
Doing CI/CD with Containers

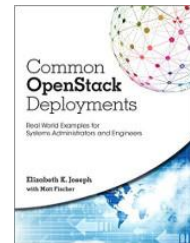
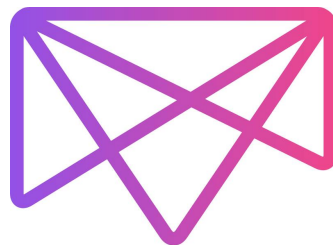
Docker Philadelphia
May 15, 2018

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

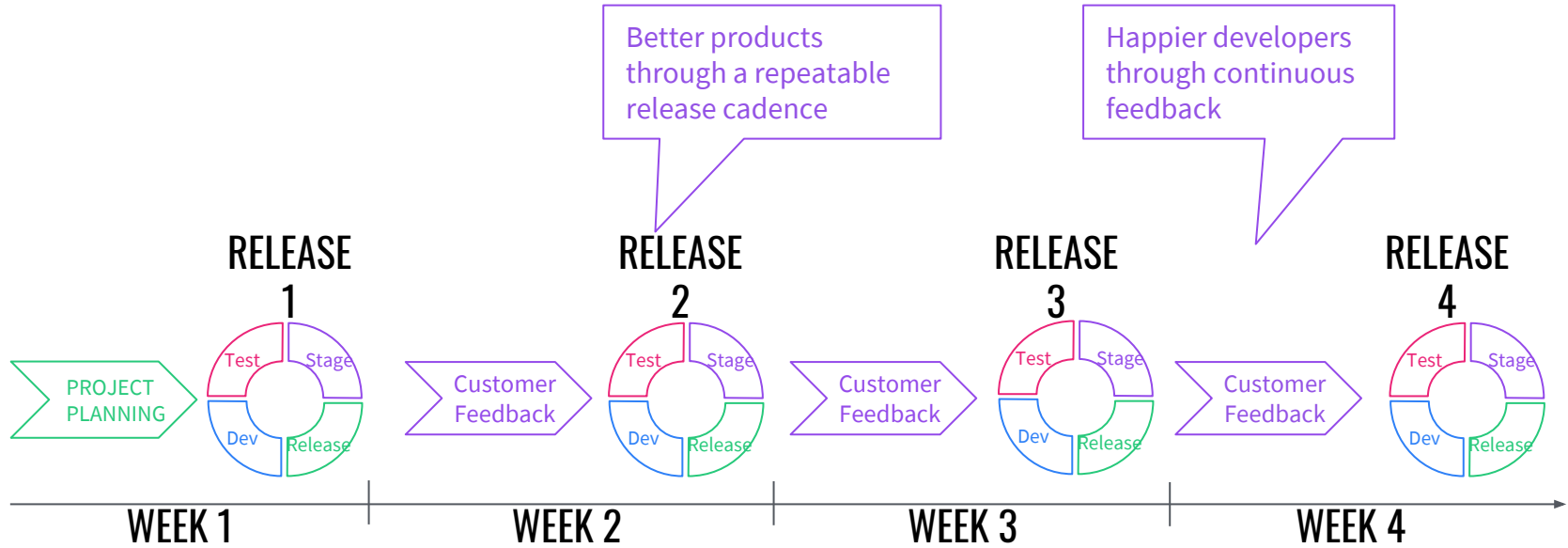


Definition: Continuous Delivery

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

Modern Release Process



CD with Containers and DC/OS: 2-pronged approach

Run everything in containers!



Organize everything efficiently!



Traditional Workload Flow Stages

Developers

Install (Local)

Development
(Local Deploy)

