# Running Apache Cassandra™ in Kubernetes

Hands on with K8ssandra

# Presenters

**Michael Burman**
Senior Software Engineer
@IBM (DataStax)
Formerly Red Hat

K8ssandra maintainer

**Alexander Dejanovski**
Senior Software Engineer
@IBM (DataStax)
Formerly The Last Pickle

K8ssandra maintainer

# Agenda

**Intro**
Setup
What is K8ssandra?

**Hands on**
Create a cluster
Schedule repairs
Upgrade
Expansion
Maintenance ops
Create a new DC

**Extras**
Replacing dead nodes
Monitoring

# Step 1: Setup

- Clone this repo: github.com/k8ssandra/cocna25
- Docker is required (Docker Desktop preferred)
- See the step1 folder README.md to create a kind cluster:

```
cd cocna25
kind create cluster --name k8ssandra -config ./kind/w4k1.32.yaml
```

- Pull the Cassandra images now

```
docker pull k8ssandra/cass-management-api:5.0.3-ubi
docker pull k8ssandra/cass-management-api:5.0.4-ubi
```

# What is K8ssandra?

A set of Kubernetes operators orchestrating the deployment of Apache Cassandra™ clusters, with multi-DC/multi-cluster support.

## Components

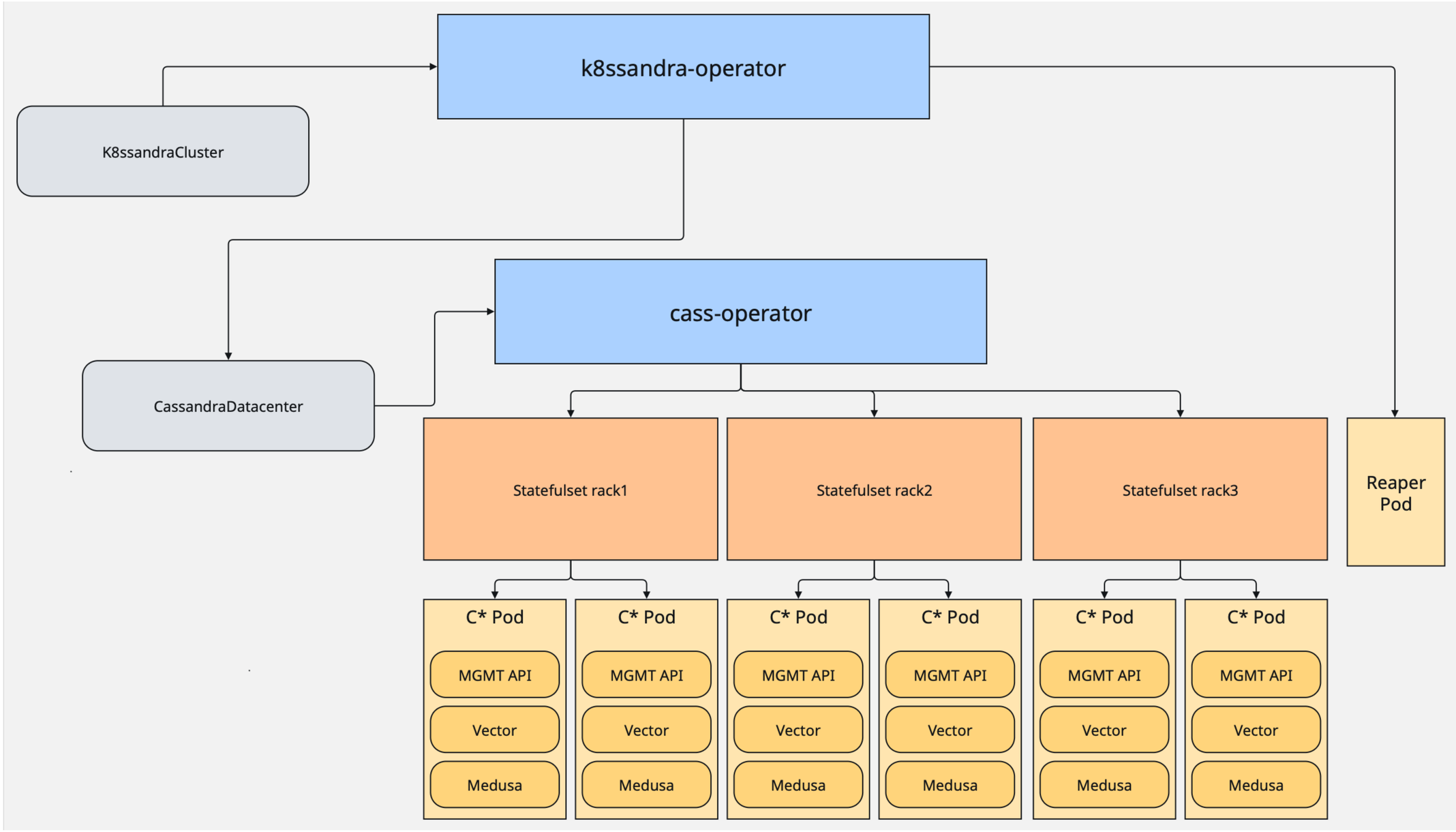
- cass-operator: single dc orchestration for Cassandra
- k8ssandra-operator: multi dc orchestration for Cassandra, with repair and backup support
- The k8ssandra management API: REST alternative to JMX for node management
- Reaper: anti-entropy repair orchestration
- Medusa: Backup and Restore for Cassandra
- Vector: Observability pipelines tool

