# DC/OS Fundamentals

Day One - Overview

MESOSPHERE

# Course Goals

Learn and understand:

- What DC/OS is and what sets it apart from others
- The various installation methods
- DC/OS cluster terminology and architecture
- The package catalog
- Accessing your cluster
- Container orchestration capabilities of Marathon
- Load-balancing, service-discovery, and networking concepts
- Security capabilities of Enterprise DC/OS

Gain hands on experience:

- Installing DC/OS
- Using various GUI and CLI based DC/OS clients
- Writing Marathon service specifications

# Course Prerequisites

- Basic understanding of software containers
- General Linux system administration and command line reference (e.g. `ls`, `mkdir`)
- Basic skills in SSH, git, JSON, and command line text editors (e.g. `vim`, `nano`)
- A laptop which has a terminal/SSH client application (iTerm, Konsole, gnome-terminal, PuTTY)

# Course Agenda - 2 Days Live / 3 Days Virtual

**Day One**

- Gain an understanding of the DC/OS use case
- Learn what makes up a DC/OS installation
- DC/OS clients (GUI, CLI)
- Leveraging the package catalog

**Day Two**

- Learn how to define and run services with Marathon
- Understand implementing health checks and readiness checks
- Learn about monitoring Marathon stats and logs

**Day Three**

- Service discovery, load balancing and the overlay network
- Security, authentication, and identity management in Enterprise DC/OS

# Classroom Lab Environment

Your instructor will provide 5 hosts running on AWS with a list of IP addresses that identify how to connect to your cluster nodes.

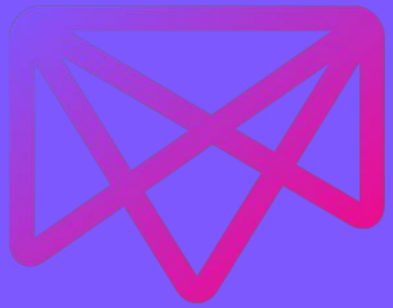
The list will include:

- 5 publicly accessible IP addresses along with internal IP addresses in the 10.0.0.x subnet
- 1 bootstrap node, 1 master node, 1 public agent, and 2 private agents

# Agenda - Day One

1. What is DC/OS?
2. Installing DC/OS overview and lab
3. DC/OS cluster architecture
4. Managing DC/OS through the GUI
5. The package catalog
6. Managing DC/OS through the CLI
7. SSH connectivity with your cluster

# What is DC/OS?



DC/OS

# Elevator Pitch

*DC/OS (**D**istributed **C**loud **O**perating **S**ystem) abstracts your data center into a single “computer”, pooling distributed workloads and resources, simplifying rollout and operations*

# DC/OS: Proven Platform for Running Modern Applications

2000... 2010... 2013 2014 2015 2016

**Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center**

Benjamin Hindman, Andy Konwinski, Matei Zaharia,
Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, Ion Stoica
*University of California, Berkeley*

Thursday 30th September, 2010, 12:57

**Abstract**

We present Mesos, a platform for sharing commodity clusters between multiple diverse cluster computing frameworks, such as Hadoop and MPI. Sharing improves cluster utilization and avoids per-framework data repli-

The solutions of choice to share a cluster today are either to statically partition the cluster and run one framework per partition, or allocate a set of VMs to each framework. Unfortunately, these solutions achieve neither high utilization nor efficient data sharing. The main

Microsoft Hewlett Packard Enterprise

60+ Partners

Strategic Investments

DCOS Launched | DC/OS OSS Project

Mesosphere

Proprietary | Apache Mesos Project

Google Borg & Omega

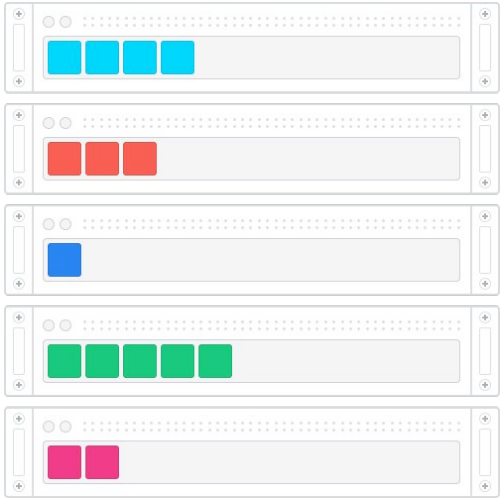
facebook. Tupperware & Bistro

airbnb PayPal NETFLIX Microsoft Azure

TWO SIGMA GROUPON verizon yelp

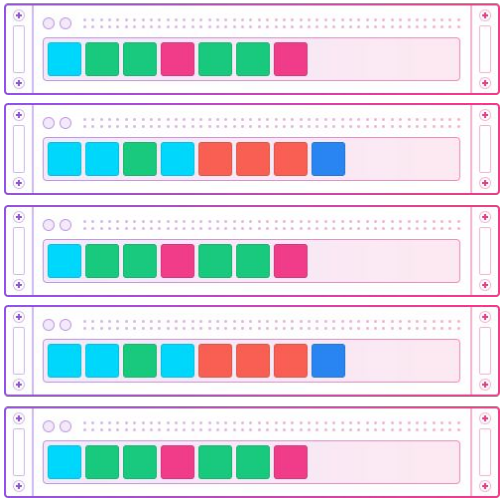
Bloomberg esri UBER

# DC/OS: Run Everything on the Same Shared Cluster



**Industry Average**
12-15% utilization

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



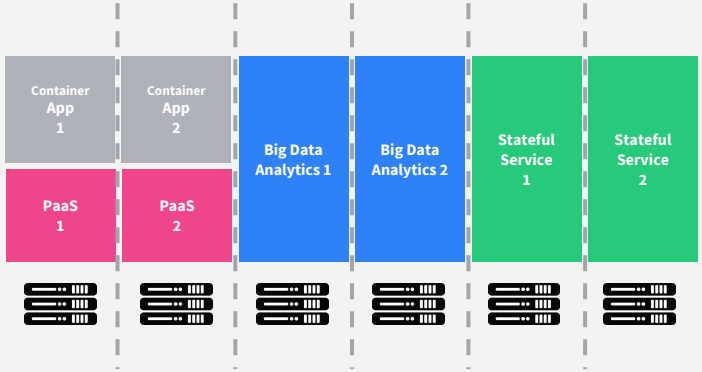
**DC/OS Datacenter**
automated schedulers, workload multiplexing onto the same machines