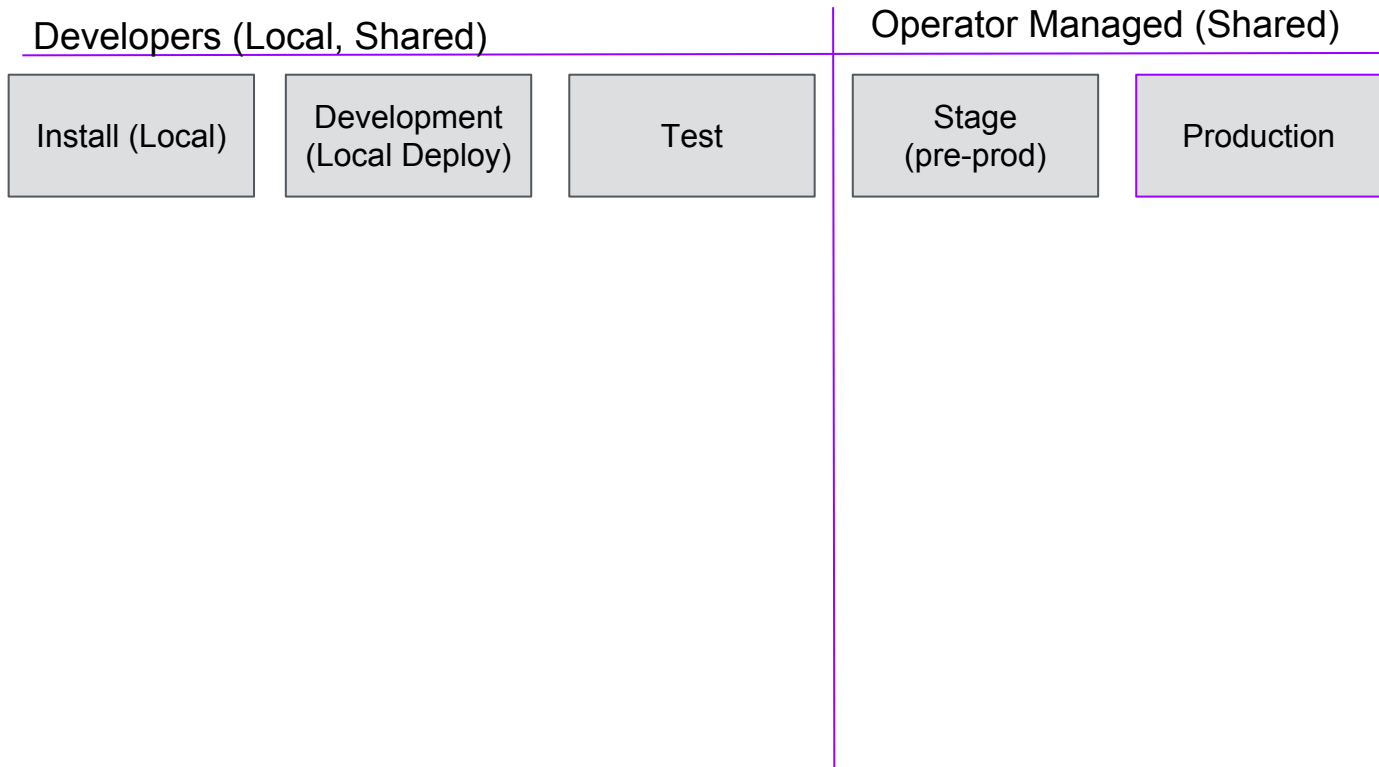
Operator Managed (Shared)

Test

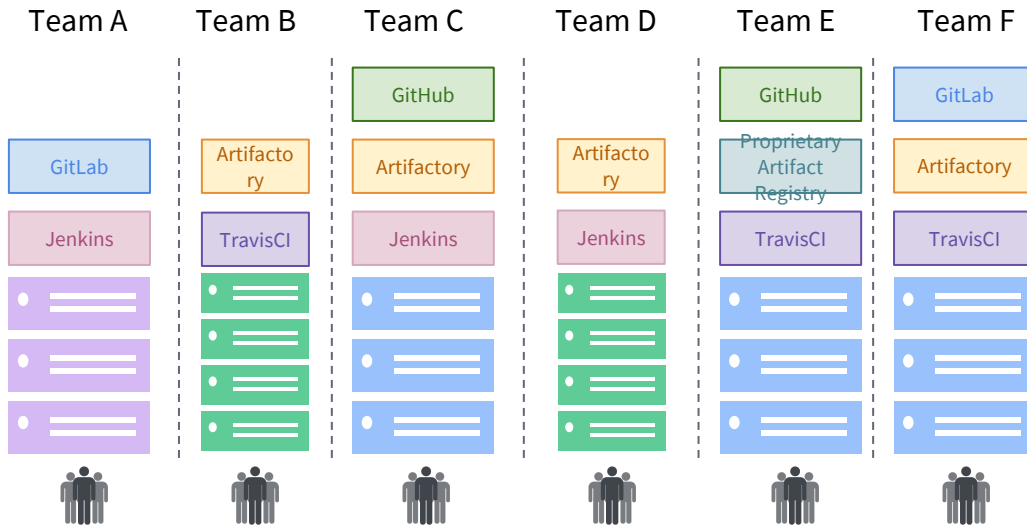
Stage
(pre-prod)

