Advanced Continuous Delivery Strategies for Containerized Applications Using DC/OS

Southern California Linux Expo 16x
March 9, 2018

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

- Developer Advocate at Mesosphere
- 4 years working on CI/CD for OpenStack
- 10+ years in Linux systems administration and engineering roles
- Author of The Official Ubuntu Book and Common OpenStack Deployments
Continuous Delivery (CD) is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time.

Modern Release Process

RELEASE 1

Better products through a repeatable release cadence

RELEASE 2

Happier developers through continuous feedback

PROJECT PLANNING

WEEK 1

WEEK 2

WEEK 3

WEEK 4

Better products through a repeatable release cadence

Happier developers through continuous feedback

Customer Feedback

Customer Feedback

Customer Feedback
CD with Containers and DC/OS: 2-pronged approach
Run everything in containers!

CC BY 2.0: https://www.flickr.com/photos/rubbermaid/6909787969/
Organize everything efficiently!
# Traditional Workload Flow Stages

<table>
<thead>
<tr>
<th>Developers</th>
<th>Operator Managed (Shared)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install (Local)</td>
<td>Test</td>
</tr>
<tr>
<td>Development (Local Deploy)</td>
<td>Stage (pre-prod)</td>
</tr>
<tr>
<td></td>
<td>Production</td>
</tr>
</tbody>
</table>
## Modern Workload Flow Stages

<table>
<thead>
<tr>
<th>Developers (Local, Shared)</th>
<th>Operator Managed (Shared)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install (Local)</td>
<td>Stage (pre-prod)</td>
</tr>
<tr>
<td>Development (Local Deploy)</td>
<td>Production</td>
</tr>
<tr>
<td>Test</td>
<td></td>
</tr>
</tbody>
</table>
Supporting various CI/CD pipelines

- Installing each service and maintaining upgrades is time-consuming, with each machine having different OS’s and tooling
  - More difficult because teams like to use many technologies and tools as building blocks
  - Spinning up CD pipeline for each application is time-consuming
- Low utilization driven by silos of developers with single-instances of tools
- Poor allocation of capacity may prevent developers from shipping code, and acquiring new HW is slow
NAIVE APPROACH

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

Industry Average
- 12-15% utilization
THE KERNEL: APACHE MESOS
Use: The primary resource manager and negotiator

Why Mesos?

- 2-level scheduling
- Fault-tolerant, battle-tested
- Scalable to 10,000+ nodes
- Created by Mesosphere founder @ UC Berkeley; used in production by 100+ web-scale companies [1]

DC/OS
DC/OS: Datacenter Operating System

- Resource management
- Task scheduling
- Container orchestration
- Logging and metrics
- Network management
- “Universe” catalog of pre-configured apps (including Jenkins, GitLab, Artifactory…), browse at [http://universe.dcos.io/](http://universe.dcos.io/)
- And much more [https://dcos.io/](https://dcos.io/)
DC/OS Architecture Overview

Services & Containers

- GitLab
- Jenkins
- Marathon
- Cassandra
- Flink
- Spark
- Artifactory
- 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 UI

https://docs.mesosphere.com/latest/gui/
Interact with DC/OS (2/2)

**CLI tool**

https://docs.mesosphere.com/latest/cli/

**API**

https://docs.mesosphere.com/latest/api/
MULTIPLEXING OF DATA, SERVICES, USERS, ENVIRONMENTS

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

Mesos/ DC/OS
- automated schedulers,
- workload multiplexing onto the same machines
## DEPLOYING APPS

<table>
<thead>
<tr>
<th>Manual</th>
<th>Automatic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scheduling</strong></td>
<td>● A sysadmin provisions one or more physical/virtual servers to host the app</td>
</tr>
<tr>
<td></td>
<td>● Mesos resource offers (two-tier scheduling) offers available resources directly to frameworks</td>
</tr>
<tr>
<td><strong>Deployment</strong></td>
<td>● By hand or using Puppet / Chef / Ansible</td>
</tr>
<tr>
<td></td>
<td>● Containers deployed, ideally using a CI/CD tool to create/update app definitions</td>
</tr>
<tr>
<td></td>
<td>● Jenkins SSHing to the machine and running a shell script</td>
</tr>
<tr>
<td></td>
<td>● Docker containers packages app and dependencies</td>
</tr>
<tr>
<td></td>
<td>● Note: all dependencies must also be present!</td>
</tr>
<tr>
<td></td>
<td>● Health checks, restarts unhealthy/failed instances</td>
</tr>
<tr>
<td><strong>Health checks</strong></td>
<td>● Nagios pages a sysadmin</td>
</tr>
<tr>
<td><strong>Service discovery</strong></td>
<td>● Static hostnames / IP addresses in a spreadsheet or config management</td>
</tr>
<tr>
<td></td>
<td>● Provides DNS resolution for running services (hostname / IP address, ports, etc)</td>
</tr>
<tr>
<td></td>
<td>● A sysadmin configures a load balancer manually or with Puppet / Chef / Ansible</td>
</tr>
<tr>
<td></td>
<td>● Load balancer configs built dynamically using cluster state</td>
</tr>
<tr>
<td><strong>Persistence</strong></td>
<td>● Individual servers with RAID 1/5/6/10, expensive SANs, NFS, etc.</td>
</tr>
<tr>
<td></td>
<td>● External/persistent volumes (REX-Ray), HDFS, etc.</td>
</tr>
<tr>
<td></td>
<td>● Dedicated, statically partitioned Ceph or Gluster storage clusters</td>
</tr>
<tr>
<td></td>
<td>● Self-healing Ceph or Gluster on Mesos / DC/OS</td>
</tr>
</tbody>
</table>
Development Team Self-Service for CI/CD