**DC/OS Multiplexing**
30-40% utilization, up to 96% at some customers

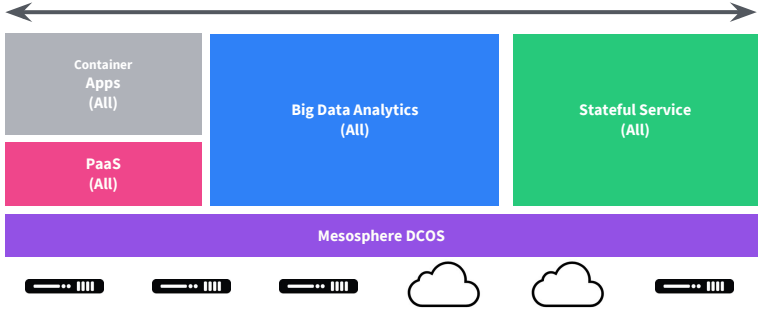
**4X**

# Hyperscale Operations

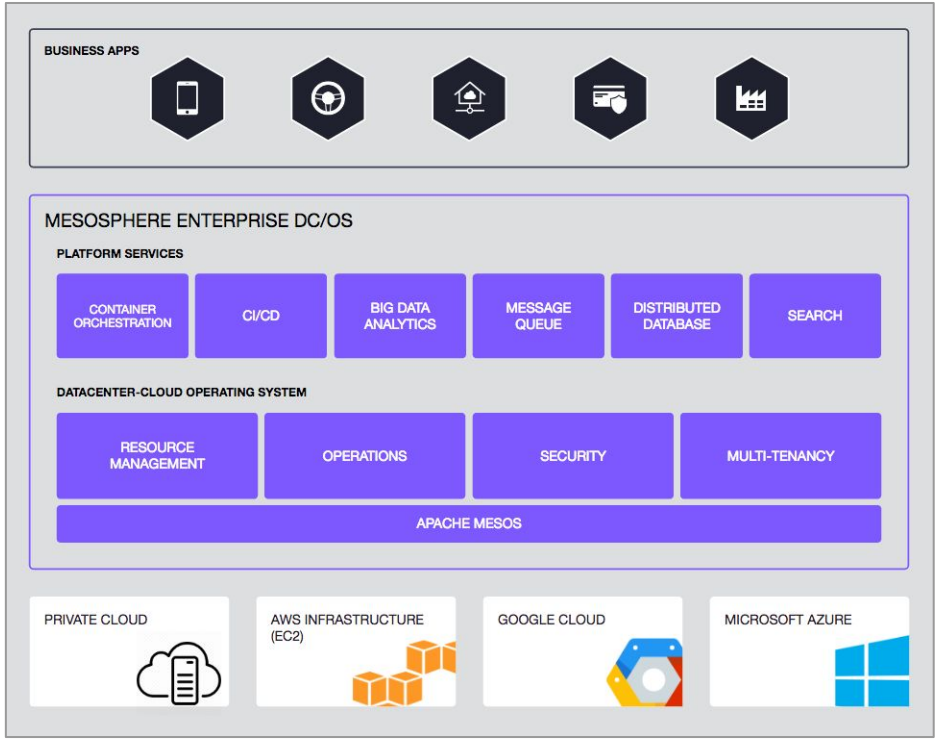
## TRADITIONAL APPROACH




## MESOSPHERE DC/OS APPROACH



# Core Concepts




## SERVICES & CONTAINERS + LEGACY WORKLOADS

- Resource Isolation

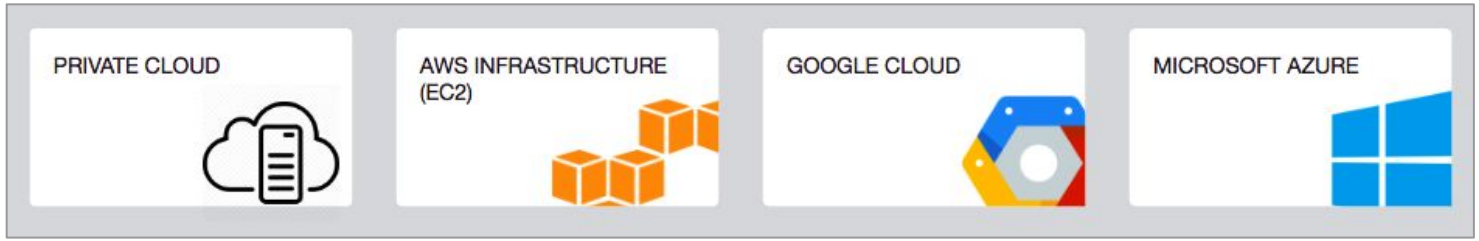
## MESOSPHERE DC/OS

- Elasticity

## ANY INFRASTRUCTURE

- Fault Tolerant & Highly Available

# Any Infrastructure

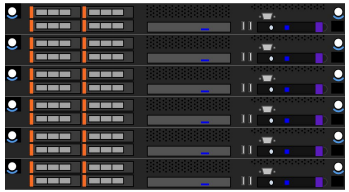


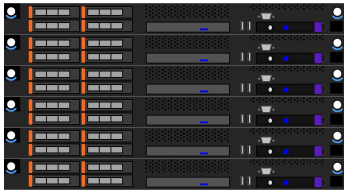

- AWS, Azure, and Google Cloud all “supported”
- On premise data centers or private clouds
- Virtualized infrastructure
- Hardware details and requirements coming up in installation section

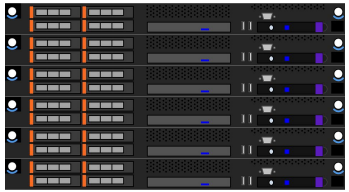
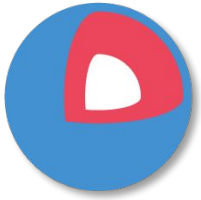
# Installation

# Cluster Building Blocks

- Clusters are made up of Linux servers where each one is referred to as a **node**
- DC/OS 1.12 supports the following operating systems:
  - Red Hat Enterprise Linux 7.3, 7.4, 7.5
  - CentOS 7.3, 7.4, 7.5
  - CoreOS 1800.6.0, 1800.7.0, 1855.4.0
  - For current supported versions, find the support matrix here
- Different nodes have different functions/responsibilities and as such have different hardware requirements and recommendations

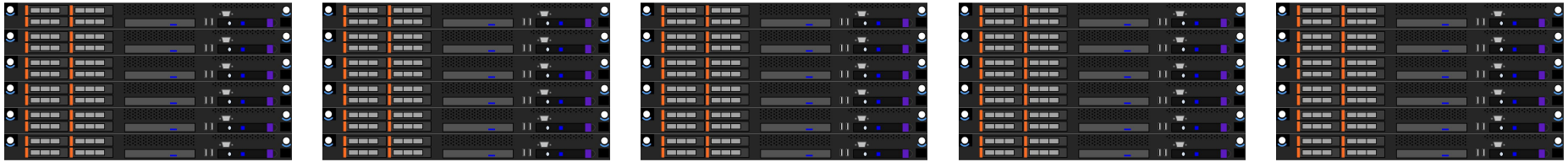




# Types of Nodes

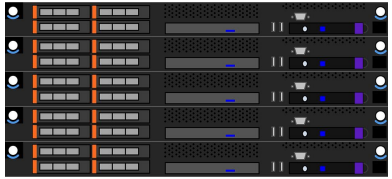
- DC/OS clusters are made up of two main types of nodes: **master** nodes and **agent** nodes
- **Masters** run the cluster, running many APIs, store the state of the cluster, and provide an entry point into the cluster
- **Agents** are often referred to as **slave** nodes in both config files, documentation, and in both CLI and GUI output - they are there to provide the resources that will be allocated to run your workloads



