
@joerg_schad
Nightmares of a Container Orchestration System

Jörg Schad

Distributed Systems Engineer

@joerg_schad

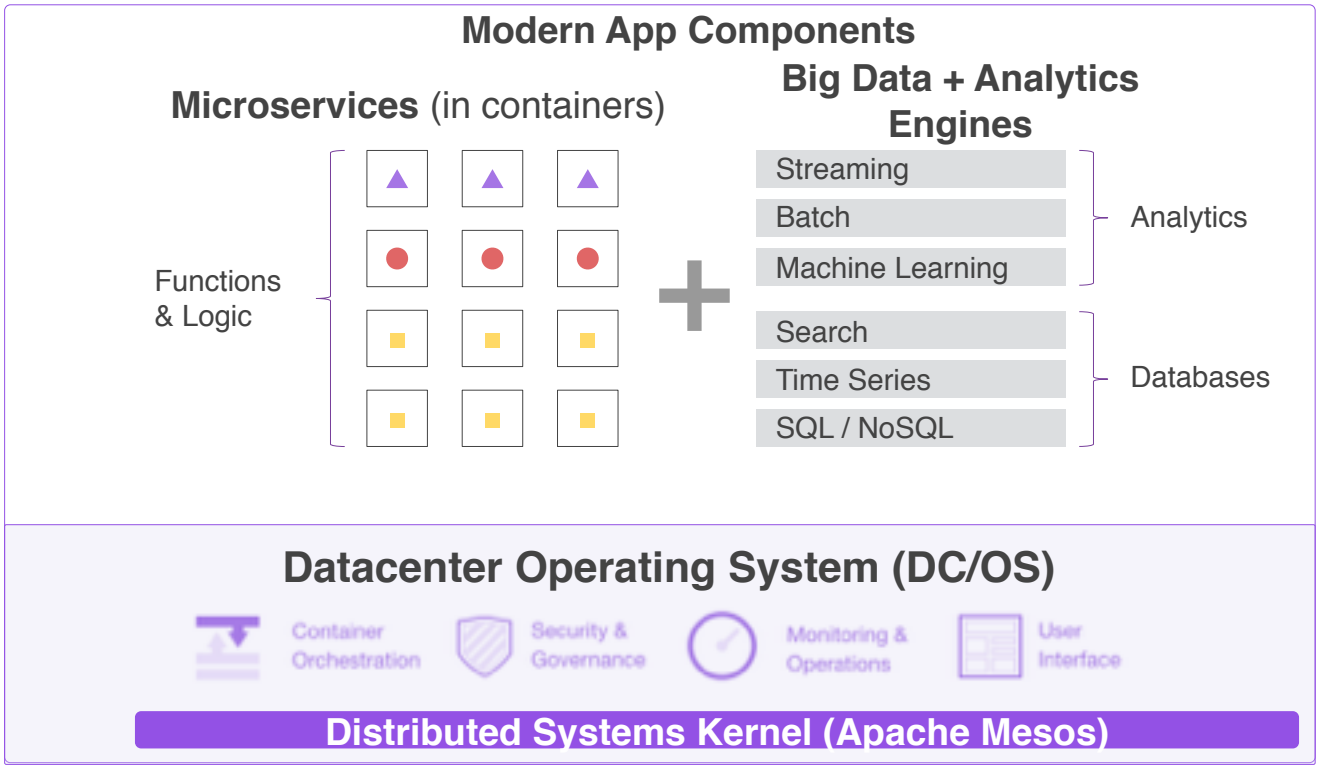
Jan Repnak

Support Engineer/
Solution Architect

@jrx

3AM…

Scope of Jan's nightmares…



Any Infrastructure (Physical, Virtual, Cloud)

Backup State

Why should we have backup?

- Backup state
  - Services
  - Cluster

Immutable Container Images

```
{
  "id": "/",
  "instances": 1,
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "ubuntu"
    }
  },
  "cpus": 0.1,
  "mem": 128,
  "cmd": "sleep 1000"
}
```

- Use tagged container images
- Keep tagged images immutable!

Private Container Registries

```
{
  "id": "/",
  "instances": 1,
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "ubuntu"
    }
  },
  "cpus": 0.1,
  "mem": 128,
  "cmd": "sleep 1000"
}
```

Dockerhub works great for our test cluster…

- Use tagged container images
- Keep tagged images immutable!
- Use a private container registry!

