Advanced Continuous Delivery Strategies for Containerized Applications Using DC/OS

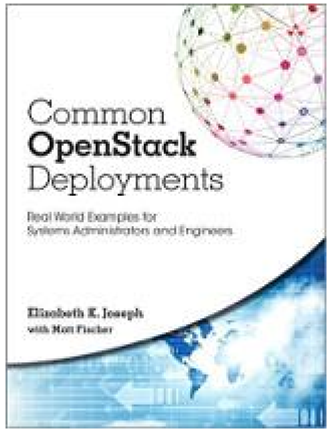
ContainerCon
@ Open Source Summit
North America 2017

Elizabeth K. Joseph
@pleia2

MESOSPHERE

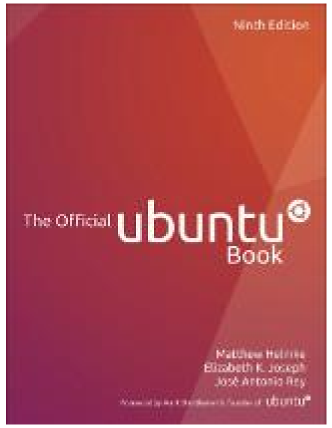
Elizabeth K. Joseph, Developer Advocate

- ❏ Developer Advocate at Mesosphere
- ❏ Spent 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


Common OpenStack Deployments

Real World Examples for Systems Administrators and Engineers

Elizabeth K. Joseph with Matt Fischer


Ninth Edition

The Official ubuntu Book

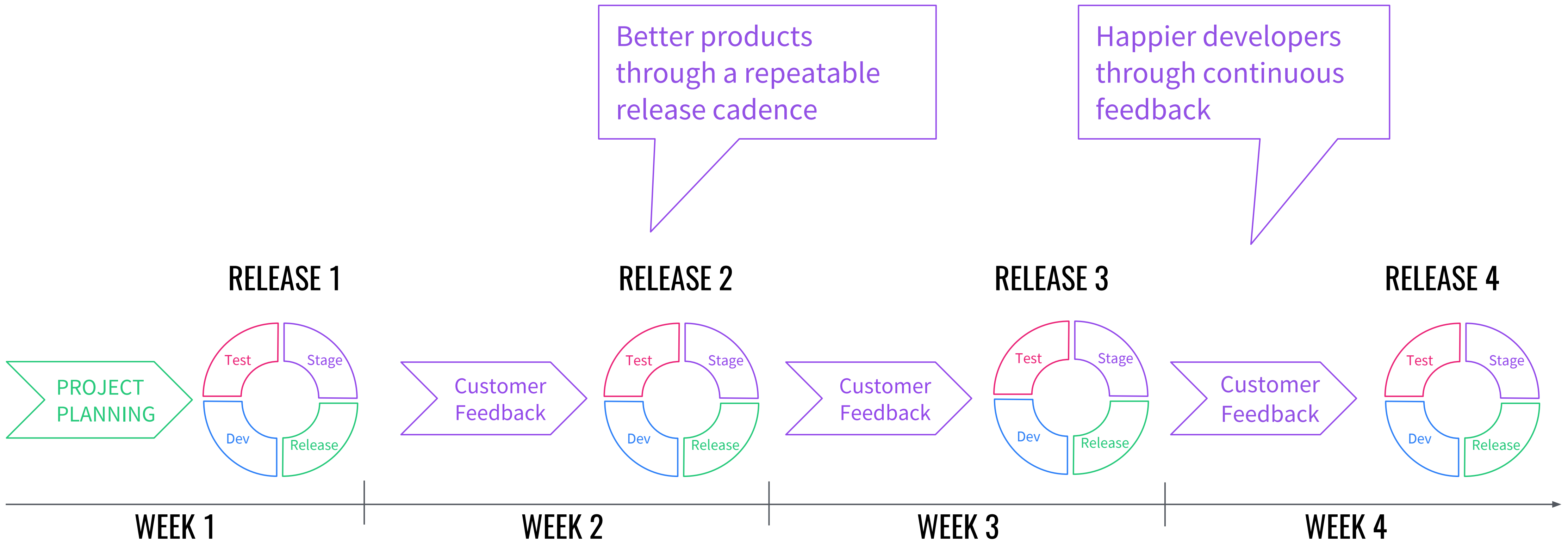
Matthew Helmke
Elizabeth K. Joseph
José Antonio Rey

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: A Key Component of Modern Release Processes

Continuous Delivery - getting workloads READY and RUNNING:

- Perform code analysis, unit tests, and integration tests (continuous integration)
- Dynamically provision environments, configure them, and manage dependencies
- Provision servers (infrastructure automation)
- Deliver and Deploy applications to environments (Dev to Stage to Prod)

