The SMACK Stack on Mesosphere DC/OS

Using Cloud Infrastructure

#OSCON
Kaitlin Carter

➔ Instructor & Content Developer at Mesosphere
➔ Develop Technical Trainings
➔ Instructional Designer
John Dohoney, Jr.

➔ Solution Architect at Mesosphere
➔ 10+ years in Digital Transformation Technologies
➔ 20+ years in Linux systems architecture
Agenda

1. Course Goals and Lab Environment
2. Intro to SMACK Stack
3. Intro to DC/OS
4. Lab 1
5. SMACK Stack Technologies on DC/OS
6. Lab 2
7. Case Study & Demo
8. Lab 3
9. Next Steps
Workshop Goals

Learn and understand:

• How to install, configure, and maintain SMACK Stack technologies on DC/OS.

• Benefits of using SMACK on DC/OS for data pipelines.

Gain hands on experience:

• Installing DC/OS with Ansible.

• Deploying a SMACK Stack.

• Deploying a application that uses the SMACK Stack.
Your lab environment consists of 7 nodes:

- **Bootstrap Node**: DC/OS CLI and Bastion host.
- **Master Node**: Controls the cluster.
- **Public Agent Node**: Facilitates communication from outside the cluster to the services running in the cluster.
- **Private Agent Nodes x4**: The nodes where our deployed services will run.

Lab Instructions:

- https://github.com/mesosphere/oscon-smack-stack
Raffle!

To participate:

- Email us confirming at education@mesosphere.com

Raffle Rules:

- There is a 1st and 2nd place.
- You can only enter once.
- Winners announced at the end of today’s session - must be present.
Raffle

1st Prize:
- Star Wars Legos
- Swag bag

2nd Prize:
- Predator 3 Drone
- Swag bag
Intro to SMACK Stack:

- History of Big Data, Slow Data, and Fast Data
- Motivation & Problems Solved
- Intro to SMACK
Fast Data: Historical Context

**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
Recent Data Architectures

- Architectures affecting Digital Transformation
- Hadoop Map-Reduce
  - Slow Data Pattern
- Lambda Architecture – *SMACK Stack application*
  - Bridge Between
    - Slow Data
    - Fast Data
- FAST Data Architecture – *SMACK Stack application*
What is “Slow Data”

● Slow Data is captured as part of a business process with no intention of its usage, intrinsic value for trends, and in some cases its presence is only a status symbol with no corporate value.
● Can not be enriched, can not be combined, and usually not de-normalized – think about it...
● Lives/Resides in “glaciers”, “lakes”, and “warehouses” and in most case if lost or deleted there is little consequence – perhaps with the exception of compliance retention
● Not capable of streaming – the delta is not that interesting, the rate of change, nor the patterns of change
Hadoop MapReduce

1. Job Submitted
2. Job queries HDFS Name-Node(s) to find data
3. Job Tracker creates execution plan and submits to Task Trackers
4. Task trackers perform task and report status to Job Tracker
5. Job Tracker manages task phases
6. Job Tracker finished task and updates status
• Transitional Architecture in many cases
• Used in an enterprise where Slow and Fast data exist
• SMACK, or “SMACK-Like” Stack used to implement system
Modern Application -> Fast Data Built-in

Use Cases:
- Anomaly detection
- Personalization
- IoT Applications
- Predictive Analytics
The SMACK Stack is based on...

- **Spark** - fast and general engine for distributed, large-scale data processing
- **Mesos** - cluster resource management system that provides efficient resource isolation and sharing across distributed applications
- **Akka** - a toolkit and runtime for building highly concurrent, distributed, and resilient message-driven applications on the JVM
- **Cassandra** - distributed, highly available database designed to handle large amounts of data across multiple datacenters
- **Kafka** - a high-throughput, low-latency distributed messaging system designed for handling real-time data feeds
Why SMACK Stack...

- It is a toolbox for many data processing architectures
- It has been “Battle-Tested” and used in many industry verticals
- Probably the shortest path to Minimum Viable Product (MVP)
- Proven to easily be scalable and highly elastic
- SMACK is a single platform for many kinds of applications
- Is well suited for deployment as a unified cluster management for a diversity of workloads
Success Model

- Shortest path to Minimum Viable Product (MVP)
- Battle-Tested, Scalable and already designed for Cloud Native
In review, the SMACK Stack is ...