Repeatable Container Builds

`docker commit` is great*…

- Use repeatable builds for images
  - Including FROM clause
- Keep images minimal
  - Multistage build
  - From scratch

UI Deployments

- Use (Marathon) endpoints for deployments!
- Version (and track) your app definitions!

Disk Usage

All our disk are full…

- Docker and logs are great in filling up disk space!
  - Images
  - Container
- Cleanup docker instances and images!
  - docker prune
  - https://github.com/spotify/docker-gc
- Monitor available disk space!

Resource Constraints

```
{
  "id": "/app-server",
  "description": "App definition version 1.",
  "cpus": 0.5,
  "mem": 32,
  "disk": 0,
```

32MB are sufficient for Docker container…

- Memory constraints are hard limits
- Consider overhead (e.g., Java)
- Difficult to approximate
- Monitor

Zookeeper Cluster Size

Our Zookeeper Cluster has 4 nodes, that is better than 3, or?

- Zookeeper quorum (i.e., #Masters) should be odd!
- Production 5 is optimal!

Health Checks

```
"healthChecks": [
  {
    "path": "/",
    "portIndex": 0,
    "protocol": "HTTP"
  }
],
```

What are health checks?

- Specify Health checks carefully
  - Different options
    - Mesos vs Marathon,
    - Command vs HTTP
- Impacts Load-Balancers and restarts
- Readiness checks

NoSQL Datastores

We replaced out Postgres instance with Cassandra, and now we get stale results

- Consider the semantics of your datastore!
  - ACID vs Base
- Model your data and queries accordingly!

Removing Stateful Frameworks

- Follow the uninstall instructions!
- Reservations and Zookeeper state!
- state.json

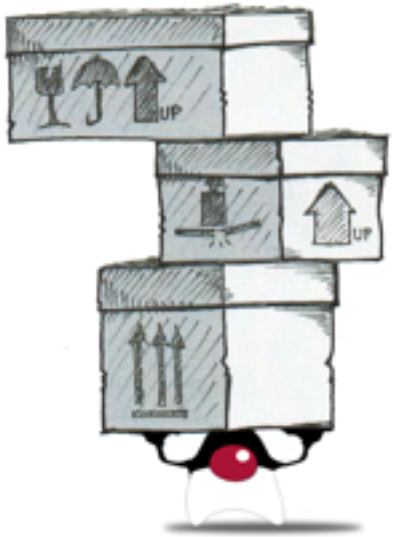
Container vs VMs

We just replaced all our VM instances by containers*…

- Be aware of different isolation semantics!

Write Once Run Any Where

The (Java) container was running fine in testing…



- Java (<9) not groups aware
  - # threads for GC
  - …
- Set default values carefully

Mesos Modules

To solve this problem, our team quickly developed this really cool Mesos Module…

- Mesos Modules can be tricky!
- Monitoring and Debugging…

Linux Distributions

We are using *obscure Linux distribution* for our (DC/OS) cluster

- If possible use tested distributions!
- Especially for DC/OS!

Services on the same node…

We are running *distributed Database* outside Mesos on the same cluster

- Be careful when running services outside Mesos but on the same cluster!
- Adjust resources accordingly!

Spreading out Master Nodes

We are running our cluster across different AWS regions..

- Be careful when distributing Master nodes across high latency links!
- Different AWS AZ ok, different region probably not!

Agent Attributes

```
—attributes='rack:abc;zone:west;
os:centos5;level:10;keys:[1000-1500]'
```

We changed the agent attributes for running cluster…

- Set agent attributes when starting an agent!
- Do not change for running agents!

