Elastic Data Services on Apache Mesos via Mesosphere’s DC/OS

Mohit Soni  
mohit@apache.org

Adam Bordelon  
me@apache.org

Strata NY, Sep 2016
● The Scene: Big Data in the Datacenter
● The Problem: Siloed clusters per-app
● The Solution: a Datacenter Operating System
  ○ The “OS”: Mesos, Marathon, Universe, UI/CLI
  ○ The “Apps”: Data Services, Microservices, Containers
● Demo
● Community of Users, Partners
● Takeaways
HYPERSCALE COMPUTING IS GOING MAINSTREAM

UNIT OF INTERACTION

PARTITION (LPAR)

MAINFRAME

DEFINITIVE APPS AND OS

- DATA / TRANSACTION PROCESSING
- UNIX, IBM OS/360

SERVER

PHYSICAL (x86)

- ERP, CRM, PRODUCTIVITY, MAIL & WEB SERVER
- LINUX, WINDOWS

VIRTUAL MACHINE

VIRTUAL

- ERP, CRM, PRODUCTIVITY, MAIL & WEB SERVER
- HYPervisor + GUEST OS

NEW FORM FACTOR FOR DEVELOPING AND RUNNING APPS

HYPERScale

- BIG DATA, INTERNET OF THINGS, MOBILE APPS
- THE DATACENTER NEEDS AN OPERATING SYSTEM

© 2016 Mesosphere, Inc. All Rights Reserved.
HYPERSCALE MEANS: CONTAINERIZATION

<table>
<thead>
<tr>
<th>Virtual Machines</th>
<th>Containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Code</td>
<td>User Code</td>
</tr>
<tr>
<td>Libraries</td>
<td>Libraries</td>
</tr>
<tr>
<td>Operating System</td>
<td>Operating System</td>
</tr>
<tr>
<td>Virtual Processor</td>
<td>Virtual Processor</td>
</tr>
<tr>
<td>Physical Processor</td>
<td>Physical Processor</td>
</tr>
</tbody>
</table>

- **Start time**: 30-45 seconds vs. < 50 ms
- **Stop time**: 5-10 seconds vs. < 50 ms
- **Workload density**: 1x vs. 10 - 100x
HYPERSCALE MEANS: MICROSERVICES ARCHITECTURE

Traditional Architecture

- Siloed teams
- Many functions in a single process
- Small number of large processes with strong inter-dependencies
- Scales monolithically

Microservices Architecture

- Cross-functional teams organized around capabilities
- Each element of functionality defined as "microservices"
- Scales individually
- Cross-functional teams creating new microservices without interdependencies
HYPERSCALE MEANS VOLUME AND VELOCITY

Days  Hours  Minutes  Seconds  Microseconds

Batch  Micro-Batch  Event Processing

Reports what has happened using descriptive analytics
Solves problems using predictive and prescriptive analytics

Billing, Chargeback  Product recommendations
Real-time Pricing and Routing  Real-time Advertising
Predictive User Interface

Solves problems using predictive and prescriptive analytics
RUNNING DATACENTER SERVICES

Traditional Approach

- Static partitioning
- Weeks to provision, manual operations
- Onboarding new technologies is difficult

Mesosphere DC/OS Approach

- Faster provisioning and simplified operations
- Easier to onboard new technologies (e.g., Kafka, Spark, Cassandra, etc)
SILOS OF DATA, SERVICES, USERS, ENVIRONMENTS

Typical Datacenter
siloed, over-provisioned servers, low utilization

DC/OS Datacenter
automated schedulers, workload multiplexing onto the same machines

Industry Average
12-15% utilization

DC/OS Multiplexing
30-40% utilization, up to 96% at some customers
HYPERSCALE CHALLENGES

- Workload variability
- Efficiency
- Interoperability
- Flexibility
- Scalability
- High Availability
- Operability
- Portability

- Isolability
- Schedulability
- Shareability
- Extensibility
- Programmability
- Monitorability
- Debuggability
- Usability
DC/OS: THE DATACENTER OPERATING SYSTEM

1. Scalable, resilient, battle-tested “kernel” for the DC/OS
2. Broadest workload coverage for containers and stateful data services
3. Broad ecosystem of partner services
4. Datacenter-level ops interface that is easy to use. Built by operators for operators

Any Server Infrastructure (Physical, Virtual, Cloud)
DC/OS (~30 OSS components)