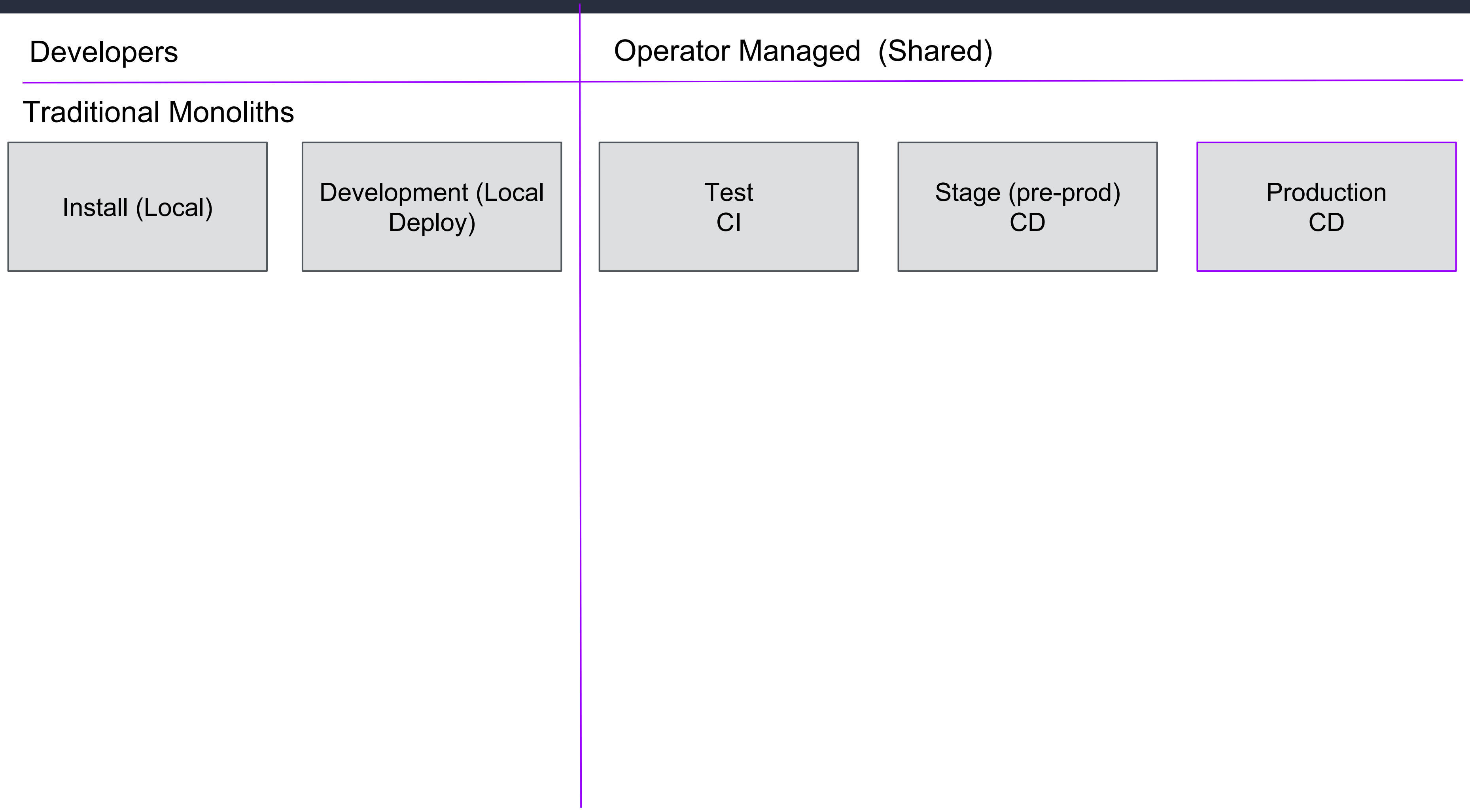
- Low risk releases
- Faster time to market
- Higher quality SW
- Lower costs
- Happier teams

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

Run everything in containers!

Organize everything efficiently!