Cluster Upgrades

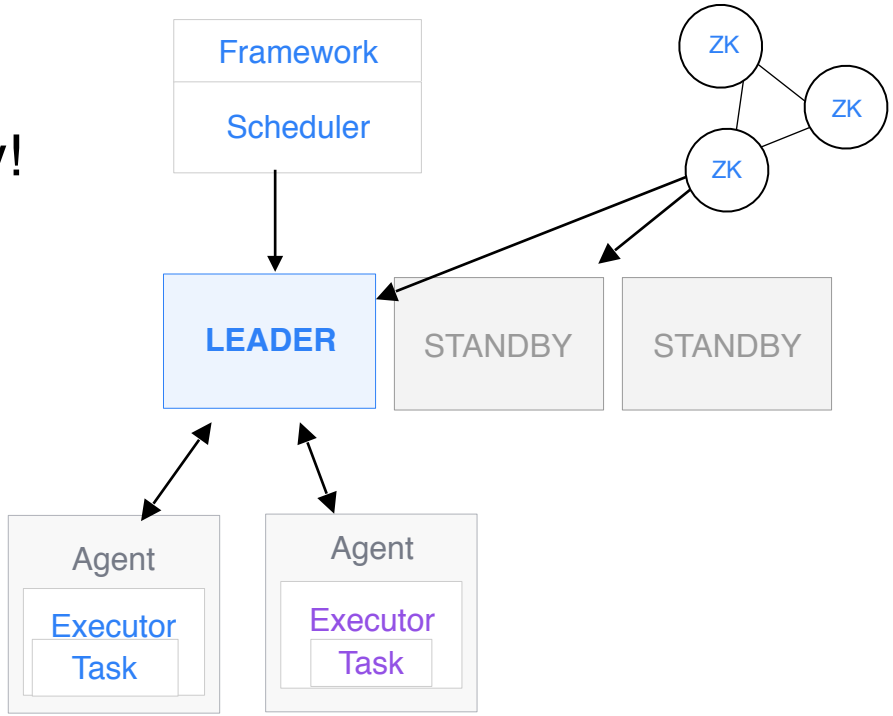
We upgraded our cluster…*

- Check state before
- Follow upgrade instructions!
- Automation
- Remember Backup!

UPGRADE PROCEDURE

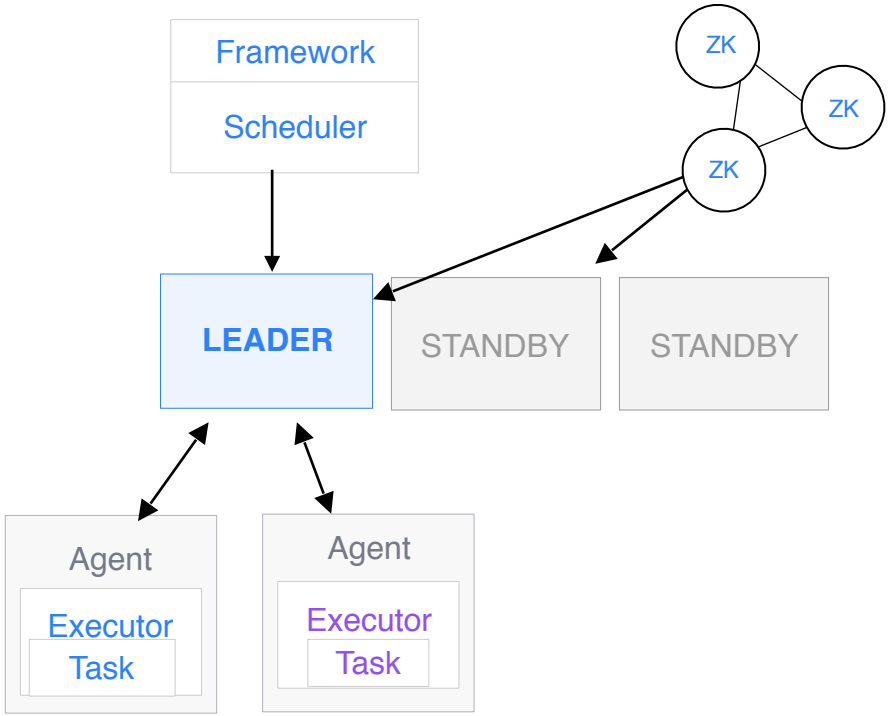
Before upgrading

1. Make sure cluster is healthy!
2. Perform backup
   a. ZK
   b. Replicated logs
   c. other state
3. Review release notes
4. Generate install bundle
   a. Validate versions