**EVENTS**
Ubiquitous data streams from connected devices

**INGEST**
Ingest millions of events per second

**ANALYZE**
Real-time and batch process data

**STORE**
Distributed & highly scalable database

**ACT**
Visualize data and build data driven applications

---

**Sensors**

**Devices**

**Clients**

---

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

---

**Mesos/DC/OS**
Intro to DC/OS:

- Core Concepts
- DC/OS Architecture
  - Containers & Container Orchestration
  - Interacting with DC/OS & the DC/OS Catalog
  - Mesos
Multiplexing of Data, Services, Users, Environments

Typical Datacenter
- siloed, over-provisioned servers
- low utilization

Apache Mesos
- automated schedulers
- workload multiplexing onto the same machines
DC/OS is...

- 100% open source (ASL2.0)
  + A big, diverse community
- An umbrella for ~30 OSS repos
  + Roadmap and designs
  + Documentation and tutorials
- Familiar, with more features
  + Networking, Security, CLI, UI, Service Discovery, Load Balancing, Packages, ...
Quick Knowledge Check

Is the mesos component in DC/OS also the foundational technology in the SMACK stack?
DC/OS Brings it All Together

- Resource management
- Task scheduling
- Container orchestration
- Logging and metrics
- Network management
- “Universe” catalog of pre-configured apps
- And much more [https://dcos.io/]
DC/OS Architecture Overview: DC/OS Components
Containers: Docker

- Rapid deployment
- Some service isolation
- Dependency handling
- Container image repository
Containers: Runtime

Docker Engine
- Docker images only
- Must be installed on all cluster nodes.

UCR
- Docker images
- Mesos containers
- GPU & CNI support
- Installs with DC/OS
Containers Orchestration: Marathon

- Built-in scheduler for long-running services and Mesos frameworks.
  - Starts and keeps applications running.
  - Similar to a distributed init system.
- A Mesos framework is a distributed system that has a scheduler.
- Mesos mechanics are fair and HA.
DC/OS Architecture Overview
Interact with DC/OS: DC/OS UI
Interacting with DC/OS: Installing Catalog Packages

Certified packages are verified by Mesosphere for interoperability with DC/OS.
Interact with DC/OS: DC/OS CLI

DC/OS CLI for Node & Cluster Management.

- dcos config
- dcos node
- dcos cluster

DC/OS CLI for App Management.

- dcos package
- dcos job
- dcos marathon
- dcos task
Interacting with DC/OS: Installing Catalog Packages

```json
{
    "service": {
        "name": "kafka",
        "user": "nobody",
        "virtual_network_enabled": false,
        "virtual_network_name": "dcos",
        "virtual_network_plugin_labels": "",
        "placement_constraint": "[["hostname"], \"MAX_PER\", \"1\"]",
        "deploy_strategy": "serial"
    }
}
```
Tour DC/OS & Demo

- DC/OS UI and CLI walk through
  - Nodes page
  - Dashboard
  - Catalog: smack packages and k8s package.
  - Services page: marathon apps
  - Jobs page: metronome
Advanced Installation

1. Prerequisites:
   - Docker
   - OS packages
   - NTP enabled
   - Overlay for Docker
   - DC/OS Package
   - /genconf
     - IP Detect
     - Config file

2. Install Process:
   - Generate installer
   - Serve install files
   - Install master
   - Install agents

   $ sudo bash dcos_install.sh master
Installing DC/OS Lab

Server Assignments:

- https://tinyurl.com/y9uq9pa6

In this lab you will:

- Install a cluster of DC/OS nodes with Ansible.
- Explore the DC/OS UI.
- Install the DC/OS CLI on the bootstrap node.
- Try out the the DC/OS CLI.
DC/OS Architecture Overview
SMACK stack

- History & Context
- Intro to Mesos
- Architecture
SMACK Stack

**EVENTS**
Ubiquitous data streams from connected devices

**INGEST**
Ingest millions of events per second

**ANALYZE**
Real-time and batch process data

**STORE**
Distributed & highly scalable database

**ACT**
Visualize data and build data driven applications

- Sensors
- Devices
- Clients

---

Apache Kafka

Apache Spark

Apache Cassandra

Akka

Mesos/DC/OS
Build Block of Modern Internet

- A cluster resource negotiator
- A top-level Apache project
- Scalable to 10,000s of nodes
- Fault-tolerant, battle-tested
- An SDK for distributed apps
- Native Docker support
Mesos: Datacenter Kernel

- Opens source Apache project.
- Resource manager.
- Pools resources from set of servers to create “one giant computer”.
- Mesos master orchestrates agent tasks.
- Mesos agents provide resources.
### Active Tasks