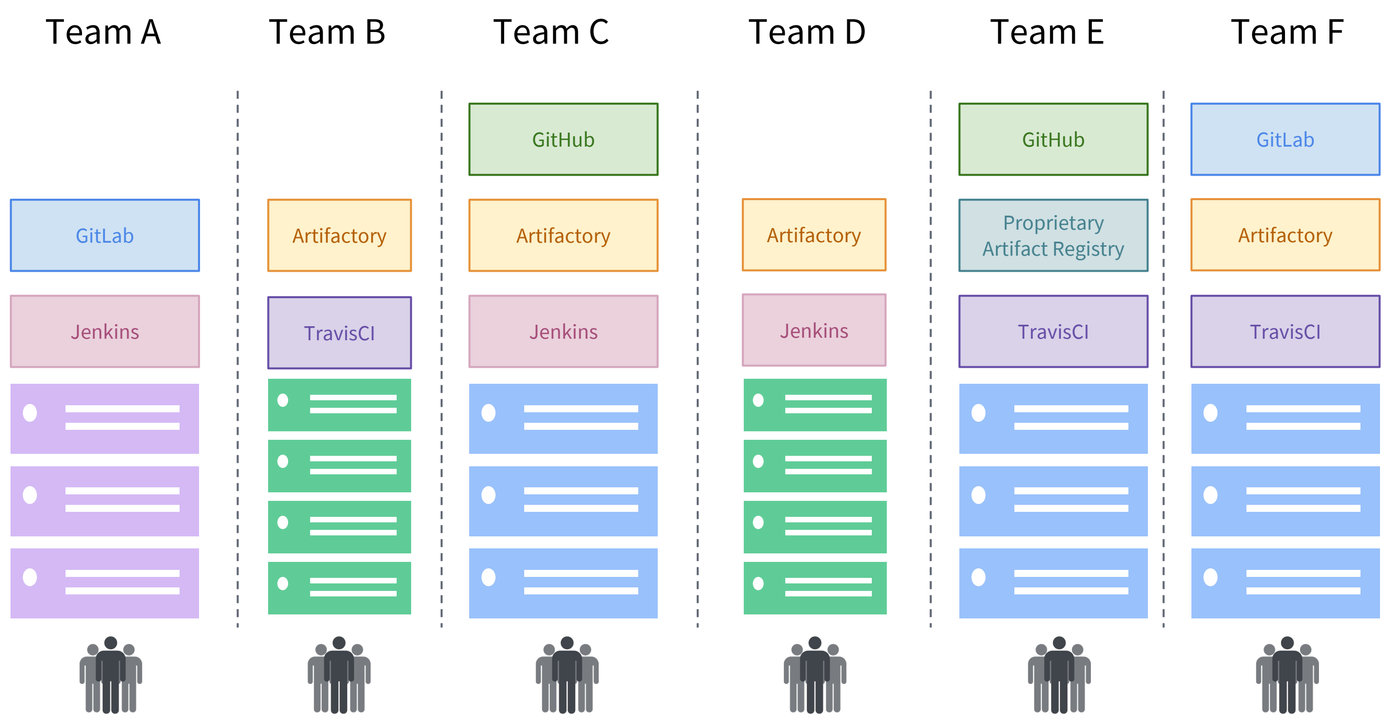
Traditional Workload Flow Stages



Modern Workload Flow Stages

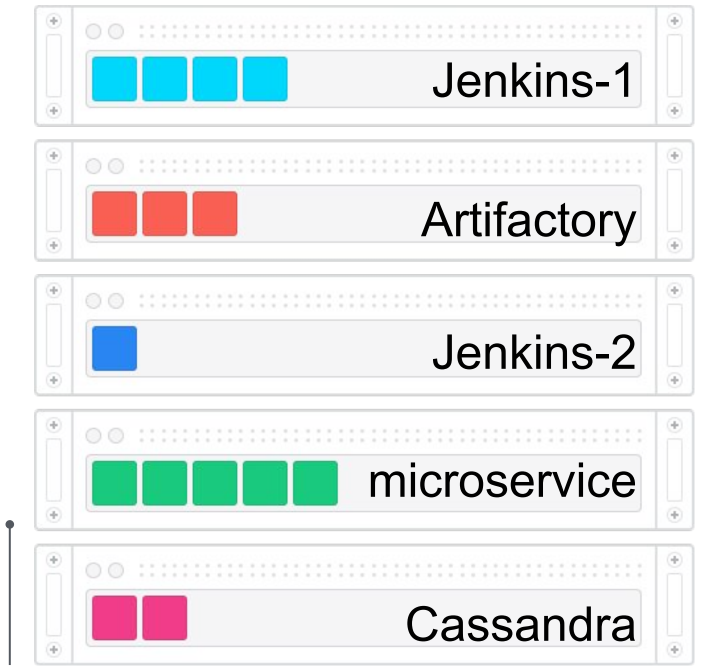
	Developers (Local, Shared)			Operator Managed (Shared)	
Stateless	Install (Local)	Development (Local Deploy)	Test	Stage (pre-prod)	Production
Stateful DB	Install (Local)	Development (Local Deploy)	Test	Stage (pre-prod)	Production
Other	Install (Local)	Development (Local Deploy)	Test	Stage (pre-prod)	Production

BUILDING & OPERATING CI/CD PIPELINES IS CHALLENGING



- 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



Industry Average
12-15% utilization

Typical Datacenter
siloed, over-provisioned servers,
low utilization

THE KERNEL: APACHE MESOS

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]