# Master Nodes - Hardware

- Master nodes control the Mesos layer and run many key processes for DC/OS
- **Must** come in odd numbers for Zookeeper election purposes
  - One: Training, testing, proof of concept
  - Three: Development
  - Five: Development or Production
- Masters do not horizontally scale - you choose your number before you install
- Base hardware requirements (see right) are sufficient for small clusters: < 20 nodes
- Size up memory/CPU for larger clusters
- Consider SSDs regardless of cluster size - Zookeeper loves them

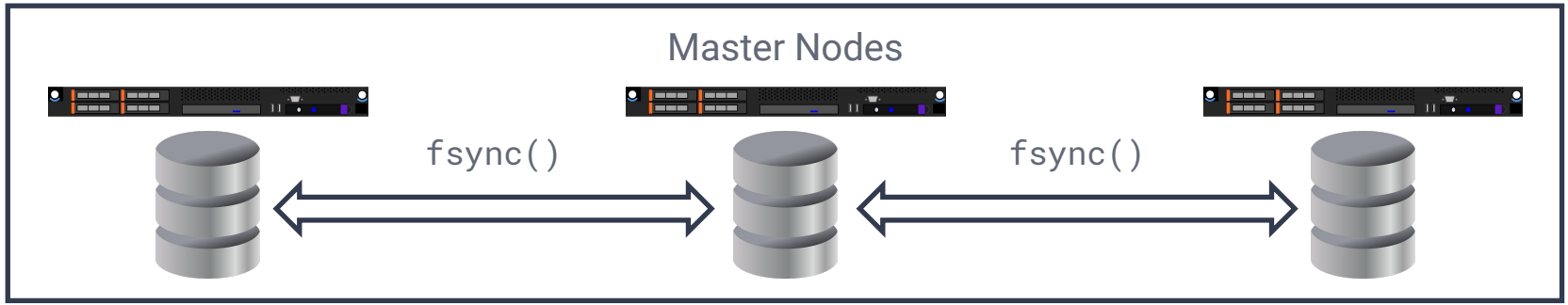
Master Nodes



| | Minimum | Recommended |
|---|---|---|
| Nodes | 1 | 3 or 5 |
| CPU | 4 cores | 4 cores |
| RAM | 32 GB | 32 GB |
| Disk | 120 GB | 120 GB |

# Data Replication Across Masters

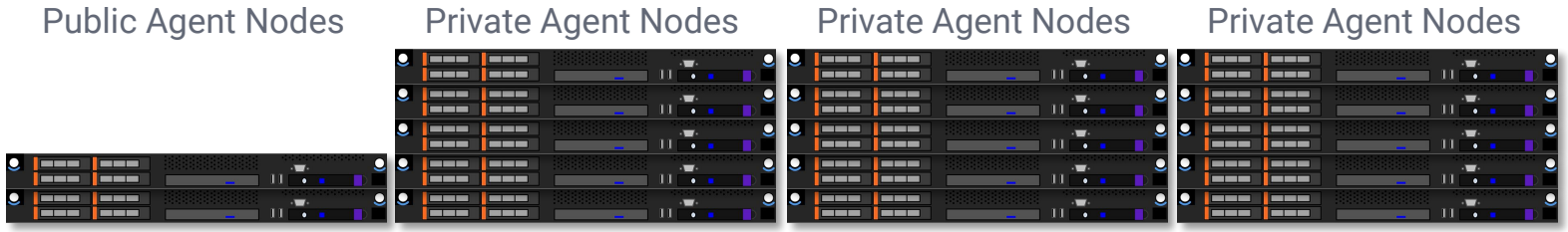
- Master nodes need to communicate and update each other often to synchronize state of the cluster
- Mesosphere recommends the following settings for DC/OS masters disk arrays in production:
  - SSDs using RAID controllers with a BBU and the RAID controller cache configured in writeback mode



# Agent Nodes - Hardware

- Agent nodes run your applications
- Agent nodes also run some DC/OS related processes
- Agents can be added to an existing cluster at any time
- Number of agents along with their sizing depends greatly on the requirements of your applications

| | Minimum | Recommended |
|---|---|---|
| Nodes | 1 | 6 or more |
| CPU | 2 cores | 2 cores |
| RAM | 16 GB | 16 GB |
| Disk | 60 GB | 60 GB |