LET DEVELOPERS USE THE TOOLS THEY WANT

- Single-command installation of services like Jenkins, GitLab, and Artifactory
- Once a service is installed, it can be run across the entire datacenter, elastically sharing all or some of the datacenter’s resources
- Ability to run application code (PaaS), containers, and distributed applications with no restrictions to application development teams

DC/OS
Development Team Self-Service for CI/CD

RELIABLE, SIMPLIFIED CI/CD INTEGRATION with DC/OS

Continuous Delivery Pipeline

- Version Control System
- Continuous Integration
- Container Orchestrator
- Production Environment
- Load Balancer

 Artifact Repo & Container Registry

Apache Mesos & DC/OS

git push
Development Team Self-Service for CI/CD

RELIABLE, SIMPLIFIED CI/CD INTEGRATION with DC/OS

Continuous Delivery Pipeline

- GitLab, Bitbucket, GitHub
- Jenkins
- Marathon, Vamp
- DC/OS (Mesos)
- Marathon-lb

Git push

Apache Mesos & DC/OS

© 2018 Mesosphere, Inc. All Rights Reserved.
Old vs. New Deploy Process

Pre-Container

- Build a Debian package
- Push deb pkg to apt server
- Wait for apt server to have deb package ready
- Boot a new AWS instance
- Run puppet on the instance, installing the deb
- Turn the instance into an AMI image
- Boot new AWS instances using new AMI
- Terminate old AWS instances

30+ minutes

“It would easily take 30 minutes for a single deploy even under ideal conditions where nothing broke.”

Container

- Build and push Docker image
- Deploy

<1 minute

“A simple service might only take 20 seconds to fully deploy under ideal conditions.”
Advanced Strategies!
Canary

“Canary release is a technique to reduce the risk of introducing a new software version in production by slowly rolling out the change to a small subset of users before rolling it out to the entire infrastructure and making it available to everybody.”  
https://martinfowler.com/bliki/CanaryRelease.html

Blue/Green

“One of the challenges with automating deployment is the cut-over itself, taking software from the final stage of testing to live production. You usually need to do this quickly in order to minimize downtime. The blue-green deployment approach does this by ensuring you have two production environments, as identical as possible. At any time one of them, let's say blue for the example, is live. As you prepare a new release of your software you do your final stage of testing in the green environment. Once the software is working in the green environment, you switch the router so that all incoming requests go to the green environment - the blue one is now idle.”  
https://martinfowler.com/bliki/BlueGreenDeployment.html
Blue/Green, Canary: Marathon

Marathon

The Marathon scheduler in DC/OS has an API that can be called by Jenkins jobs to specify how a deployment is completed. Since it’s a custom configuration, you can be as specific as you need, but it does make it a more complicated approach.

Get started at
https://mesosphere.github.io/marathon/docs/blue-green-deploy.html

More tips in the Zero Downtime Deployments Lab by
https://github.com/mhausenblas/zdd-lab
Blue/Green, Canary: Vamp

**Vamp**

This is can be simplified by using the open source Vamp tooling. Vamp easily hooks into DC/OS, leveraging your existing Marathon scheduler but with specific definitions around other types of deployments.

Vamp is available in the DC/OS Universe catalog.

Get started at [https://vamp.io/documentation/how-vamp-works/v0.9.5/architecture-and-components/](https://vamp.io/documentation/how-vamp-works/v0.9.5/architecture-and-components/)

Watch in action on DC/OS in “Doing Real DevOps with DC/OS” by Julien Stroheker of Microsoft at MesosCon EU back in October 2017: [https://www.youtube.com/watch?v=hNAWHZhMNf8](https://www.youtube.com/watch?v=hNAWHZhMNf8)
Basic CD pipeline Demo with GitLab & Jenkins
Questions?

Elizabeth K. Joseph
@pleia2
ejoseph@mesosphere.com

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