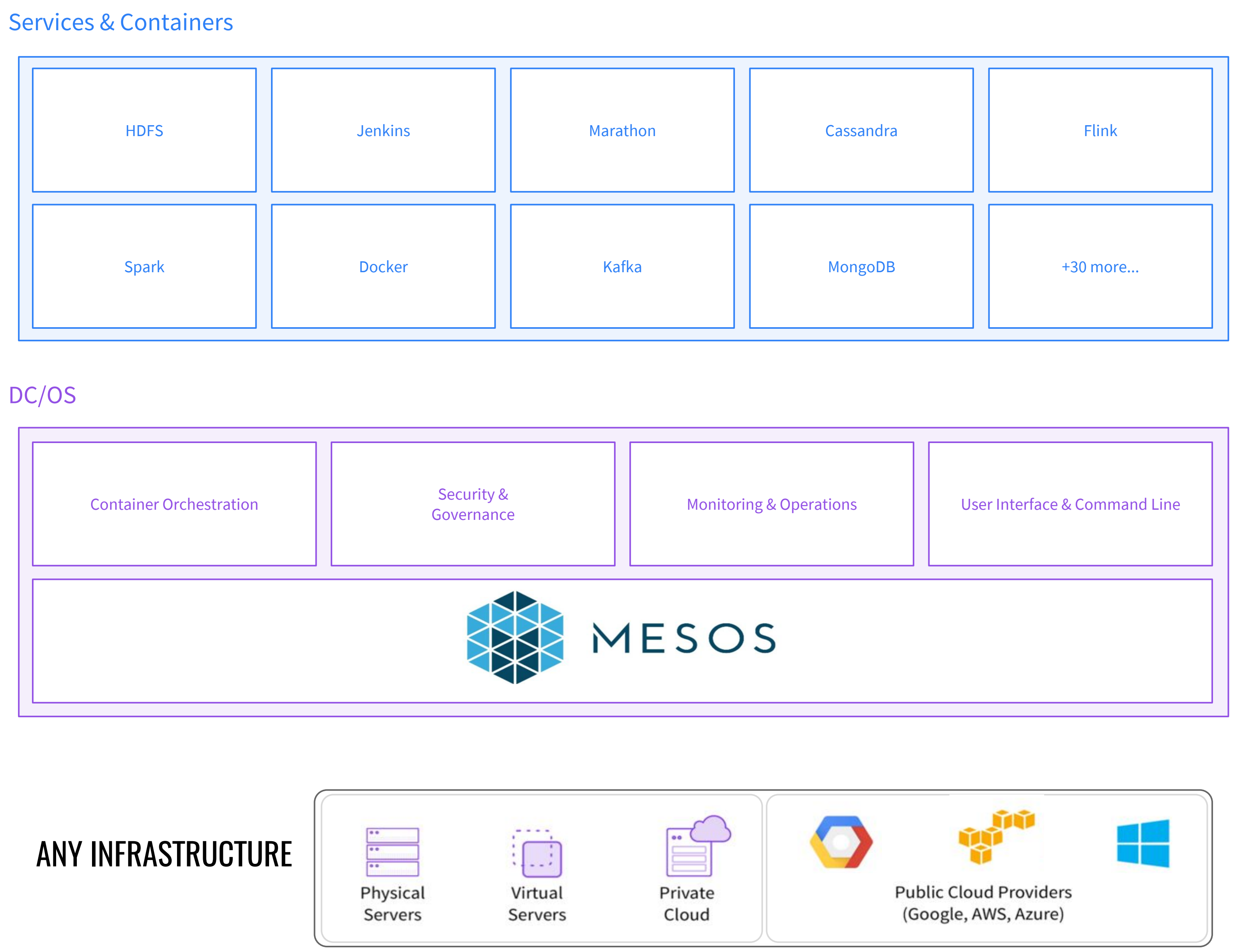
[1] http://mesos.apache.org/documentation/latest/powered-by-mesos/