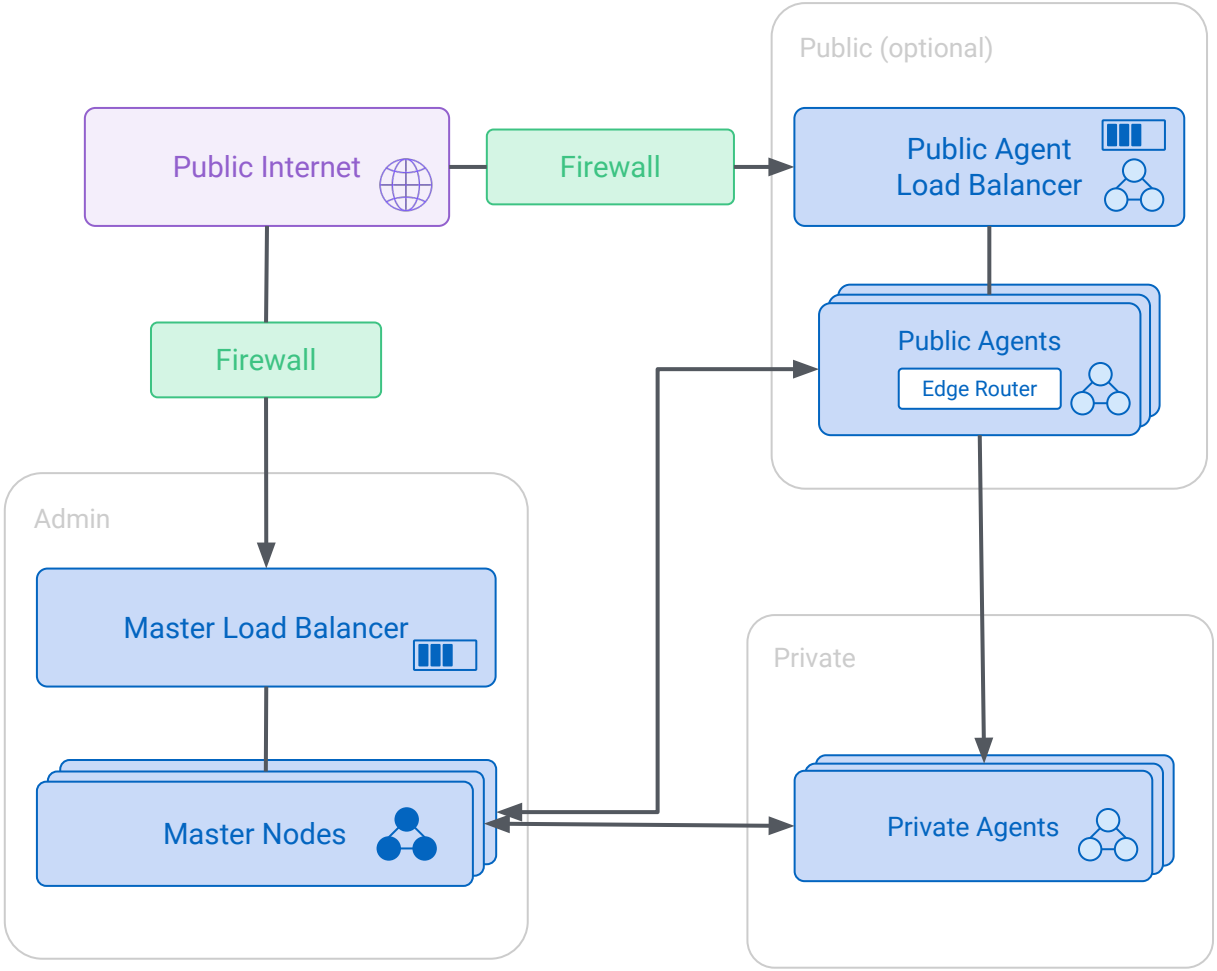


# Agent Nodes: Public vs. Private

There are two types of agent nodes: **public** agents and **private** agents

| Public Agents | Private Agents |
|---|---|
| Run services that facilitate connections outside the cluster - usually a load balancer | The default agent and run most everything else |
| Exist on a network that is accessible from outside the cluster | Exist on a network that is accessible only within the cluster |

# Network Topology



# Master and Agent Disk and Filesystem Requirements

- The `/opt` directory must not be mounted on LVM or shared storage
  - DC/OS is installed under `/opt/mesosphere` meaning this path must be available when `/` is mounted at boot
- There should be at least 10GB free in the `/var` directory - this is where container sandboxes are found
- Mesos persistent data is stored in `/var/lib/mesos`

# Bootstrap Node

- A single bootstrap node is required to install the cluster
- The bootstrap node does not participate in the cluster it installs - only hosts the bits required to install (and later upgrade) masters and agents
- Can be backed up and shut down after the installation is complete
- Minimum 2 core CPU and 16GB of RAM are required to service the installation process
- Must have open networking between bootstrap node and all potential cluster nodes

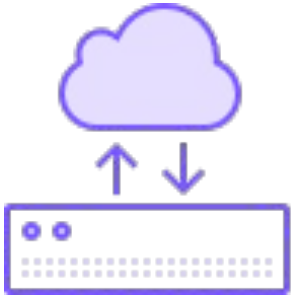
# Software Requirements - All Nodes

- `tar`, `xz`, `unzip`, `curl`, `ipset`, and `docker` are required on all nodes
- SELinux must use the `targeted` policy when placed in `enforcing` mode
  - Documentation
- Supported Docker versions can be found at the version support matrix here
- See details for storage drivers and other Docker settings:
  - Documentation

# Three Installation Methods

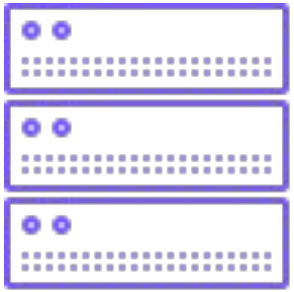
## Development

- Run a cluster on your laptop
- Good for first-time users or developers intending to build services or modify DC/OS
- Free



## Evaluation

- Good for fast demos and POCs
- Can be done on a cloud provider through automation tools such as CFN, ARM, and Terraform
- Can be done on-prem through automation tools such as Ansible, Chef, and Puppet



## Production

- Good for long-lived clusters running business-critical workloads
- Manual, by default
- Can be automated through automation tools such as Ansible, Chef, and Puppet

# Installation Method: Development

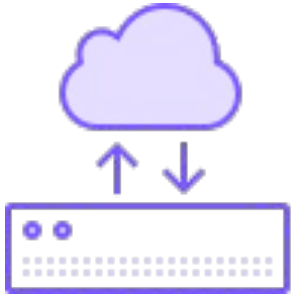
Available Tools:

- DC/OS Vagrant - DC/OS on virtual machines using Vagrant + VirtualBox
- DC/OS E2E - DC/OS on containerized nodes using Docker

Tool Selection Scenarios:

- DC/OS Vagrant works on Windows, MacOS, or Linux while DC/OS E2E requires MacOS or Linux
- DC/OS E2E is substantially faster to deploy
- DC/OS Vagrant more accurately simulates a real cluster be designating resources allocated to each node, while DC/OS E2E allows over-subscription of your machine resources
- DC/OS E2E is more stable across releases because it only requires Docker

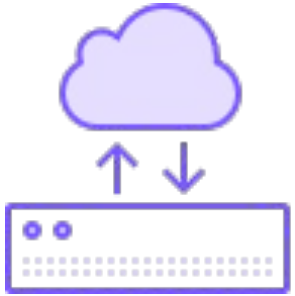
# Installation Method: Evaluation



Available Tools for Cloud-based Clusters:

- DC/OS on AWS with the Universal Installer
- DC/OS on Azure with the Universal Installer
- DC/OS on GCP with the Universal Installer

# Installation Method: Evaluation



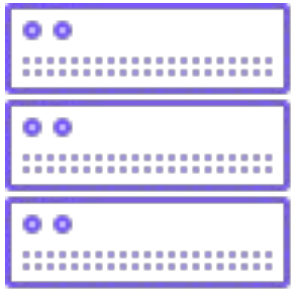
Available Tools for On-Prem Clusters:

- DC/OS installation with Ansible
- DC/OS installation with Chef
- DC/OS installation with Puppet

Limitations:

- All tools are community driven and not officially supported by Mesosphere

# Installation Method: Production



- Most commonly used install - used for both production and development clusters
- Nodes can be:
  - On-prem physical
  - On-prem virtual
  - Private cloud instances
  - Public cloud instances
- All masters must reside in a single region
- Agents can reside in different regions
- All installation steps are performed manually on each node - typically gets automated through a configuration management tool of choice

# Installation Configuration - `ip-detect`

- On your bootstrap node, you must provide an `ip-detect` script that will be distributed to every node in the cluster
- This file is required for nodes to understand which network interface to bind to for internal cluster communication
- Once a node is installed, it's IP address cannot change - if it does you must reinstall the node
- The script expects to live in a specific location on the filesystem of the bootstrap node (see: right)
- See the docs for further requirements:
  - [Documentation]

```
$ tree -R -L 2

├── dcos_generate_config.ee.sh
├── genconf
│   ├── license.txt
│   ├── ip-detect
│   ├── config.yaml
```

# Installation Configuration - `license.txt`

- For Enterprise DC/OS a valid software license is required to install your cluster
- File should be on disk of your bootstrap node, found in the the `genconf` directory and named `license.txt`
- Contact your sales rep to get a valid license key
- Licenses can be managed through DC/OS CLI
  - [Documentation]

```
$ tree -R -L 2

├── dcos_generate_config.ee.sh
├── genconf
│   ├── license.txt
│   ├── ip-detect
│   ├── config.yaml
```

# Installation Configuration - `config.yaml`

- Installation process and configuration is specified by a configuration file named `config.yaml`
- Used to declare masters, public, and private agents, and much more!
- [Documentation]

```yaml
bootstrap_url: http://10.0.0.5
cluster_name: MyOrg-Production
master_discovery: static
master_list:
- 10.0.0.50
resolvers:
- 8.8.8.8
- 8.8.4.4
exhibitor_storage_backend: static
fault_domain_awareness: true
security: permissive
```

# `config.yaml` - Zookeeper Settings

`exhibitor_storage_backend: static`

- Use this setting to run Zookeeper on your master nodes and store its data locally

`exhibitor_storage_backend: zookeeper`

- Use this setting to use an existing Zookeeper cluster that is separate from the DC/OS cluster - requires further settings in `config.yaml` for login credentials, etc.