<table>
<thead>
<tr>
<th>Framework ID</th>
<th>Task ID</th>
<th>Task Name</th>
<th>Role</th>
<th>State</th>
<th>Started</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>62df48e-df8aa-4309-94f0-73d5e94ab01e-0001</td>
<td>bus-demo_dashboard.37943816-8677-11e7-b432-425fcb45b8</td>
<td>dashboard.bus-demo</td>
<td>slave_public</td>
<td>RUNNING</td>
<td>a minute ago</td>
<td>10.0.5.101</td>
</tr>
<tr>
<td>62df48e-df8aa-4309-94f0-73d5e94ab01e-0001</td>
<td>bus-demo_ingest.0999da65-8676-11e7-b432-425fcb45b8</td>
<td>ingest.bus-demo</td>
<td>slave_public</td>
<td>RUNNING</td>
<td>9 minutes ago</td>
<td>10.0.1.204</td>
</tr>
<tr>
<td>62df48e-df8aa-4309-94f0-73d5e94ab01e-0004</td>
<td>broker-2__581647a0-6953-4cfe-af96-356d04535c38</td>
<td>broker-2</td>
<td>kafka-role</td>
<td>RUNNING</td>
<td>12 minutes ago</td>
<td>10.0.3.240</td>
</tr>
<tr>
<td>62df48e-df8aa-4309-94f0-73d5e94ab01e-0004</td>
<td>broker-1__d24b1885-860b-4ae9-9feb-502ffced5fe</td>
<td>broker-1</td>
<td>kafka-role</td>
<td>RUNNING</td>
<td>13 minutes ago</td>
<td>10.0.3.7</td>
</tr>
<tr>
<td>62df48e-df8aa-4309-94f0-73d5e94ab01e-0004</td>
<td>broker_0__eb077cd0-f416-4918-9c53-1f5b1ea8c14d</td>
<td>broker-0</td>
<td>kafka-role</td>
<td>RUNNING</td>
<td>13 minutes ago</td>
<td>10.0.1.204</td>
</tr>
<tr>
<td>62df48e-df8aa-4309-94f0-73d5e94ab01e-0001</td>
<td>kafka.5a668774-8675-11e7-b432-425fcb45b8</td>
<td>kafka</td>
<td>slave_public</td>
<td>RUNNING</td>
<td>13 minutes ago</td>
<td>10.0.0.56</td>
</tr>
<tr>
<td>62df48e-df8aa-4309-94f0-73d5e94ab01e-0003</td>
<td>node-2__a9c29921-d7e1-4a32-8eb5-4fd37b25065d</td>
<td>node-2</td>
<td>cassandra-role</td>
<td>RUNNING</td>
<td>14 minutes ago</td>
<td>10.0.3.7</td>
</tr>
<tr>
<td>62df48e-df8aa-4309-94f0-73d5e94ab01e-0003</td>
<td>node-2__a9c29921-d7e1-4a32-8eb5-4fd37b25065d</td>
<td>node-2</td>
<td>cassandra-role</td>
<td>RUNNING</td>
<td>14 minutes ago</td>
<td>10.0.3.7</td>
</tr>
</tbody>
</table>
Mesos Architecture

Two-level Scheduling

1. Agents advertise resources to Master
2. Master offers resources to Framework
3. Framework rejects or uses resources
4. Agent reports task status to Master
Hey Master, I have 4 CPUs, 4 GB of RAM, and 100 GB of disk space available.

Great, I’ll make a note of it!
Mesos in Action - User Request

Hey Marathon, I need an nginx container that needs 1 CPU and 1 GB of RAM

Great, I’ll ask the Master

Mesos Master

Mesos Private Agent
Hey Mesos Master, I need an agent that has 1 available CPU and 1 GB of RAM available.

Sounds good, here are agents that are capable of fulfilling those requirements.

Marathon Scheduler

Mesos in Action - Scheduler Request

Mesos Master

Mesos Private Agent
Great, I'm on it!

Agent, you've been selected to spawn an nginx container that is allocated 1 CPU and 1 GB of RAM - here's all the information I received from the scheduler needed to launch this application.

Mesos in Action - Container Launch
Mesos in Action - Container Running

Mesos Private Agent

Docker Executor

nginx

Hey Marathon, that nginx container you asked for is up and running

Mesos Master

Hey Master, I got that container you were asking for up and running

OK great, I will let the scheduler know

OK great, I will let the end users know
Quick Knowledge Check

How many leading Mesos masters can you have in a DC/OS cluster?