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/

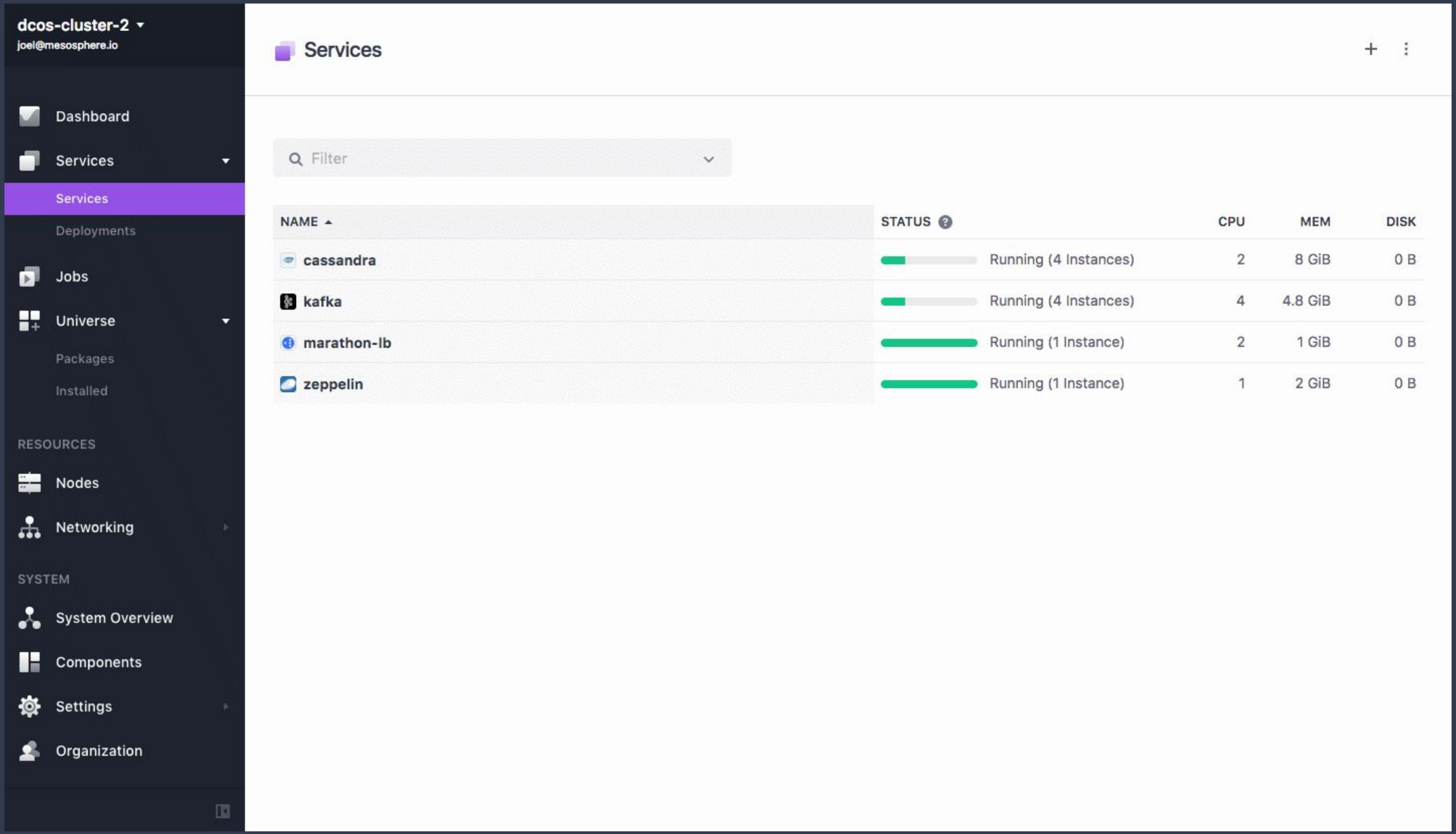
DC/OS Architecture Overview



Interact with DC/OS (1/2)

Web-based GUI

https://dcos.io/docs/latest/usage/webinterface/



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)