`exhibitor_storage_backend: aws_s3`

- Use this setting to run Zookeeper on your master nodes and store its data in an S3 bucket – requires further settings in `config.yaml` for login credentials, etc.

`exhibitor_storage_backend: azure`

- Use this setting to run Zookeeper on your master nodes and store its data in an Azure storage account – requires further settings in `config.yaml` for login credentials, etc.

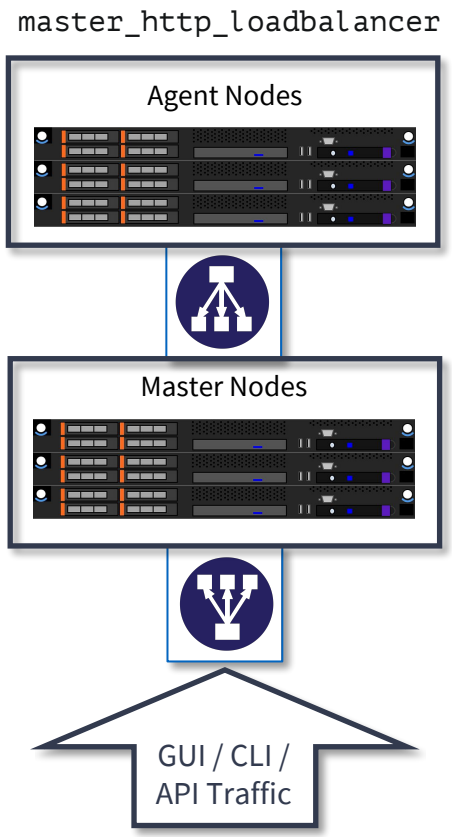
# `config.yaml` - Master Discovery

`master_discovery: static`

Use this setting if you have a static list of IPs to assign to your masters or if you only have a single master node
Useful for situations where you have full control over your IPs
If you replace or rebuild a master node, you must reuse the same IP of the node that is being replaced

`master_discovery: master_http_loadbalancer`

Use this setting to use dynamic IPs for your masters
You must stand up load balancers between your master nodes and their agent nodes and then specify this load balancer URL in the `config.yaml`
You must also place a load balancer between your masters and any administrative traffic to your cluster – API calls, GUI logins, CLI etc.

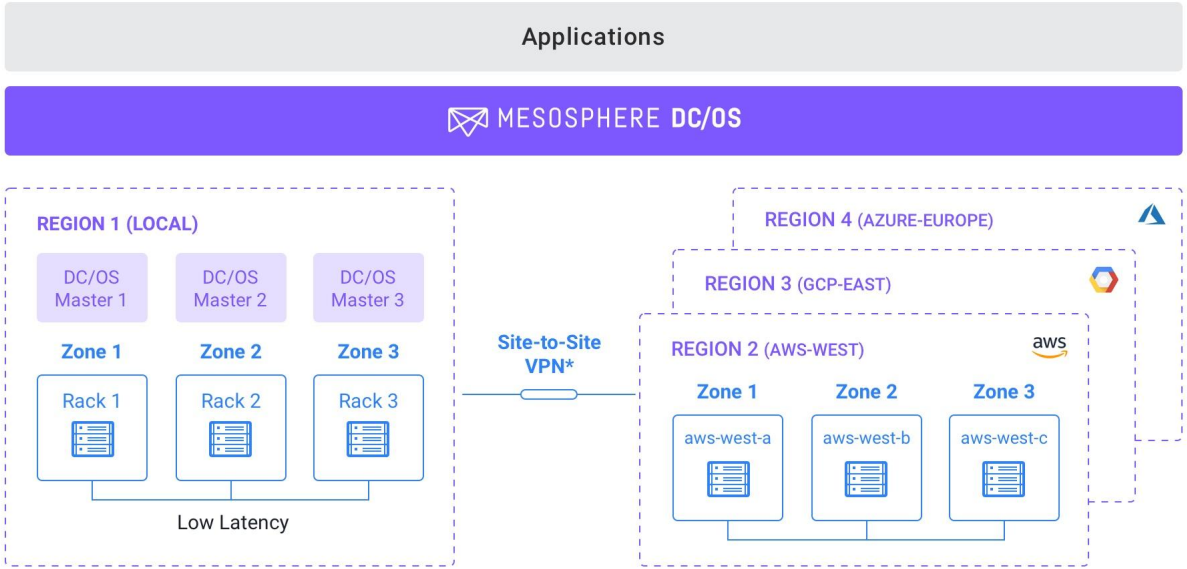


# `config.yaml` - Capacity Bursting

- New in DC/OS 1.11
- Provides the ability to add cloud instances as agents for peak capacity bursting
- New stateless and stateful services can be scheduled to run on cloud instance agents
- Requirements at install time:
  - `fault_domain_awareness`: true in `config.yaml`
  - fault domain detect script in `genconf` directory (Example)
  - Create tarball of `genconf/serve` after installer has been executed
- Adding remote capacity:
  - Provision remote infrastructure (e.g. EC2 instances)
  - Copy tarball to each node
  - Install DC/OS as public/private agents on each node

# Capacity Bursting

## Regions (Clouds) & Zones (Fault Domains)




*Max latency between regions is <100ms

# `config.yaml` - Other Key Configurations

`security_mode: permissive`

- Cluster encryption turned on, but unencrypted frameworks still allowed

`security_mode: strict`

- Encrypt all the things! (Details covered in security section of training)

`bouncer_expiration_auth_token_days: '0.5'`

- Auth timeout specified in days including fractions using the Python float syntax in quotes
- Default: '5'

`resolvers:`

- Authoritative upstream DNS resolvers outside the cluster – max 3
- Care should be taken since these can only be changed by a full upgrade of the cluster

Examples in the documentation

# Installation Process - High Level

1. Download the installation file (`dcos_generate_config.ee.sh`) from the [Mesosphere Support Portal]
2. Upload installation file to Bootstrap node
3. In the same directory as the installation file, create a directory called `genconf` and write your `config.yaml`, `ip-detect`, and `license.txt` files
4. Generate install bits
5. Serve install bits over HTTP or HTTPS
6. Install DC/OS on all nodes from the Bootstrap node

# Installation Process - Detailed Steps

1. Upload the install file to the bootstrap node and create `genconf/config.yaml`, `genconf/ip-detect`
2. If installing DC/OS Enterprise Edition, create `genconf/license.txt`
3. Run the installer with `sudo bash dcos_generate_config.ee.sh` to generate the install bits
4. Run a Docker container from the `nginx` container image and configure it to serve out the generated install bits
5. SSH in to each of your nodes and copy the install script that was generated on your bootstrap node:
   a. `curl -O http://<bootstrap_ip>/dcos_install.sh`
6. Execute the script with the appropriate argument indicating what type of node you are installing:
   a. `sudo bash dcos_install.sh [master|slave|slave_public]`
7. Login to Exhibitor to watch all the master come online and for the cluster to form:
   a. `http(s)://<master_ip>/exhibitor`
8. Login to the DC/OS UI and start deploying your workloads!
   a. `http(s)://<master_ip>`

