Using DC/OS for Continuous Delivery

DevPulseCon 2017

Elizabeth K. Joseph, @pleia2
Mesosphere
Elizabeth K. Joseph, Developer Advocate, Mesosphere

- 15+ years working in open source communities
- 10+ years in Linux systems administration and engineering roles
- 4 years working on CI/CD for the OpenStack project
- Founder of OpenSourceInfra.org
- Author of *The Official Ubuntu Book* and *Common OpenStack Deployments*
Apache Mesos: The datacenter kernel

http://mesos.apache.org/
Marathon

- Mesos can’t run applications on its own.
- A Mesos framework is a distributed system that has a scheduler.
- Schedulers like Marathon start and keep your applications running. A bit like a distributed init system.
- Mesos mechanics are fair and HA
- Learn more at https://mesosphere.github.io/marathon/
Introducing DC/OS

Solving common problems, including:

- Container orchestration
- Resource and network management
- Task scheduling
- Unified logging and metrics
- “Universe” of pre-configured services (including Jenkins, Cassandra, Kafka…)

Learn more and contribute at https://dcos.io/
DC/OS Architecture Overview

Services & Containers

- HDFS
- Jenkins
- Marathon
- Cassandra
- Flink
- Spark
- Docker
- Kafka
- MongoDB
- +30 more...

DC/OS

- Container Orchestration
- Security & Governance
- Monitoring & Operations
- User Interface & Command Line

ANY INFRASTRUCTURE

- Physical Servers
- Virtual Servers
- Private Cloud
- Public Cloud Providers (Google, AWS, Azure)
Interact with DC/OS (1/2)

Web-based GUI

https://dcos.io/docs/latest/usage/webinterface/
Interact with DC/OS (2/2)

CLI tool
https://dcos.io/docs/latest/usage/cli/

API
https://dcos.io/docs/latest/api/
The Mesos plugin for Jenkins allows Jenkins to dynamically launch Jenkins slaves on a Mesos cluster.

“Put simply, whenever the Jenkins Build Queue starts getting bigger, this plugin automatically spins up additional Jenkins slave(s) on Mesos so that jobs can be immediately scheduled! Similarly, when a Jenkins slave is idle for a long time it is automatically shut down.”

Source: https://github.com/jenkinsci/mesos-plugin
1. Spin up a new Jenkins agent using the Mesos plugin. This agent runs inside a Docker container on one of our DC/OS agents.
2. Clone the git repository
3. Build a Docker container based off the Jekyll Docker image that includes the content stored in /site and push it to DockerHub.
4. Run the newly created container and a Linkchecker container that runs a basic integration test against the container, checking that the web server comes up correctly and that all links being served are valid (i.e. no 404s).
5. Manually trigger a Marathon deployment of the newly created container to the DC/OS base Marathon instance. If the application already exists, Marathon will simply upgrade it.
6. Make the application available on a public agent at port 80 using Marathon-lb.
Demo: Spin up 50 jobs!

Creates 50 build jobs that take a random amount of time between 1 and 2 minutes. These jobs will randomly fail.

- The Mesos plugin will spin up build agents on demand for these jobs, using as much capacity as your cluster has available.
- When these jobs are finished, the Jenkins tasks will terminate and the resources will be relinquished back to other users of your cluster.
DC/OS

1.9 Features
DC/OS 1.9 - Data Services Ecosystem

DATA SERVICES ECOSYSTEM

- Alluxio
- Couchbase
- Datastax DSE
- Elastic (ELK)
- Redis
- Apache Flink

OPERATIONS

WORKLOADS
DC/OS 1.9 - Operations

DATA SERVICES ECOSYSTEM

OPERATIONS

WORKLOADS

- Remote Container Shell
- Unified Metrics
- Unified Logging
- Deployment Failure Debugging
- Upgrades & Configuration updates
REMOTE CONTAINER SHELL

- Open encrypted, interactive, remote session to your containers
- Remotely execute commands for real time app troubleshooting
- Provide developers access to their own applications, not the entire host or cluster
Access application, DC/OS and OS logs