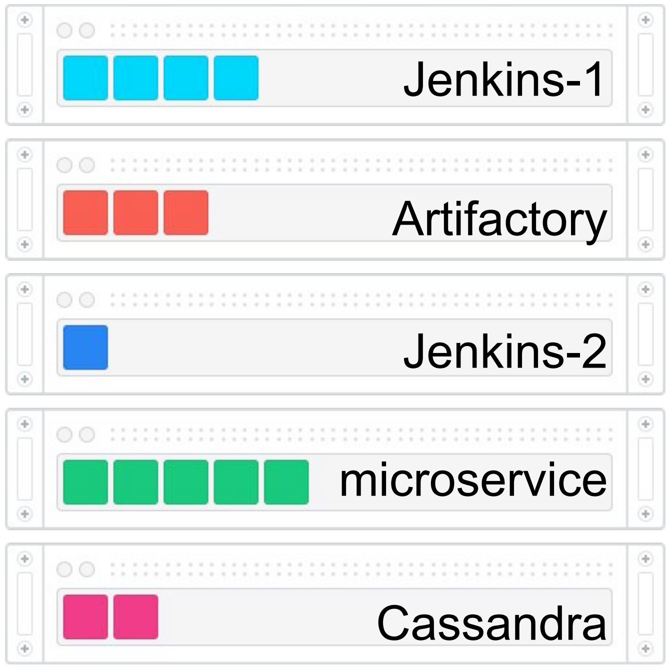
CLI tool

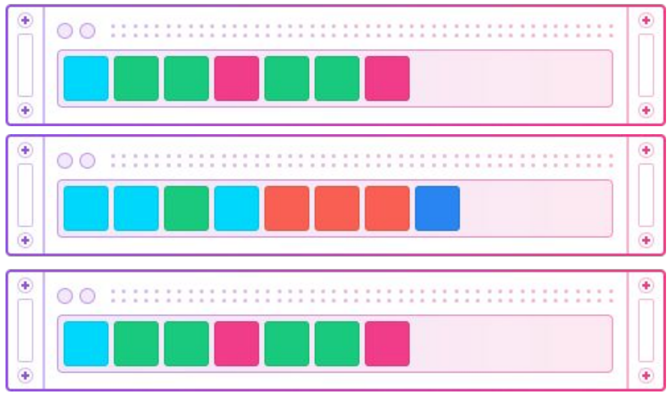
https://dcos.io/docs/latest/usage/cli/

API

https://dcos.io/docs/latest/api/

MULTIPLEXING OF DATA, SERVICES, USERS, ENVIRONMENTS





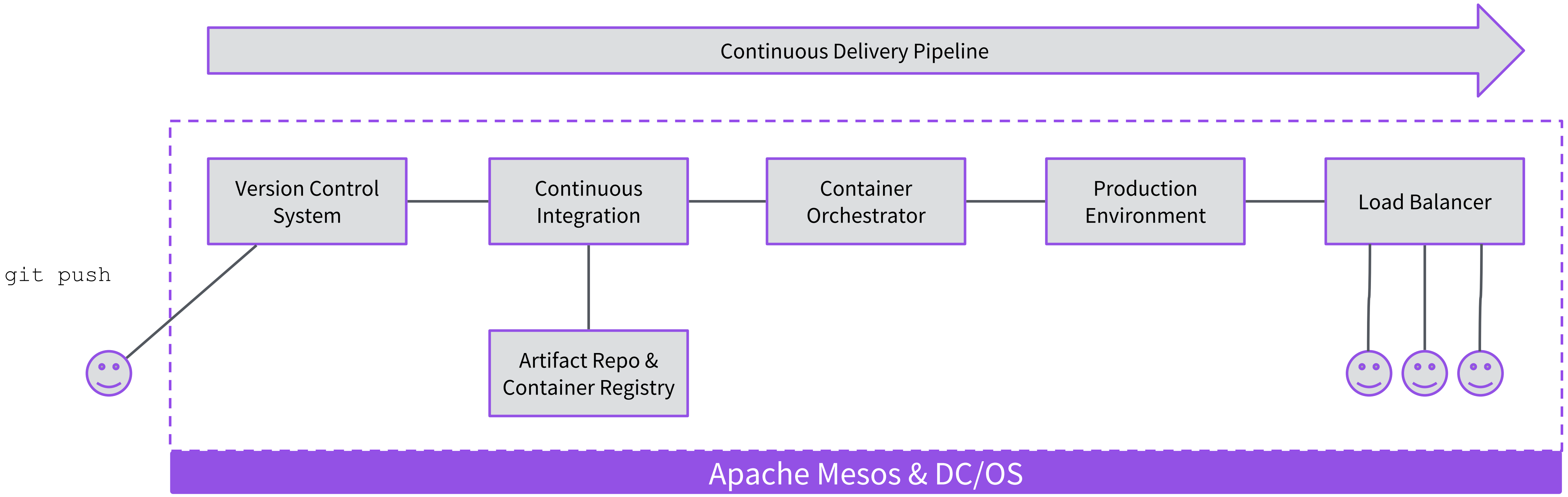
Typical Datacenter
siloed, over-provisioned servers, low utilization

Mesos/ DC/OS
automated schedulers, workload multiplexing onto the same machines

1

Development Team Self-Service for CI/CD

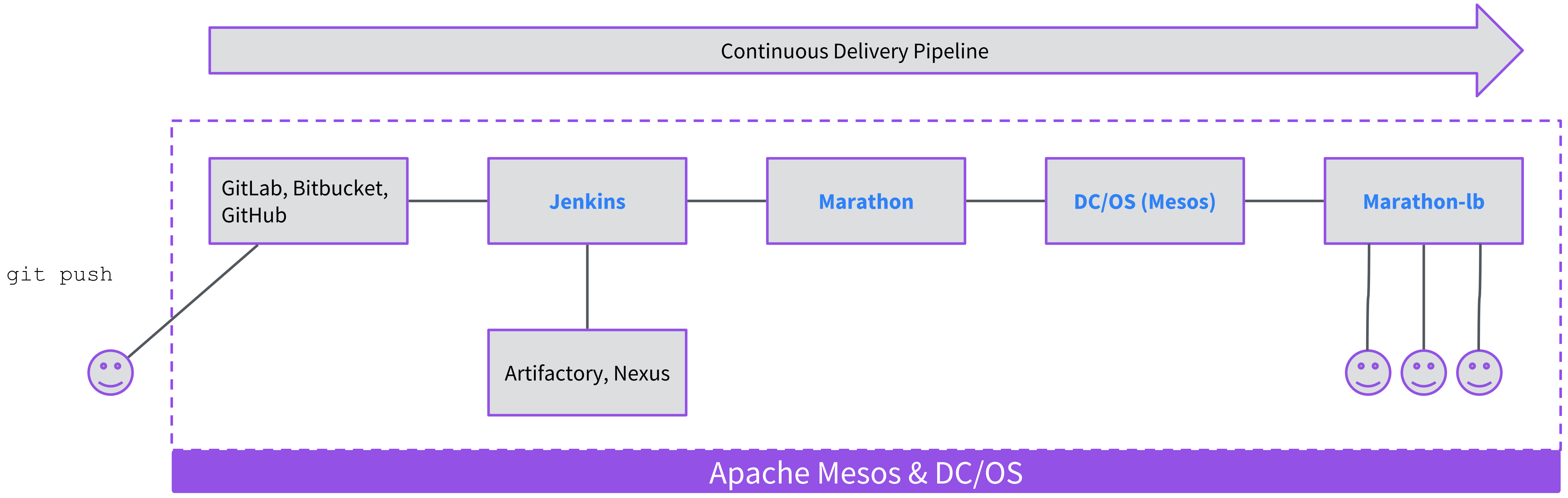
RELIABLE, SIMPLIFIED CI/CD INTEGRATION with DC/OS