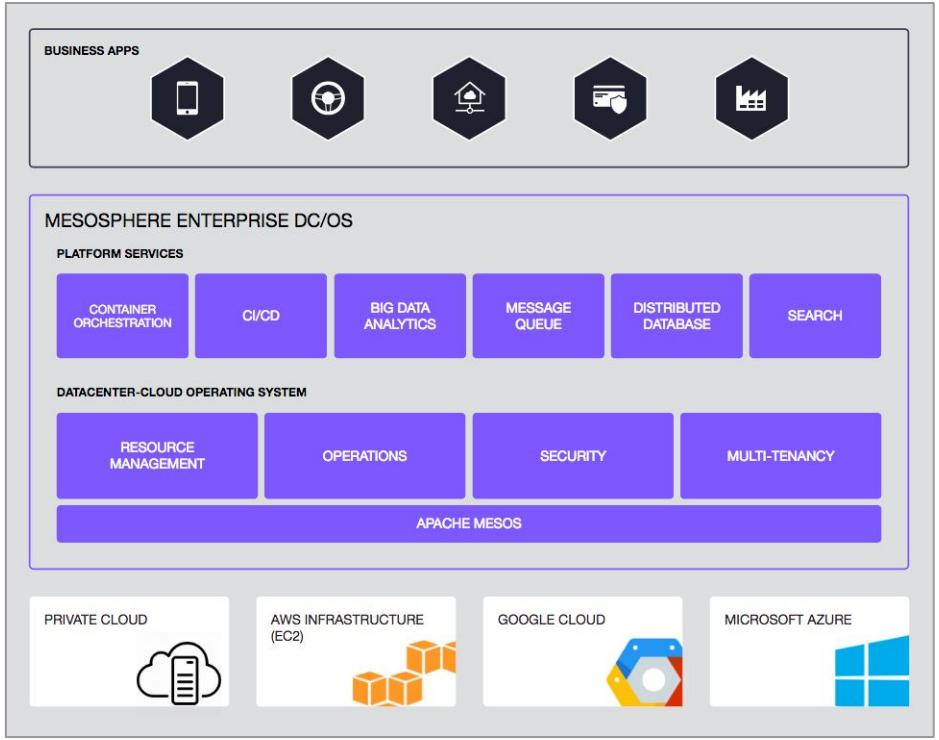
# Lab 1

Install DC/OS

# Architecture

Anatomy of a DC/OS Cluster

# Components Review




## SERVICES & CONTAINERS + LEGACY WORKLOADS

- Resource Isolation

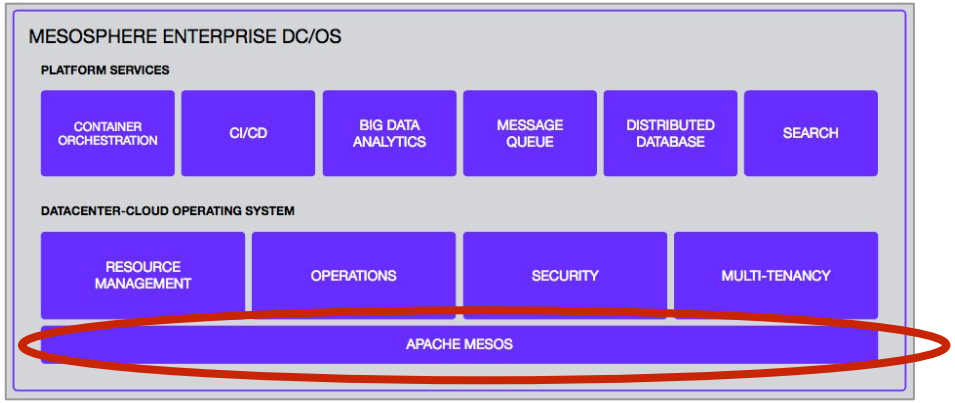
## MESOSPHERE DC/OS

- Elasticity

## ANY INFRASTRUCTURE

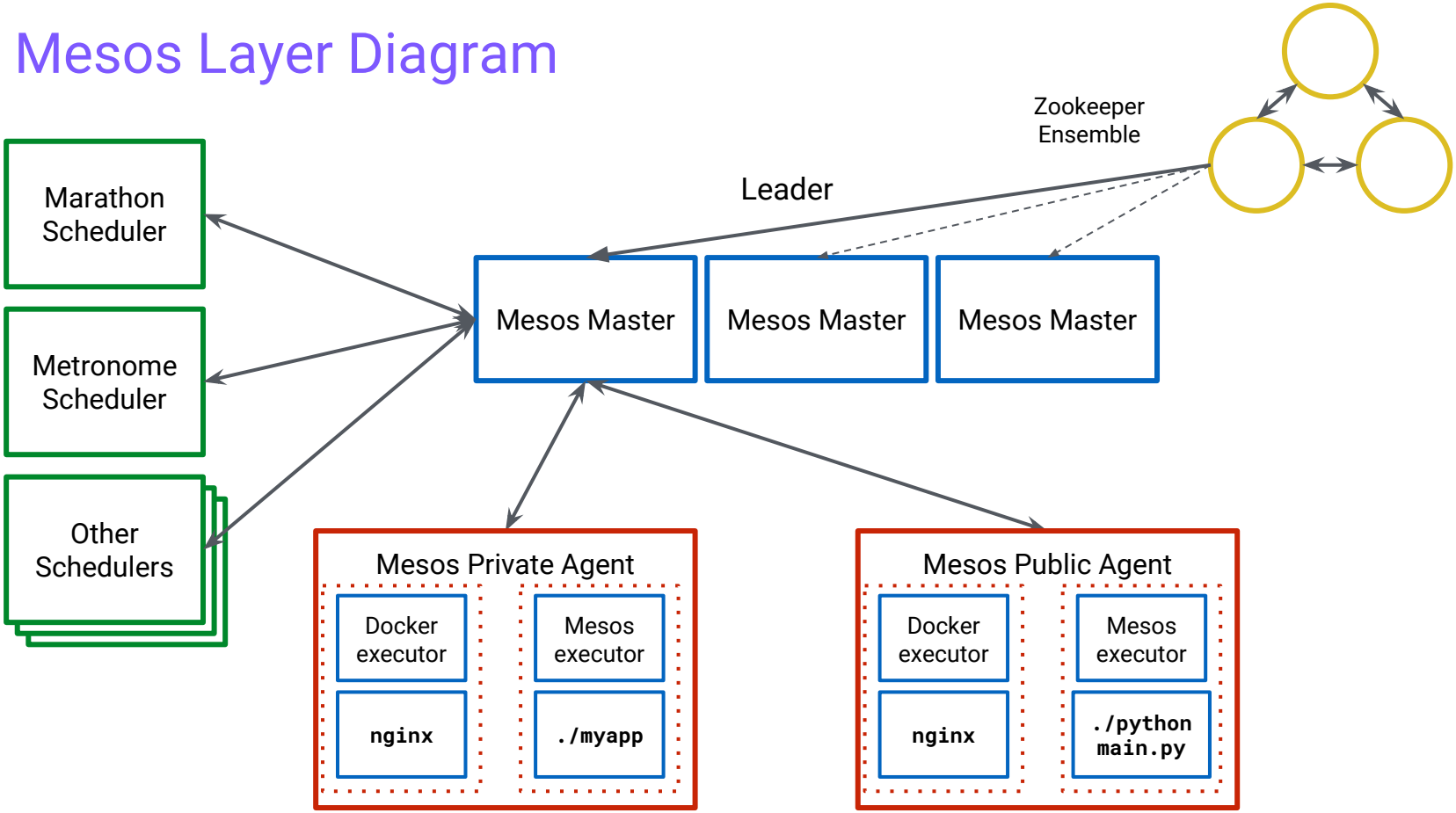
