Continuous Delivery with Containers

Elizabeth K. Joseph
@pleia2
Elizabeth K. Joseph

- Developer Advocate at Mesosphere
- 10+ years in Linux systems administration and engineering roles
- 4 years working on CI/CD for OpenStack
- Author of *The Official Ubuntu Book* and *Common OpenStack Deployments*
Definition: Continuous Delivery

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.

Via https://en.wikipedia.org/wiki/Continuous_delivery
Goal: A Modern Release Process with CD

Better products through a repeatable release cadence

Happier developers through continuous feedback

RELEASE 1  
PROJECT PLANNING

RELEASE 2  
Customer Feedback

RELEASE 3  
Customer Feedback

RELEASE 4
Run everything in containers!

CC BY 2.0: https://www.flickr.com/photos/rubbermaid/6909787969/
Organize everything efficiently!
Utilization

Typical Datacenter
siloed, over-provisioned servers,
low utilization (12-15% bare metal, 30% for VMs)

Containerization Platform
automated schedulers, workload multiplexing
onto the same machines
Supporting various pipelines

Team A
- GitLab
- Jenkins
- TravisCI

Team B
- Artifactory
- Jenkins
- TravisCI

Team C
- GitHub
- Artifactory
- Jenkins

Team D
- GitHub
- Artifactory
- Proprietary Artifact Registry

Team E
- GitLab
- TravisCI
- Proprietary Artifact Registry

Team F
- Artifactory
- TravisCI
Services & Containers

- GitLab
- Jenkins
- Marathon
- Cassandra
- Flink
- Spark
- Artifactory
- Kafka
- MongoDB
- Spinnaker

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)
Docker

Use: Container

Why Docker?
- De facto standard that developers are familiar with
- Portable Dockerfiles for sharing image build source
- Ease of use for building, storing, and deploying containers
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

- Resource management
- Task scheduling
- Container orchestration
- Logging and metrics
- Network management
- “Universe” catalog of pre-configured apps (including Jenkins, GitLab, Artifactory…), browse at https://mesosphere.com/service-catalog
- And much more https://dcos.io/
DC/OS Web-based UI
DC/OS CLI

$ dcos cluster list
$ dcos node
$ dcos package install jenkins
$ dcos node ssh --master-proxy --leader

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

Continuous Delivery Pipeline

GitLab git repository → Jenkins → Marathon scheduler → Marathon-lb

GitLab image registry

Apache Mesos & DC/OS

Marathon-lb serving website via port 80

git push
CI/CD Demo

Using Jenkins, the Jenkins+Mesos plugin, and GitLab to test and deploy an nginx-based website.

https://github.com/dcos/demos/tree/master/cicd
Advanced Strategies!
Canary and Blue/Green Deployments

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
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/

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
Questions?

Elizabeth K. Joseph
lyz@princessleia.com / ejoseph@mesosphere.com
Twitter: @pleia2

Demo: https://github.com/dcos/demos/tree/master/cicd