)
MySQL NDB Operator
Kubernetes Operators

- Declarative approach
- Manages services “like a human”
- Based on Custom Resource Definitions
Operator reconciliation driving towards desired state

$ kubectl apply -f operator-crd.yaml
MySQL NDB Cluster in Kubernetes

Deployment
MySQL
StatefulSet
MySQL
MySQL
Data Node
Data Node
Mgm Node
Data Node
Data Node
Mgm Node
NDB Cluster

Ndb operator
mysql controller
NDB controller
backup / restore controller

kubectl

Kubernetes
Operator Demo
Ndb Custom Resource Definition

```yaml
apiVersion: mysql.oracle.com/v1alpha1
kind: Ndb
metadata:
  name: example-ndb
spec:
  containerImage: mysql/mysql-cluster:8.0.22
  nodecount: 2
  redundancyLevel: 2
  mysqld:
    nodecount: 2
```
Thank You

Bernd Ocklin
Snr Director
MySQL Cluster Development