Stateful Services on DC/OS

Santa Clara, California | April 23th – 25th, 2018
Who Am I?

- Shafique Hassan
- Solutions Architect @ Mesosphere
- “Operator”
Agenda

- DC/OS Introduction and Recap
- Why Stateful Services on DC/OS?
- Introduction to the DC/OS SDK
- Demo
  - Deploying a Data Service on DC/OS
- Wrap-Up and Summary
Takeaways for this session

- Why DC/OS is the best place to run stateful services
- Introduction to the DC/OS SDK and how you can leverage it to build your own stateful services on DC/OS
Why Stateful Services on DC/OS?
### DC/OS 101

**1. Broad workload coverage**
Run today & tomorrow’s applications including traditional J2EE, containers, analytics & ML

**2. Application-aware automation**
Automate workload-specific operating procedures to “as-a-Service” anything from Kubernetes to data services

**3. Intelligent resource pooling**
Optimize workload density for highest utilization with resource guarantees

**4. Unified hybrid cloud operations**
Securely manage cloud, datacenter, and edge infrastructures from a single control plane
DC/OS Hybrid Cloud

Edge and Multi-Cloud Federation

- Minimize footprint at edge or remote infrastructures
- Consistent operations across clouds

Business Continuity & Disaster Recovery

- Deploy applications to multiple clouds simultaneously
- Workloads automatically deployed across fault domains (Racks or Cloud Availability Zones)

Cloud Bursting

- Easily add and remove cloud capacity to on-premise clusters
The DC/OS Catalog

Over 100 Services Made For Enterprise DC/OS

- Fast Data and Big Data
- Scalable datacenter-wide services
- Open source & Partner-supported packages

Mesosphere Enterprise DC/OS
Why Run Stateful Services on DC/OS?

<table>
<thead>
<tr>
<th></th>
<th>On-demand provisioning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Single command install of services</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Simplified operations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Runtime software upgrade</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Runtime application settings update</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Monitoring &amp; metrics</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Managed persistent storage volumes</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Elastic data infrastructure</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Data services and containerized applications share resources</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Deploy instances with different versions on the same infrastructure fabric</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Resize instances</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add more instances</td>
<td></td>
</tr>
</tbody>
</table>
The SMACK Stack

- **EVENTS**: Ubiquitous data streams from connected devices
- **FEEDS**: Ingest millions of events per second
- **ANALYTICS**: Real-time and batch process data
- **STORAGE**: Distributed & highly scalable database
- **REACTIVE APP**: Scalable, resilient, data driven applications

- Integrated set of data services to ingest, analyze, and store streaming data
- Simple deployment and operations to get your apps to market faster
- Highly available so you don’t miss a single customer interaction
- Increased utilization of hardware and cloud resources through workload consolidation

**Mesosphere DC/OS**
DC/OS Summary

Traditional Approach:
Slow, Expensive, Hard

- Manual & applications-specific configurations are slow and difficult to maintain
- Cluster sprawl and low utilization
- High risk with unique “snowflake” configurations in cloud or datacenter silos

DC/OS Approach:
Datacenter-cloud as a single computer

1. **Application-aware automation** for complete lifecycle automation of platform services
2. **Workload pooling** and density optimization for dramatic cost savings
3. **Unified hybrid cloud operations** with high availability, security, and multi-tenancy
The DC/OS SDK
DC/OS SDK

- A declarative orchestration abstraction for Apache Mesos and DC/OS
- An Apache Mesos scheduler factory
- Simplify the framework development process
- Current frameworks include MongoDB, Kubernetes, Kafka, Cassandra, Elastic, HDFS, EdgeLB, Zookeeper, Jenkins, Spark and more on the way
Components of a Service

- **Mesos**
  - Foundation of a DC/OS cluster; Resource manager

- **Zookeeper**
  - SDK Schedulers use Zookeeper as their persistent store across restarts

- **Marathon**
  - “Init system” of a DC/OS cluster

- **Scheduler**
  - Management layer of the service; exposes endpoints and maintains services nodes