- 1
- 3
- 5
SMACK stack

- Context
- Intro to Spark
- Installing, Configuring, & Managing
SMACK Stack

EVENTS
Ubiquitous data streams from connected devices

INGEST
Ingest millions of events per second

ANALYZE
Real-time and batch process data

STORE
Distributed & highly scalable database

ACT
Visualize data and build data driven applications

Apache Kafka
Apache Spark
Apache Cassandra
Akka

Sensors
Devices
Clients

Mesos/ DC/OS
Streaming Analytics

Micro-batching

- Apache Spark (Streaming)

Native Streaming

- Apache Flink
- Apache Storm/Heron
- Apache Apex
- Apache Samza
Spark: Streaming Analytics

Typical Use: distributed, large-scale data processing; micro-batching

Why Spark Streaming?

- Micro-batching creates very low latency, which can be faster
- Well defined role means it fits in well with other pieces of the pipeline
Spark: Architecture

- Spark SQL
- Spark Streaming
- MLib (machine learning)
- GraphX (graph processing)
- Spark core (RDD)
- Mesos
- Standalone
- YARN

Filesystem (local, HDFS, S3) or data store (HBase, Cassandra, Elasticsearch, etc.)
## DC/OS Spark Package

<table>
<thead>
<tr>
<th>Service</th>
<th>Spark Package Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
<td>DC/OS Spark configuration properties</td>
</tr>
<tr>
<td>Hdfs</td>
<td></td>
</tr>
</tbody>
</table>

**Service**

name

**spark**

**cpus**

1

**mem**

1024

**role**

*
DC/OS Spark Package Parameters

Service

- Name
- CPU
- Mem
- User
- Role for Spark Dispatcher
- “Quota” parameter - restricts resource usage.

HDFS

- HDFS configuration file location

Security

- Kerberos
- Kerberos configuration
DC/OS Spark Package Default Parameters

Service

- 1 CPU
- 1 GB Memory
- Root user for executor
- Role for Spark Dispatcher is "*

Security

- Kerberos is disabled

HDFS

- DC/OS HDFS default configuration
Interacting with Spark

Spark UI

- Monitor Jobs

DC/OS CLI Subcommands

- Submit & Monitor jobs

DC/OS CLI

- dcos task exec -it

Connection Information from UI

- Dispatcher and dispatcher proxy LB info.
SMACK stack

- Intro to Akka
- Configuring
SMACK Stack

- **EVENTS**: Ubiquitous data streams from connected devices
- **INGEST**: Ingest millions of events per second
- **ANALYZE**: Real-time and batch process data
- **STORE**: Distributed & highly scalable database
- **ACT**: Visualize data and build data driven applications

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

**Supporting Technologies**
- **Mesos/DC/OS**
- **Sensors**
- **Devices**
- **Clients**
Akka is a toolkit for building highly concurrent, distributed, and resilient message-driven applications for Java and Scala.

- Simple
- Highly Performant
- Elastic
- Reactive
SMACK stack

- History & Context
- Intro to Cassandra
- Installing, Configuring, & Managing
SMACK Stack

**EVENTS**
Ubiquitous data streams from connected devices

**INGEST**
Ingest millions of events per second

**ANALYZE**
Real-time and batch process data

**STORE**
Distributed & highly scalable database

**ACT**
Visualize data and build data driven applications

---

**Sensors**

**Devices**

**Clients**

---

Apache Kafka
Apache Spark
Apache Cassandra
Akka

---

Mesos/DC/OS
History of Distributed Storage

NoSQL
- ArangoDB
- MongoDB
- Apache Cassandra
- Apache HBase

Filesystems
- Quobyte
- HDFS

Time-Series Datastores
- InfluxDB
- OpenTSDB
- KairosDB
- Prometheus

SQL
- MemSQL
Cassandra

Typical Use: No-dependency, time series database

Why Cassandra?