- UI and CLI, Cluster Installer/Bootstrapper
- **Resource Management**
- **Container Orchestration: Services** & Jobs
- **Services Catalog**, Package Management
- Virtual Networking, Load Balancing, DNS
- Logging, Monitoring, Debugging

**ENTERPRISE DC/OS**

- TLS Encryption
- Identity & Access Management
- Secrets Management
- Enterprise-grade Support
# DATACENTER RESOURCE MANAGEMENT

## Production-proven Web-Scale Cluster Resource Managers

<table>
<thead>
<tr>
<th>Resource Manager</th>
<th>Year</th>
<th>License</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Borg/Omega</td>
<td>~2001</td>
<td>Proprietary</td>
<td>Production-proven at scale on 10Ks hosts @ Twitter</td>
</tr>
<tr>
<td>Tupperware/Bistro</td>
<td>~2007</td>
<td>Proprietary</td>
<td>Built in collaboration with Google to overcome some Borg Challenges</td>
</tr>
<tr>
<td>Apache Mesos</td>
<td>2010+</td>
<td>Open Source (Apache License)</td>
<td>Production proven at scale on 10Ks hosts @ Twitter, Built at UC Berkeley AMPLab by Ben Hindman (Mesosphere Co-founder)</td>
</tr>
</tbody>
</table>

- Built at UC Berkeley AMPLab by **Ben Hindman** (Mesosphere Co-founder)
- Built in collaboration with Google to overcome some Borg Challenges
- Production proven at scale on 10Ks hosts @ Twitter
What is Mesos? A distributed systems kernel

Mesos is built using the same principles as the Linux kernel, only at a different level of abstraction. The Mesos kernel runs on every machine and provides applications (e.g., Hadoop, Spark, Kafka, Elasticsearch) with APIs for resource management and scheduling across entire datacenter and cloud environments.

**LINEAR SCALABILITY**
Industry proven to easily scale to 10,000s of nodes.

**HIGH AVAILABILITY**

**CONTAINERS**
Native support for launching containers with Docker and AppC images.

**PLUGGABLE ISOLATION**
First class isolation support for CPU, memory, disk, ports, GPU, and modules for custom resource isolation.

**TWO LEVEL SCHEDULING**
Support for running cloud native and legacy applications in the same cluster with pluggable scheduling policies.

**APIS**
HTTP APIs for developing new distributed applications, for operating the cluster, and for monitoring.

**WEB UI**
Built-in Web UI for viewing cluster state and navigating container sandboxes.

**CROSS PLATFORM**
Runs on Linux, OSX and Windows. Cloud provider agnostic.
MESOS ARCHITECTURE

LEADER

MESOS MASTER QUORUM

ZK

Agent 1

Agent N

Marathon
Scheduler

Myriad
Scheduler

Myriad
Executor
Task

Myriad
Executor
Task

Myriad
Executor
Task

Marathon
Executor
Task

Marathon
Executor
Task

Marathon
Executor
Task

ZK

ZK

ZK
Marathon is a DC/OS service for long-running services such as:
- web services
- application servers
- databases
- API servers

Services can be Docker images or JARs/tarballs plus a command

Marathon is not a Platform as a Service (PaaS), but a powerful RESTful API that can be used for building your own PaaS

https://mesosphere.github.io/marathon/docs/generated/api.html
## DATA PROCESSING AT HYPERSCALE - MESOSPHERE INFINITY

<table>
<thead>
<tr>
<th>EVENTS</th>
<th>INGEST</th>
<th>STORE</th>
<th>ANALYZE</th>
<th>ACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubiquitous data streams from connected devices</td>
<td>Ingest millions of events per second</td>
<td>Distributed &amp; highly scalable database</td>
<td>Real-time and batch process data</td>
<td>Visualize data and build data driven applications</td>
</tr>
</tbody>
</table>

### Tools

- **Apache Kafka**
- **Apache Cassandra**
- **Apache Spark**
- **Akka**

---

© 2016 Mesosphere, Inc. All Rights Reserved.
INFINITY USE CASES

IOT APPLICATIONS: Harness the power of connected devices and sensors to create groundbreaking new products, disrupt existing business models, or optimize your supply chain.

ANOMALY DETECTION: Detect in real-time problems such as financial fraud, structural defects, potential medical conditions, and other anomalies.