Easily troubleshoot applications with critical metadata such as container id and app id

Integrate easily with existing logging systems
UNIFIED METRICS

- Single API for system, container and application metrics
- Metadata such as host id and container id are automatically added to assist in debugging
- Integrate easily with existing metrics systems
DEPLOYMENT FAILURE DEBUGGING

- Understand why your application is not deploying
- Understand which nodes in the cluster can accommodate the role, constraints, cpu, mem, disk and port requirements for your app
UPGRADES AND CONFIG UPDATES

- Generate new config for cluster nodes
  
  $ dcos_generate_config.sh --generate-node-upgrade-script <installed_cluster_version>

- Single command upgrade script for individual nodes
  
  $ curl -O <Node upgrade script URL>
  $ sudo bash ./dcos_node_upgrade.sh
DC/OS 1.9 - Workloads

DATA SERVICES ECOSYSTEM

OPERATIONS

WORKLOADS

- Pods
- GPU based scheduling
PODS

- Schedule, deploy and scale multiple containers on the same host(s) while sharing IP address and storage volumes
- All containers in a pod instance run as if they are running on a single host in pre-container world
- Useful for migrating legacy applications or building advanced micro services (side car containers)
• Traditional monolithic apps on VMs usually have support services such as log shipper, message queuing clients

• Many support services assume colocation on same host, and local-host access to networking and storage

• Pods simplify moving legacy monolithic apps to containers, reducing risk and accelerating migrations
● Advanced Micro Services patterns require colocating containers together

● Support services include for example:
  ○ Logging or monitoring agents,
  ○ Backup tooling & Proxies
  ○ Data change watchers & Event publishers

● Pods simplify the building and maintenance of complex such microservices
GPU: WHY GPU?

- GPUs are needed for many machine learning and deep learning applications.
- GPUs are essential for real-time or near real time machine learning models.
- GPUs deliver from 10X to 100X performance for some applications, resulting lower $$$/IOPS and more productivity to data science teams.
- GPU applications include real time fraud detection, genome sequencing, cohort analysis and many others.

**GPU ACCELERATION**

*Training A Deep, Convolutional Neural Network*

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>Training Time CPU</th>
<th>Training Time GPU</th>
<th>GPU Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 images</td>
<td>64 s</td>
<td>7.5 s</td>
<td>8.5X</td>
</tr>
<tr>
<td>128 images</td>
<td>124 s</td>
<td>14.5 s</td>
<td>8.5X</td>
</tr>
<tr>
<td>256 images</td>
<td>257 s</td>
<td>28.5 s</td>
<td>9.0X</td>
</tr>
</tbody>
</table>

- {LSTM/WE22 winning model: "Supervision"}
- 7 layers
- 5 convolutional layers < 2 fully-connected
- ReLU, pooling, dropout, response normalization
- Implemented with Caffe

- Dual 10-core Ivy Bridge CPUs
- 1 Tesla K60 GPU
- CPU times utilized Intel MKL, BLAS library
- GPU acceleration from CLEDA metrix libraries (uBLAS)
GPU BASED SCHEDULING

- Test Locally with Nvidia-Docker, deploy to production with DC/OS
- Isolate GPU instances and schedule workloads just like CPU and memory, guaranteeing performance
- Efficiently Share GPU resources across data science team
- Simplify migrating machine learning models across from dev to production, and across clouds
OTHER IMPROVEMENTS

- Mesos 1.2
- Marathon 1.4
- Docker 1.12 and 1.13 (17.03-ce) support
- Centos 7.3 and CoreOS 1235.12.0 support
- Performance improvements across all networking features.
- CNI support for 3rd party CNI plugins.
- 100s of additional bugfixes and tests
Contact and Learn More

Elizabeth K. Joseph
Twitter: @pleia2 & @dcos
Email: ejoseph@dcos.io

Web: https://dcos.io
Source: https://github.com/dcos
Demo: https://github.com/mesosphere/cd-demo