# Backup/Restore

Check Aaron Ploetz' session Saturday at 3:10pm:

Cassandra backups made easy with Medusa and Kubernetes

# K8ssandra Architecture



# Step 2: Install k8ssandra-operator

1. Install cert-manager (required)
2. Install k8ssandra-operator

Check the **README.md** in the **step1** folder in the cloned repo

# Step 3: Create a K8ssandraCluster

Create a K8ssandraCluster object using kubectl

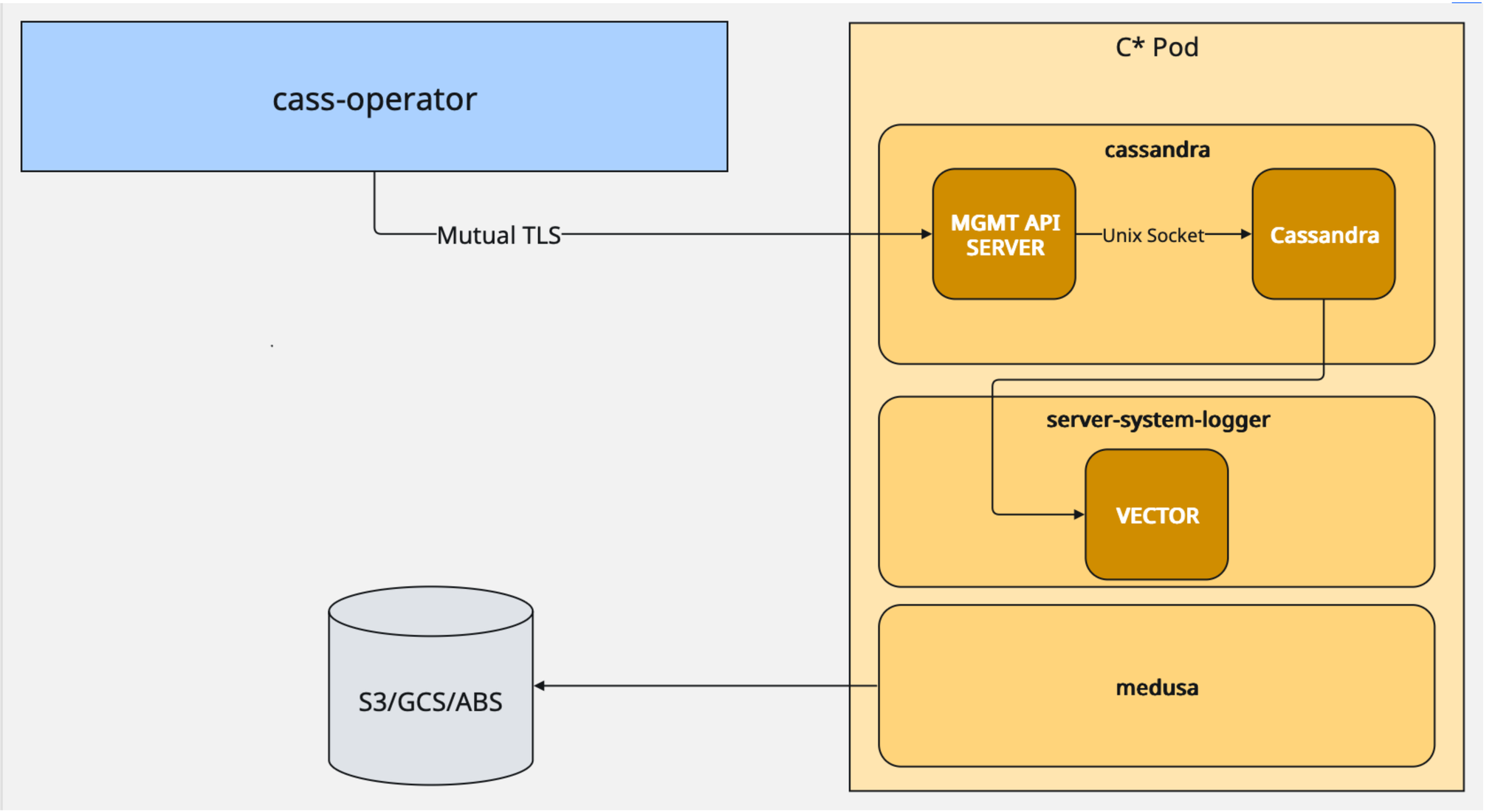
Check the **README.md** in the **step2** folder in the cloned repo

```
kubectl apply -f ./step3/k8ssandra-cluster.yaml -n db
```