PREDICTIVE ANALYTICS: Manage risk and capture new business opportunities with real-time analytics and probabilistic forecasting of customers, products and partners.

PERSONALIZATION: Deliver a unique experience in real-time that is relevant and engaging based on a deep understanding of the customer and current context.
Community Frameworks:

APACHE MYRIAD (incubating)
“PRODUCTION GRADE” DC/OS SERVICE

Composed of:
● Permanent Tasks
● Transient Tasks

Goals of service:
● Deployment and maintenance of tasks
● Provide fault-tolerance
● Prevent leakage of resources via strict accounting
BUILT-IN FAULT-TOLERANCE

- Reliable data recovery
  - Reserved resources
  - Persistent volumes
- Minimize re-replication
  - Transient failures (like network partitions) shouldn’t lead to re-replication of data
SERVICE OPERATIONS

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

- Human friendly way of thinking
- Debuggable by design
- Monitor progress
- Fault-tolerant
FAULT-TOLERANCE
FAULT-TOLERANCE

- Persist target
- Reconstruct current state
- Generate plan
• DC/OS Service for Apache Kafka
• DC/OS Service for Apache Cassandra
• Apache Myriad (incubating) for Apache Hadoop/YARN

All of above running on a single DC/OS cluster powered by Apache Mesos.
DEMO - SUMMARY

- Easy install of new Data Services
- Fault tolerant to crashes
- Re-configuration, horizontal scaling
- Generally applicable to services
  - Heterogeneous (HDFS, Myriad)
  - Uniform but stateful (Kafka, Cassandra)
  - Stateless
Forging Ahead with Mesos, Containers and DC/OS

Having now run our event streaming and big data ingestion pipeline services in production on DC/OS, across 3 regions, over the last year, we've achieved the following results:

- A **66%** reduction in AWS Instances
- Cost Improvements up to **57%**
- An impressive **40 sec** time to deploy a new build with zero downtime
- A **3 min** time to stand up a new region
- **100% Uptime**
- Total Resources needed: **1 DevOps Engineer**

http://cloudengineering.autodesk.com/blog/2016/04/auto
desk-is-forging-ahead-with-dcos.html
VERIZON SUCCESS STORY

Challenges

- Verizon needed infrastructure that could handle the **volume and speed** of data that users of its **go90 video streaming** generate
- Needed to easily deploy and run **Spark** (data processing engine) and **Kafka** (messaging queue)

DC/OS Solution

- Mesosphere DC/OS allowed Verizon to easily deploy and run Spark and Kafka, for a **recommendation engine** and **real-time quality of service** to improve user experience
- Chose Mesosphere DC/OS for hybrid cloud capabilities, to **move from AWS to Verizon’s private datacenter**

Larry Rau from @Verizon with @flo
Launching 50,000 containers in seconds with @mesosphere #DCOS
THE COMMUNITY
TAKEAWAYS

- **Elastic**: Scale your cluster and apps, with minimal operational overhead or cluster reaction time
- **Multi-workload**: Hadoop, Spark, Cassandra, Kafka, and arbitrary microservices/containers/scripts
- **Resilient**: Every DC/OS component is replicated and fault-tolerant; SDK makes it easy to build a resilient app scheduler to handle task failures
- **Scalable**: Proven in production on clusters of 10,000s nodes
- **Efficient**: Improve cluster utilization, reduce costs, and increase productivity by letting developers focus on apps, not infrastructure
- **Isolated**: cgroups and namespaces to isolate cpu/gpu, mem, network/ports, disk/filesystem (with/without docker runtime)
Join the DC/OS Community

Connect with our community of users and browse the latest DC/OS news.

GitHub
Are you interested in helping us make DC/OS even better? Let’s work together! Check out our source code on GitHub.
View repositories →

Slack
Have any questions? Our Slack channel is the best place to get help. Just send us a request to automatically receive your invitation.
Join chat →

Mailing List
Want to stay in the loop and connect with other community members? Our public mailing list has all the latest updates. Join the discussion.
Join users@dcos.io →
RESOURCES

- https://dcos.io
- https://mesos.apache.org/
- https://github.com/mesosphere/dcos-cassandra-service
- https://github.com/mesosphere/dcos-kafka-service
- https://myriad.incubator.apache.org
- https://github.com/mesosphere/dcos-commons
Thank You!