- Fault Tolerant & Highly Available

# DC/OS and Apache Mesos

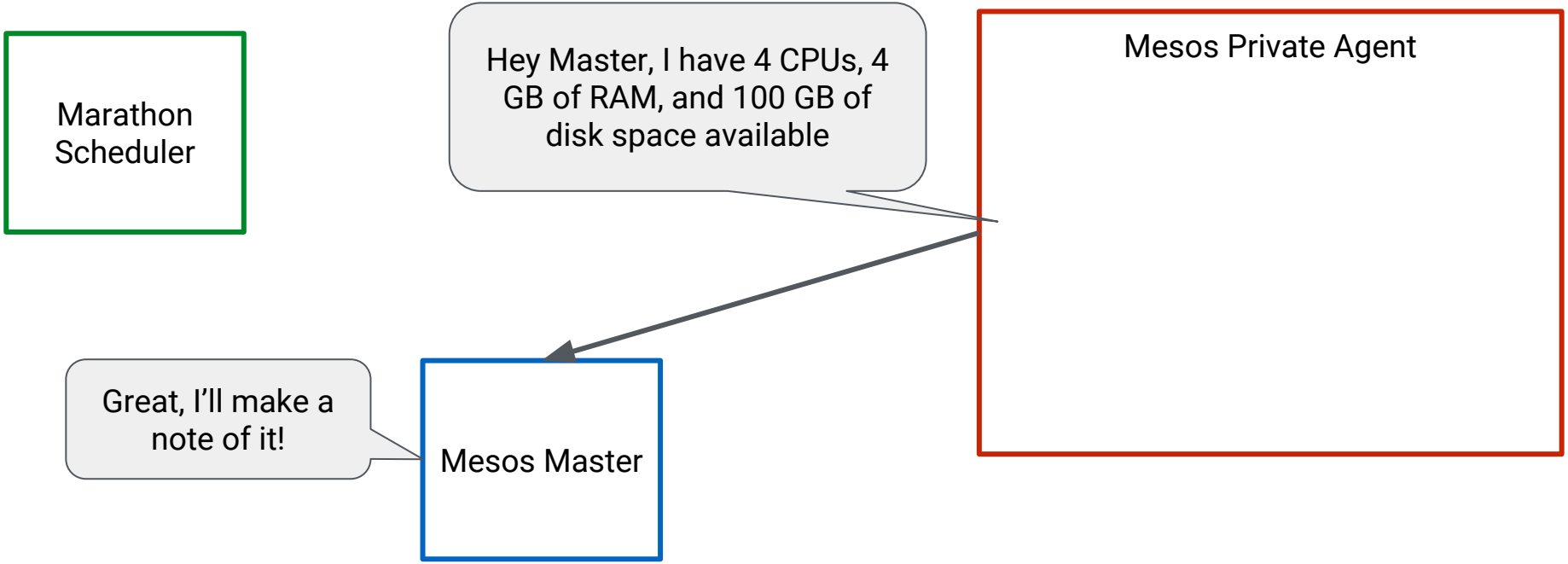



- Apache Mesos is the open source fundamental building block to DC/OS
- Apache project: [https://mesos.apache.org](https://mesos.apache.org)
- DC/OS 1.12 is packaged with Apache Mesos 1.5
- Acts as the kernel in DC/OS
- Function and operation in the following slides…