- A top level Apache project born at Facebook and built on Amazon’s Dynamo and Google’s BigTable
- Offers continuous availability, linear scale performance, operational simplicity and easy data distribution
Cassandra Architecture

- Cassandra is eventually consistent
- Multiple parameter to tweak read/write consistency
  - Write Strategies:
    - Any, One, Quorum, All, ..
  - Read Strategies:
    - One, Quorum, ALL
- Granularity: single row/key
DC/OS Package Definition

Service

Nodes

Cassandra

Service

DC/OS Apache Cassandra service configuration properties

name *

```
cassandra
```

user *

```
nobody
```

service account
DC/OS Cassandra Package Parameters

Service
- Cluster name
- Data Center
- Region

Nodes
- Number of nodes
- Placement constraints
- Racks
- Resources*

Cassandra:
- Practitioner
- Hinted handoff
- Concurrent reads and writes
- tombstone*
DC/OS Cassandra Package Default Parameters

Node
- 3 nodes
- Placement constraint: 1 Cassandra node per DC/OS private agent.
- .5 CPU
- 10 GB Diskspace
- 4 GB RAM

Cassandra
- Hinted handoff enabled
- Partitioner is Murmur3partitioner
- Concurrent Reads 16
- Concurrent Writes 32
Interacting with Cassandra

Connection information from UI or CLI

- Node address and port
- DNS for service

DC/OS CLI: dcos task exec

- Connect to a task

Cqlsh

- Connect to the cluster data store.

Backup & Restore with DC/OS CLI

- Backup to AWS or Azure
- Restore

API

- Replace a node
- Restart a node
- Pause a node
SMACK stack

- Messaging Queues
- Intro to Kafka
- Installing, Configuring, & Managing
SMACK Stack

EVENTS
Ubiquitous data streams from connected devices

INGEST
Ingest millions of events per second

ANALYZE
Real-time and batch process data

STORE
Distributed & highly scalable database

ACT
Visualize data and build data driven applications

Apache Kafka
Apache Spark
Apache Cassandra
Akka

Mesos/ DC/OS
Messaging Queues

Message Brokers

- Apache Kafka
- ØMQ, RabbitMQ, Disque

Log-based Queues

- fluentd, Logstash, Flume

see also queues.io
Kafka

Typical Use: A reliable buffer for stream processing

Why Kafka?

- High-throughput, distributed, persistent publish-subscribe messaging system
- Created by LinkedIn; used in production by 100+ web-scale companies [1]
Kafka: Delivery Guarantees

- At most once—Messages may be lost but are never re-delivered
- At least once—Messages are never lost but may be re-delivered (Kafka)
- Exactly once—Messages are delivered once and only once (this is what everyone actually wants, but it’s tricky)

Murphy’s Law of Distributed Systems:

*Anything that can go wrong, will go wrong ... partially!*
DC/OS Kafka Package

Service

Brokers

Kafka

Service

DC/OS service configuration properties

name

kafka

user

nobody

service account
DC/OS Kafka Package Parameters

**Service**
- Service name
- Placement contraints
- Region
- Deploy strategy

**Kafka**
- Topic management
- Logging

**Brokers**
- Resources*
- Number of brokers
DC/OS Kafka Package Defaults

Service
- Service name: Kafka
- Placement constraints: 1 Kafka broker per DC/OS private agent.
- Region: unselected.
- Deploy strategy: Serial

Brokers
- Resources*
- Number of brokers: 3

Kafka
- Topic management*
- Logging*
Interacting with Kafka

Connection information from UI or CLI

- VIP load balancing
- Node address and port
- DNS for service

DC/OS CLI: dcos task exec

- Connect to a task

Kafka API

- Manage nodes
- Manage topics

DC/OS CLI Subcommands

- Manage topics
SMACK Stack Lab 2

In this lab you will use a script to install:

- Spark
- Cassandra
- Kafka
Case Study & Demo:

- Los Angeles Metro
- Final Lab
SMACK Stack Demo: Los Angeles Metro

Available for you to try at: https://github.com/mesosphere/oscon-smack-stack
SMACK Stack Lab 3

In this lab you will:

● Generating data
● Using Akka
● Monitoring the pipeline
Next Steps:

- Community
- Get Help
- Raffle Winners
Community

Join the Community: dcos.io/community

Get Help
- Mailing List
- Slack
- StackOverflow

Get Involved
- JIRA
- GitHub
- Working Groups

Get Updates
- Twitter @dcos
- YouTube
- Meetup
Self-Service: Documentation

DC/OS Documentation: [https://docs.mesosphere.com](https://docs.mesosphere.com)
- Versioned
- Release Notes
- Component

Service Docs: [https://docs.mesosphere.com/service-docs/](https://docs.mesosphere.com/service-docs/)
- Specific to Certified Packages
- Versioned
- Release Notes
Raffle!
Questions?

@dcos
chat.dcos.io
users@dcos.io
/dcos
/dcos/examples
/dcos/demos