Production

Modern Workload Flow Stages



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

NAIVE APPROACH



**Industry
Average**

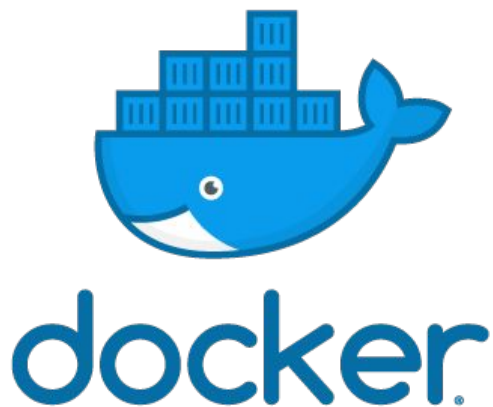
12-15%

utilization siloed, over-provisioned servers,
low utilization

Typical Datacenter

Docker

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

The “kernel”: Apache Mesos

APACHE MESOS



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]

[1]

<http://mesos.apache.org/documentation/latest/powered-by-mesos/>

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/>
- And much more <https://dcos.io/>



DC/OS Architecture Overview

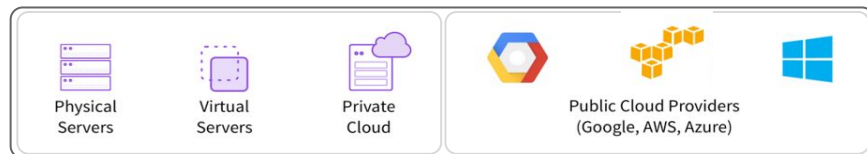
Services & Containers



DC/OS



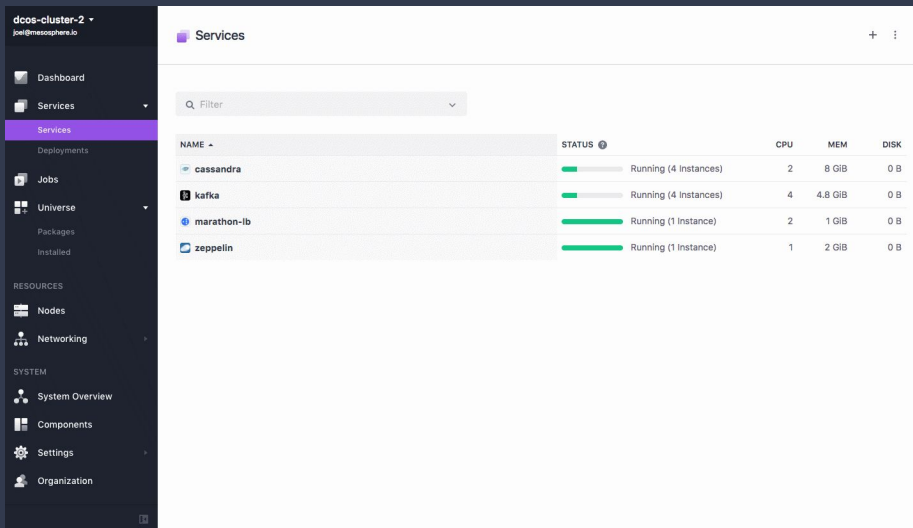
ANY
INFRASTRUCTURE



Interact with DC/OS (1/2)

Web-based UI

[https://docs.mesosphere.com/latest/
gui/](https://docs.mesosphere.com/latest/gui/)



The screenshot displays the DC/OS Services web-based UI. The left sidebar shows the navigation menu with 'Services' selected. The main content area shows a table of running services with columns for NAME, STATUS, CPU, MEM, and DISK.

NAME	STATUS	CPU	MEM	DISK
cassandra	Running (4 Instances)	2	8 GiB	0 B
kafka	Running (4 Instances)	4	4.8 GiB	0 B
marathon-lb	Running (1 Instance)	2	1 GiB	0 B
zeppelin	Running (1 Instance)	1	2 GiB	0 B

Interact with DC/OS (2/2)

CLI tool

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

API

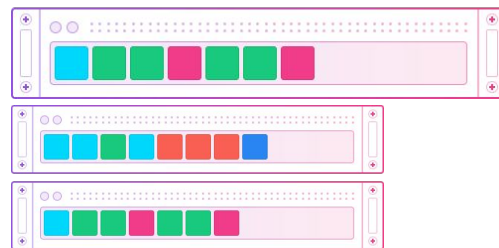
<https://docs.mesosphere.com/latest/api/>

MULTIPLEXING OF DATA, SERVICES, USERS, ENVIRONMENTS



Typical Datacenter

siload, over-provisioned servers,
low utilization



Mesos/ DC/OS

automated schedulers, workload
multiplexing onto the same machines

DEPLOYING APPS

Manual

Scheduling

- A sysadmin provisions one or more physical/virtual servers to host the app

Deployment

- By hand or using Puppet / Chef / Ansible
- Jenkins SSHing to the machine and running a shell script
- Note: all dependencies must also be present!