1

Development Team Self-Service for CI/CD

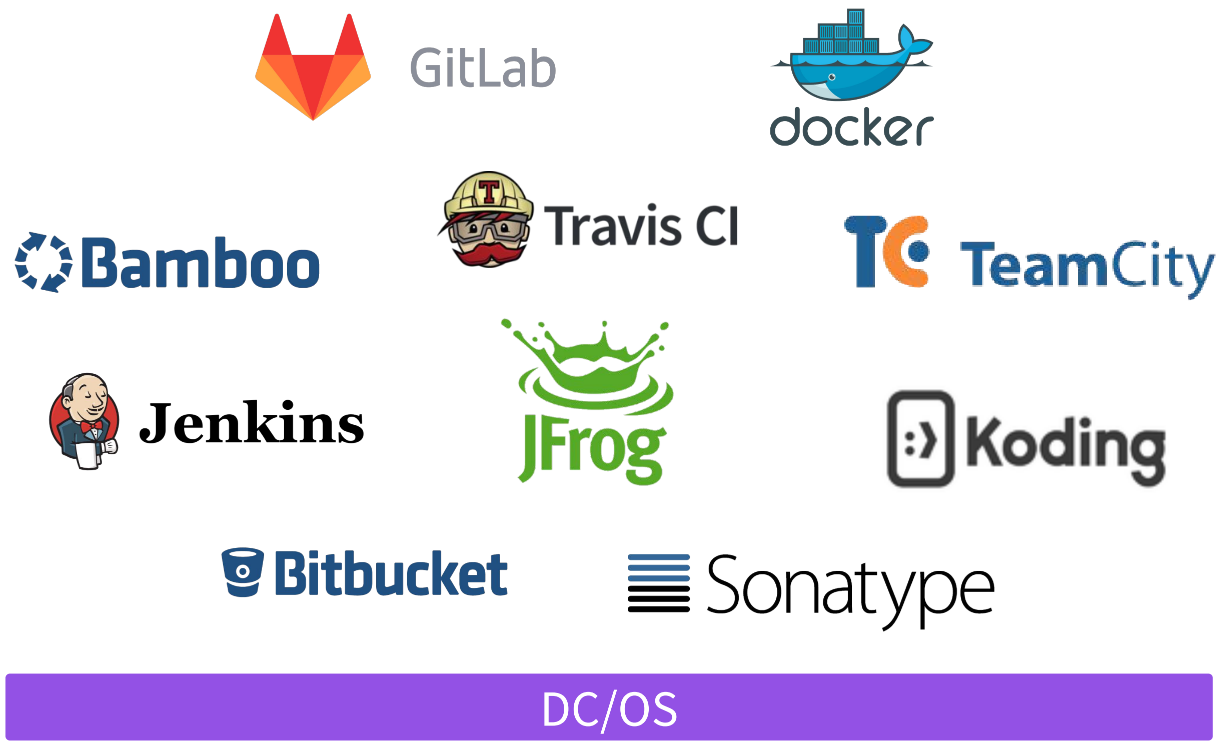
RELIABLE, SIMPLIFIED CI/CD INTEGRATION with DC/OS



1

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

A MODERN RELEASE PROCESS

1 **Development Team Self-Service for CI/CD**

- Scale services instances to provide on-demand Build/Test/Staging infrastructure with reduced time & cost to provision
- Manage multiple installations for different groups; centralized role based access control to all cluster resources
- Choose the tools you want and get support from partners for enterprise tools integrated with DC/OS

2 **Elastic Scaling with Resource Optimization for build bursting**

- Teams share the same pool of resources, dramatically increasing utilization (6,000 builds/day on 46 physical machines - eBay)
- Use CI/CD tools of your choice with DC/OS, and run everything on the same shared infrastructure
- Health checks to ensure developer tools are always up and running; if an instance fails, it is automatically restarted without data loss