Eventually you should have 3 pods in the db namespace:

```
NAME                       READY   STATUS    RESTARTS   AGE
test-dc1-rack1-sts-0       2/2     Running   0          7m57s
test-dc1-rack1-sts-1       2/2     Running   0          7m57s
test-dc1-reaper-0          1/1     Running   0          2m40s
```

# Management API



# Step 3: Checking the logs

There are multiple containers running in a database pod:

- cassandra: runs the management-api server as primary process and outputs its log in the standard output
- server-system-logger: runs an instance of vector (vector.dev) which outputs the system.log file from the cassandra process. Also manages customizable observability pipelines

Tail the system.log output:

```
kubectl logs test-dc1-rack1-sts-0 -c server-system-logger -n db --follow
```

# Step 3: Create keyspaces

Read and decode the test-superuser secret username and password:

```
kubectl get secret test-superuser -o jsonpath='{.data.username}' -n db | base64 --decode
kubectl get secret test-superuser -o jsonpath='{.data.password}' -n db | base64 -decode
```

"Exec" into one of the db pods:

```
kubectl exec -i -t -n db test-dc1-rack1-sts-0 -c cassandra -- sh -c
"clear; (bash || ash || sh)"
```

Check the topology by running nodetool:

```
nodetool status
```

# Step 3: Create keyspaces

Run cqlsh:

```
cqlsh -u <superuser name> -p <superuser password> test-dc1-service
```

Create the keyspaces:

```
create keyspace myks1 WITH replication = {'class': 'SimpleStrategy',
'replication_factor': 2};
create keyspace myks2 WITH replication = {'class': 'SimpleStrategy',
'replication_factor': 2};
create table myks1.table1 (id int PRIMARY KEY, val text);
create table myks2.table2 (id int PRIMARY KEY, val text);
```

# Step 4: Scheduling repairs

We have a Reaper instance deployed and configured automatically.
Let's connect to the UI:

```
kubectl port-forward svc/test-dc1-reaper-service -n db 8080:8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

Then open your browser at: http://127.0.0.1:8080/webui

# Step 4: Scheduling repairs

Enabling auto-scheduling:

```
reaper:
  autoscheduling:
    enabled: true
```

Apply the updated manifest from the step4 folder:

```
kubectl apply -f ./step4/k8ssandra-cluster.yaml -n db
```

# Step 5: Upgrading Cassandra

Upgrading Cassandra requires changing the serverVersion in the manifest:

```
apiVersion: k8ssandra.io/v1alpha1
kind: K8ssandraCluster
metadata:
  name: test
spec:
  cassandra:
    serverVersion: 5.0.4
```

Apply the updated manifest from the step5 folder:

```
kubectl apply -f ./step5/k8ssandra-cluster.yaml -n db
```

Wait for the rolling restart to go through.

# Step 5: Upgrading Cassandra

Use a K8ssandraTask to perform an upgradesstables round:

```
apiVersion: control.k8ssandra.io/v1alpha1
kind: K8ssandraTask
metadata:
  name: upgradesstables-dc1
spec:
  cluster:
    name: test
  template:
    jobs:
      - args: {}
        command: upgradesstables
        name: ''
```

Apply the manifest from step5:

```
kubectl apply -f ./step5/upgradesstables-task.yaml -n db
```

# Step 5: Upgrading Cassandra

Watch the execution status:

```
% kubectl get K8ssandraTask -n db
NAME                    JOB                 SCHEDULED   STARTED   COMPLETED
upgradesstables-dc1     upgradesstables                 4s
```

# Step 5: Available K8ssandraTasks

- Upgradesstables
- Restart
- Cleanup
- Flush
- Garbage collect
- Compact
- Scrub

# Step 6: Expanding to a new DC

Expanding a cluster to a new DC requires the addition of a new entry in .spec.cassandra.datacenters:

```
datacenters:
  - metadata:
      name: dc1
    size: 2
    racks:
      - name: rack1
  - metadata:
      name: dc2
    size: 2
    racks:
      - name: rack1
```

Apply the updated manifest from the step6 folder:

```
kubectl apply -f ./step6/k8ssandra-cluster.yaml -n db
```

# Step 7: Replacing a dead/faulty node

Replacement of a faulty or dead (Cassandra) node is performed through a K8ssandraTask custom resource.

It will perform the following operations:

- take down the faulty pod and delete its PVC
- restart the replacement pod and inject the `-Dcassandra.replace_address_first_boot=x.x.x.x` flag

Apply the updated manifest from the step7 folder:

```
kubectl apply -f ./step7/replacenode-task.yaml -n db
```

# Step 8: Setting up monitoring

The Vector instance running in each db pod is configurable through the K8ssandraCluster CRD.

Follow the step8 folder readme to:

- Install Prometheus and Grafana
- Set up Vector to read the metrics and send them to Prometheus
- Install the dashboards and access them.