Health checks

- Nagios pages a sysadmin

Service discovery

- Static hostnames / IP addresses in a spreadsheet or config management
- A sysadmin configures a load balancer manually or with Puppet / Chef / Ansible

Persistence

- Individual servers with RAID 1/5/6/10, expensive SANs, NFS, etc.
- Dedicated, statically partitioned Ceph or Gluster storage clusters

Automatic

- Mesos resource offers (two-tier scheduling) offers available resources directly to frameworks

- Containers deployed, ideally using a CI/CD tool to create/update app definitions
- Docker containers packages app and dependencies

- Health checks, restarts unhealthy/failed instances

- Provides DNS resolution for running services (hostname / IP address, ports, etc)
- Load balancer configs built dynamically using cluster state

- External/persistent volumes (REX-Ray), HDFS, etc.
- Self-healing Ceph or Gluster on Mesos / DC/OS

Development Team Self-Service for CI/CD

LET DEVELOPERS USE THE TOOLS THEY WANT



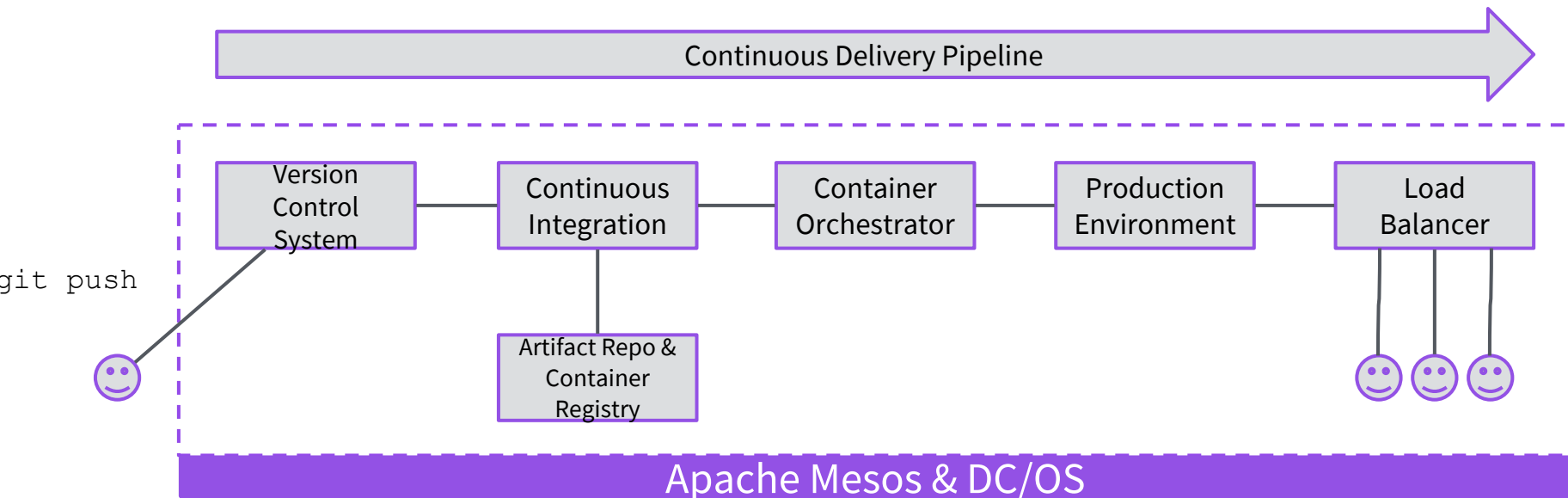
DC/OS

- 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



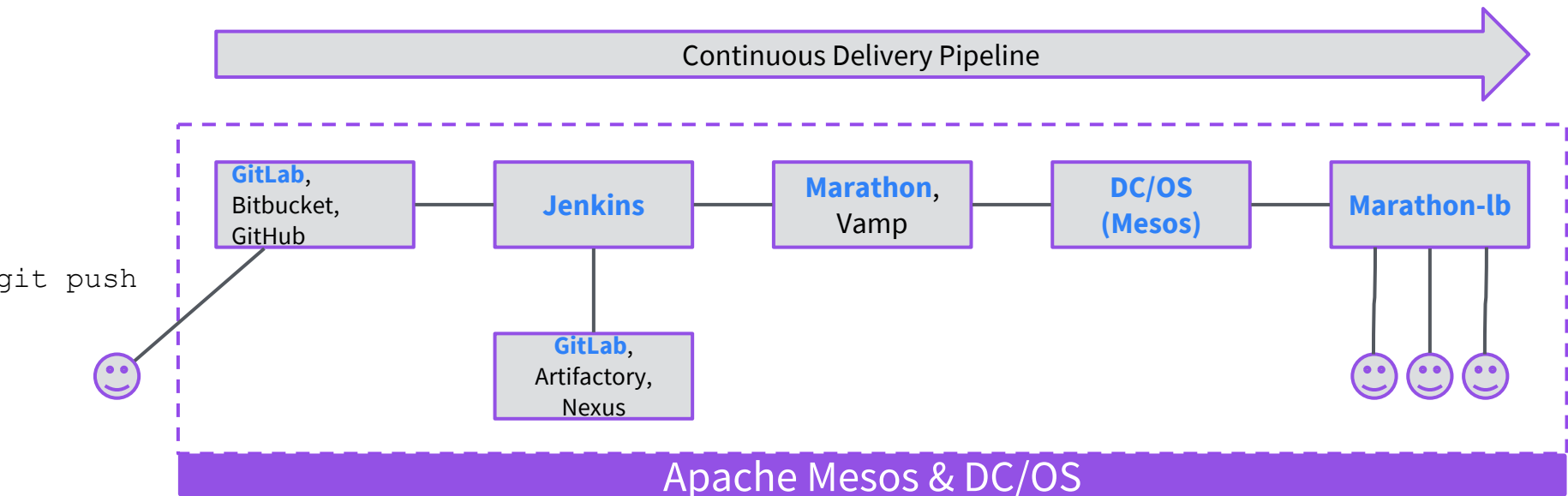
Development Team Self-Service for CI/CD

RELIABLE, SIMPLIFIED CI/CD INTEGRATION with DC/OS

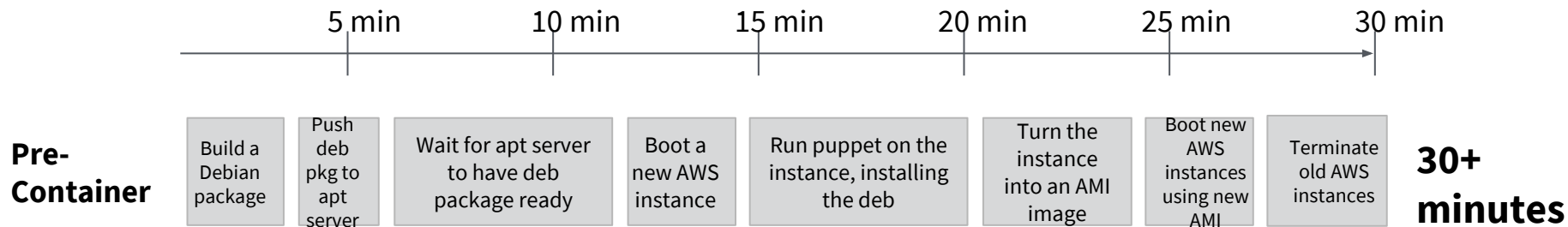


Development Team Self-Service for CI/CD

RELIABLE, SIMPLIFIED CI/CD INTEGRATION with DC/OS

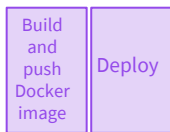


Old vs. New Deploy Process



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

Container



**<1
minute**

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

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>

More tips in the Zero Downtime Deployments Lab by

<https://github.com/mhausenblas/zdd-lab>



Blue/Green, Canary: Vamp

Vamp

This 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>



Basic CD pipeline Demo with GitLab & Jenkins



@dcos



chat.dcos.io



users@dcos.io



/dcos

/dcos/examples

/dcos/demos

Questions?

Elizabeth K. Joseph

@pleia2

ejoseph@mesosphere.com