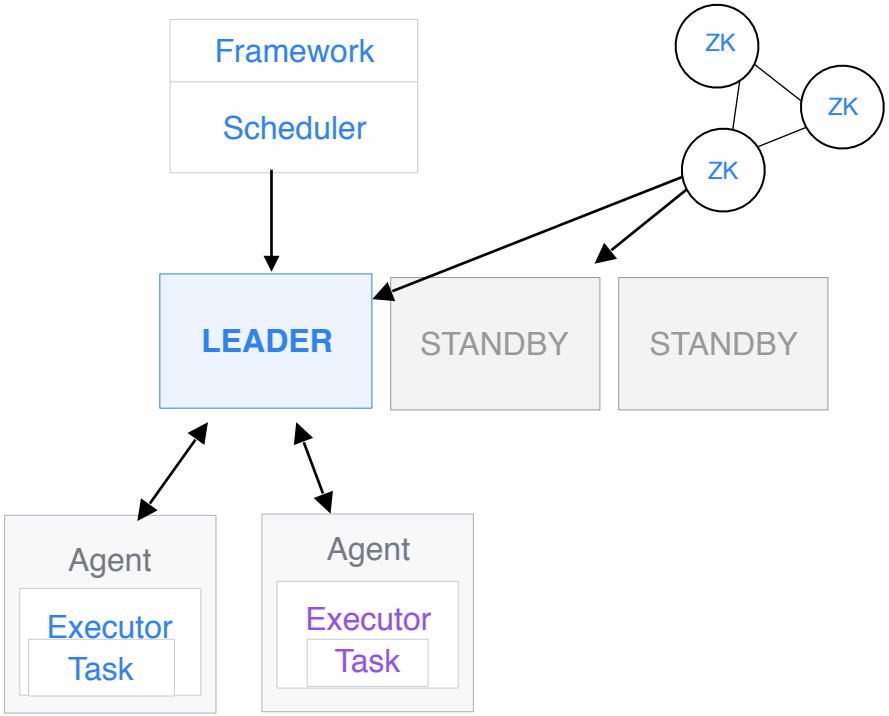
UPGRADE PROCEDURE

1. Master rolling upgrade
   a. Start with standby
   b. Uninstall DC/OS
   c. Install new DC/OS
2. Agent rolling upgrade
3. Framework upgrades



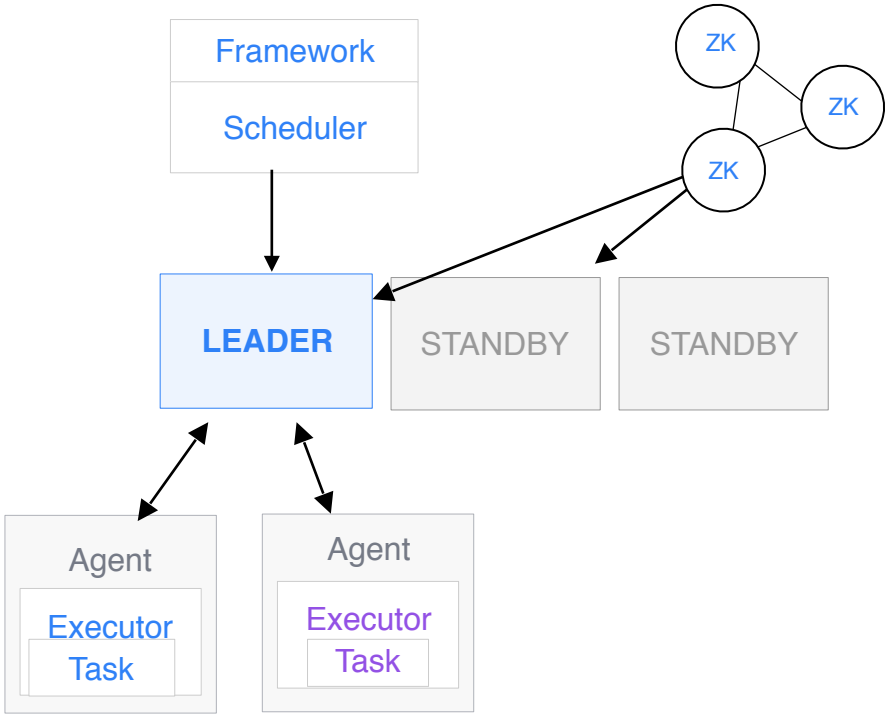
UPGRADE PROCEDURE

1. Master rolling upgrade
2. Agent rolling upgrade
   a. Uninstall DC/OS
   b. Install new DC/OS
3. Framework upgrades



UPGRADE PROCEDURE

1. Master rolling upgrade
2. Agent rolling upgrade
3. Framework upgrades
   a. Orthogonal to DC/OS
   b. Ensure changes don't affect existing apps