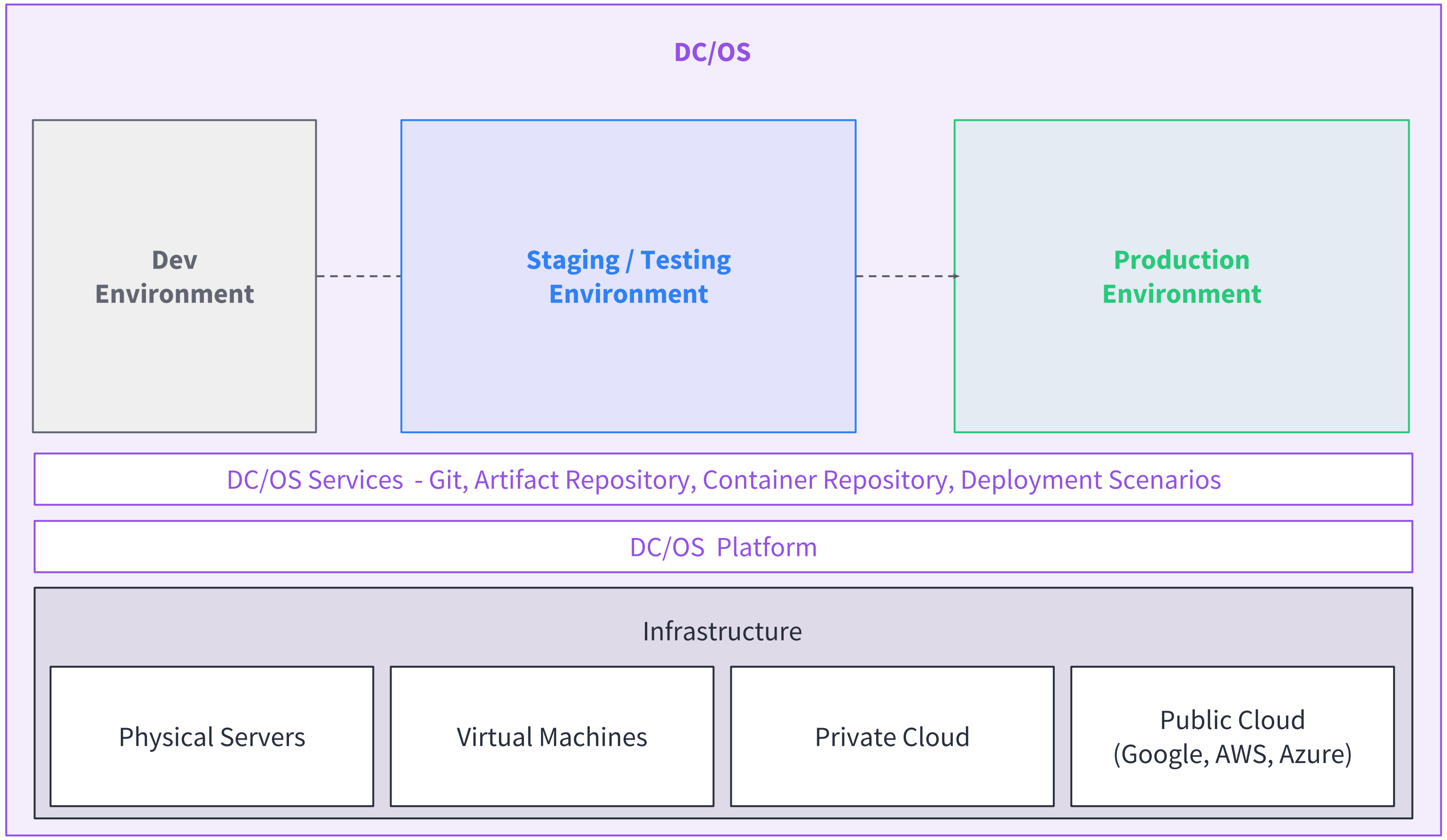
3 **Build and deploy traditional and modern apps on the same infrastructure**

- Identical infrastructure across Test/Staging/Production with strong isolation

3

Build and deploy modern apps on the same infra

APPLICATION LIFECYCLE


DC/OS
Dev Environment
Staging / Testing Environment
Production Environment
DC/OS Services - Git, Artifact Repository, Container Repository, Deployment Scenarios
DC/OS Platform
Infrastructure
Physical Servers
Virtual Machines
Private Cloud
Public Cloud (Google, AWS, Azure)

DC/OS

- Identical Infrastructure across Test/Staging/Production with strong isolation
- Self service access

BENEFITS

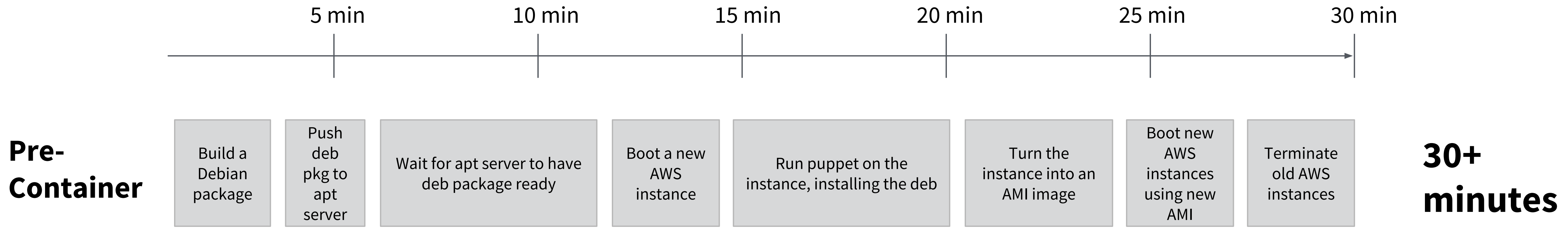
Less developer time troubleshooting environment issues

Easy experimentation with new technologies

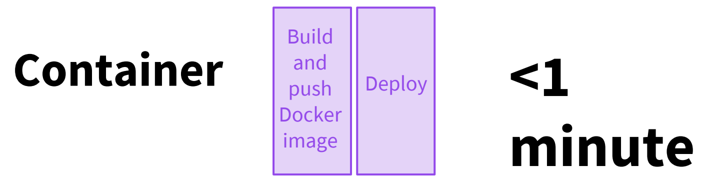
DEPLOYING APPS

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

Old vs. New Deploy Process


5 min
10 min
15 min
20 min
25 min
30 min
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.”



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

http://labs.strava.com/blog/mesos/

@dcos

chat.dcos.io

users@dcos.io

/dcos
/dcos/examples
/dcos/demos

Questions?

Elizabeth K. Joseph
Twitter: @pleia2
Email:
ejoseph@mesosphere.com