- **Packaging**
  - Packaging schema for SDK services; defines how options are exposed
Mesos Recap: Anatomy of a Resource Offer

Available compute resources

Mesos Master(s)

Master offers resources to scheduler

Scheduler

Scheduler accepts or declines an offer

Resource offer accepted, launch executors/tasks

Mesos Agent
16 CPUs
128 GB RAM
1 TB disk

Executor
Tasks

Mesos Agent
16 CPUs
128 GB RAM
1 TB disk

Executor
Tasks

Mesos Agent
16 CPUs
128 GB RAM
1 TB disk

Executor
Tasks

Tasks

Tasks

Tasks
DC/OS SDK

SDK

Platform

Services

Kafka

Cockroach

Spark

Best Practices

Platform Feature Integration

Apache Mesos API

Tools and Utilities

Documentation

Finite State Machine
Execution Plans
Automated Recovery

Universe Packaging
App Configuration
Networking & Discovery
Storage
Security
Monitoring

Offer Evaluation
Resource Accounting
Task Reconciliation

Developer Environment
Integration Test Framework

Developer Guide
Tutorials & Code Samples
API Reference

DC/OS
DC/OS SDK Features

- Horizontal scale out
- Vertical scaling
- Service discovery
- Virtual Networks (CNI)
- Readiness checks
- Health checks
- Custom recovery
- Persistent volumes
- Resource sets
- Operator friendly tools (API)

- Sidecars
- Placement constraints
- Configuration templating
- Rolling updates (configuration)
- Rolling upgrades (binaries)
- GPUs
- Fine-grained plan control
- Secrets (EE)
- Security (EE)
- TLS provisioning (EE)
DC/OS SDK Anatomy

POD: What?

PLAN: How and When?
DC/OS SDK Anatomy: Pods

pods:
  kafka:
    count: {{BROKER_COUNT}}
    placement: {{PLACEMENT_CONSTRAINTS}}
    tasks:
      broker:
        cpus: {{BROKER_CPUS}}
        memory: {{BROKER_MEM}}
        goal: RUNNING
DC/OS SDK Anatomy: Plans

plans:
  deploy:
    strategy: serial
  phases:
    Deployment:
      strategy: {{DEPLOY_STRATEGY}}
      pod: kafka
Why build Stateful Services using the DC/OS SDK?

- **Ease of install**: DC/OS UI and DC/OS CLI
- **Persistent storage volumes**: DC/OS reservations and persistent storage volumes for data safety and durability.
- **Runtime configuration update**: Update configuration during runtime.
- **Runtime software upgrade**: Upgrade software during runtime.
- **Fault domain aware placement and data replication**: Frameworks automatically provision nodes and intelligently replicate data across fault domains.
- **Monitoring and metrics**: Frameworks send metrics to customer provided statsd metrics service for health and capacity monitoring.
Runtime Configuration Updates

- Minimize disruption to running services.
- Detect errors early and “rollback”.
- Tight integration with DC/OS.
Software and Configuration Updates

Change settings post installation:
- Runtime update of configuration options
- Breakpoints for operator inputs
- Rollback

```
dcos kafka update start --options=config.json
dcos kafka update status
dcos kafka update pause
dcos kafka update resume
```
Summary

- DC/OS presents a great option to run any application “as-a-Service” on any infrastructure.

- The DC/OS SDK allows for technologies to be run as stateful services on DC/OS with reduced operational complexity and increased agility.
Thank You!
Resources

Documentation for data frameworks on DC/OS
https://docs.mesosphere.com/services/

SDK
https://github.com/mesosphere/dcos-commons
https://mesosphere.github.io/dcos-commons/developer-guide/
https://docs.mesosphere.com/services/ops-guide/
Introducing gh-ost: triggerless, painless, trusted online schema migrations

11:20 - 12:10, Matterhorn 2
The “Operator” and “Developer”

The “Operator”

• Operates the platform
  - IaaS, PaaS or XaaS
• Responsible for keeping the lights on and effective utilization of infrastructure

The “Developer”

• And here
  — And maybe even here