Software Upgrades

We have automatic updates enabled for Docker…

- Follow upgrade instructions!
- Backup!
- **Explicit control of versions!**

Day 2 Operations

Our POC app is deployed in our production environment, time for vacation…

- Day 2 Operations is the actually challenging part!

Day 2 Operations

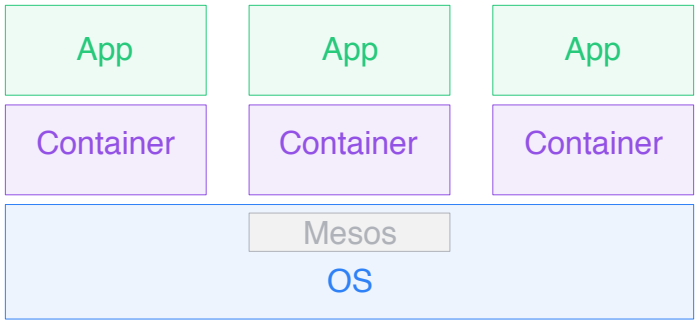
- Configuration **Updates** (ex: Scaling, re-configuration)
- Binary **Upgrades**
- Cluster **Maintenance** (ex: Backup, Restore, Restart)
- **Monitor** progress of operations
- **Debug** any runtime blockages

METRICS

- Measurements captured to determine health and performance of cluster
  - How utilized is the cluster?
  - Are resources being optimally used?
  - Is the system performing better or worse over time?
  - Are there bottlenecks in the system?
  - What is the response time of applications?

DC/OS METRIC SOURCES

- Mesos metrics
  - Resource, frameworks, masters, agents, tasks, system, events
- Container Metrics
  - CPU, mem, disk, network
- Application Metrics
  - QPS, latency, response time, hits, active users, errors



Production Checklist

MESOS CHECKLIST

- ❑ Monitor both Masters and Agents for flapping (i.e., continuously restarting). This can be accomplished by using the `uptime` metric.
- ❑ Monitor the rate of changes in terminal task states, including TASK_FAILED, TASK_LOST, and TASK_KILLED

MESOS MASTER CHECKLIST

- ❑ Use five master instances in production. Three is sufficient for HA in staging/test
- ❑ Place masters on separate racks, if possible
- ❑ Secure the teardown endpoints to prevent accidental framework removal.

MESOS AGENT CHECKLIST

- ❑ Set agent attributes before you run anything on the cluster. Once an agent is started, changing the attributes may break recovery of running tasks in the event of a restart. See also https://issues.apache.org/jira/browse/MESOS-1739.
- ❑ Explicitly set the resources on the nodes to leave capacity for other services running there *outside* of Mesos control. For example, HDFS processes running alongside Mesos.

ZOOKEEPER CHECKLIST

- Run with security and ACLs, see the `--zk=` and `--master=` flags on the master and slaves respectively. If you do enable ACLs, they must be enabled **before** nodes are created in ZK.
- Backup ZooKeeper snapshots and log at regular intervals. -
  - Guano or zkConfig.py (Want Snapshots + Transaction Log)
- Marathon, Chronos, and other frameworks store state in ZK. The first Marathon should store state in the same ZK as Mesos master.
- Userland apps ***should NOT*** store state in the ZK cluster shared by Mesos and Marathon. Examples of userland apps include Storm, service discovery tools, and additional instances of Marathon and Chronos.

ZOOKEEPER CHECKLIST

- ❑ Monitor ZK's JVM metrics, such as heap usage, GC pause times, and full-collection frequency.
- ❑ Monitor ZK for: number of client connections, total number of znodes, size of znodes (min, max, avg, 99% percentile), and read/write performance metrics

@joerg_schad @dcos

ANY QUESTIONS?

@dcos

chat.dcos.io

users@dcos.io

/groups/8295652

/dcos
/dcos/examples
/dcos/demos