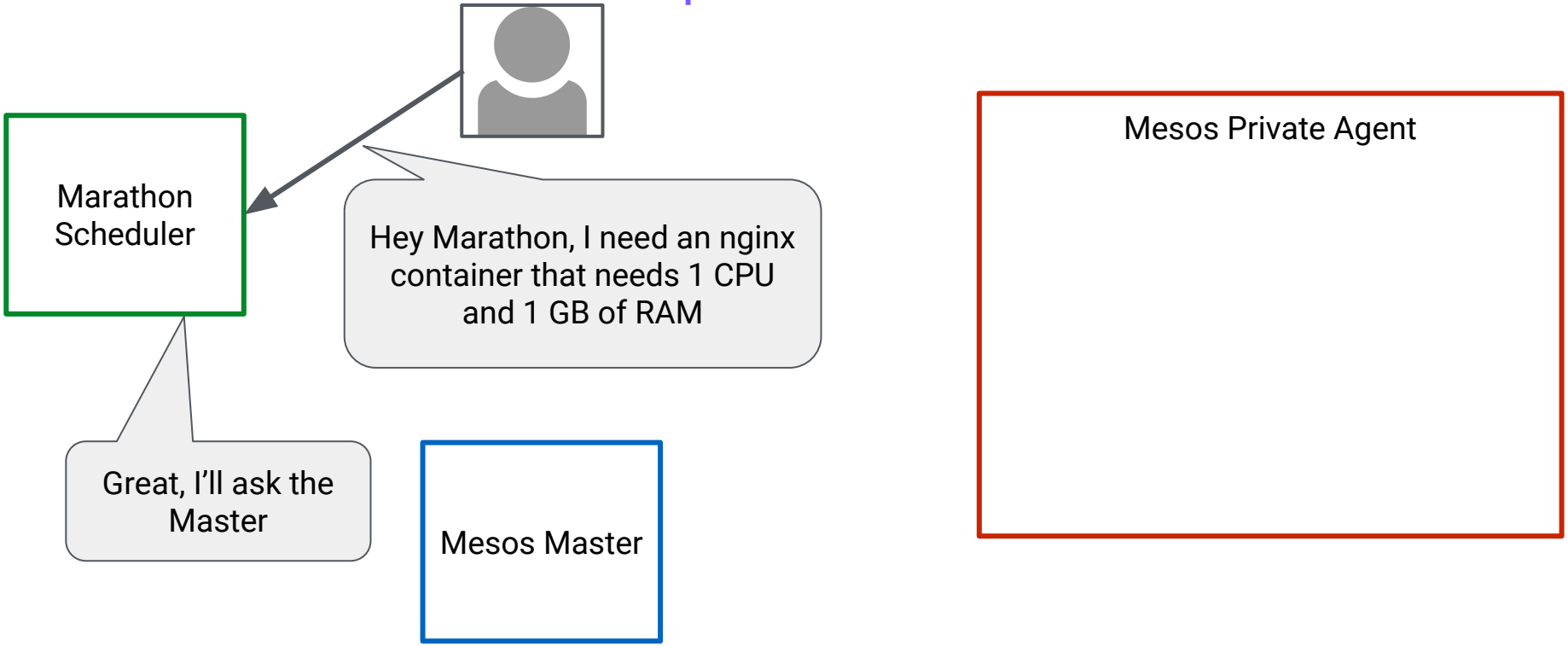
# Mesos Layer Diagram



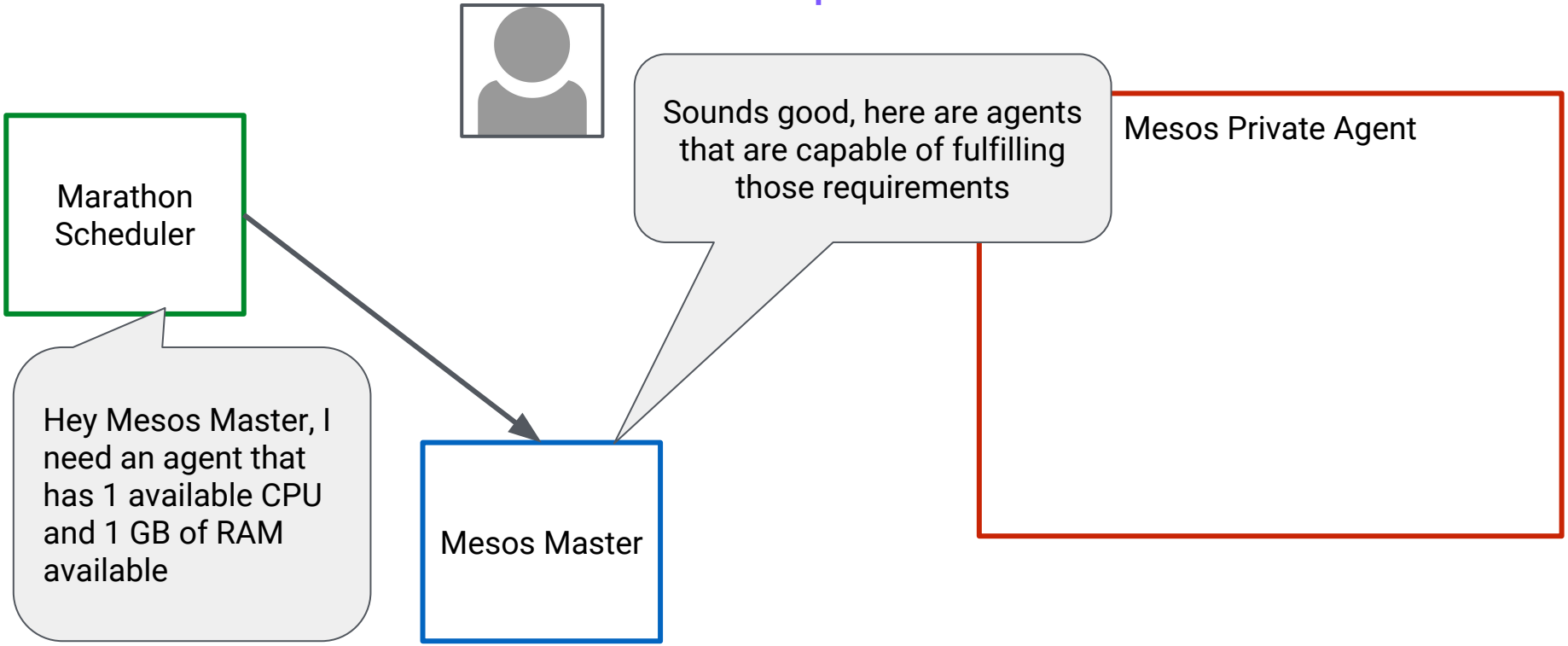
# Mesos in Action - Resource Offer



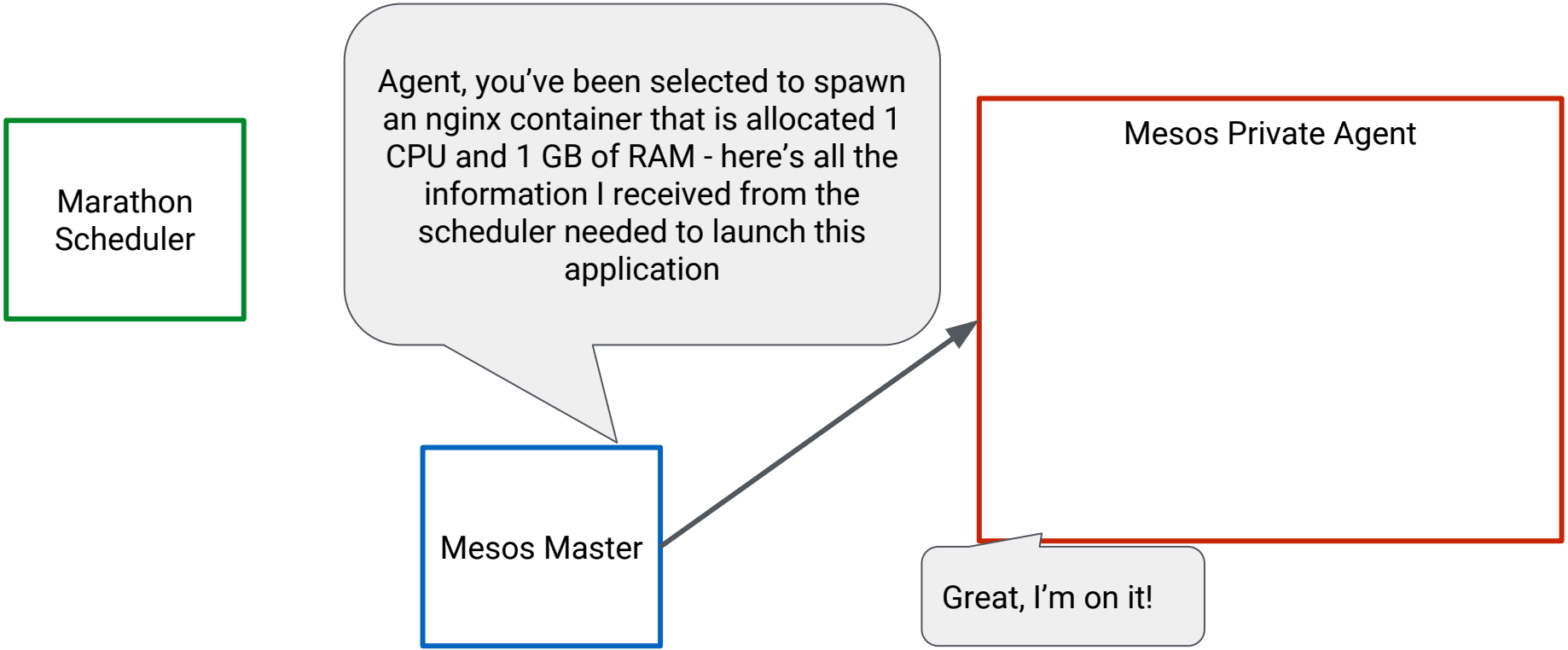
# Mesos in Action - User Request



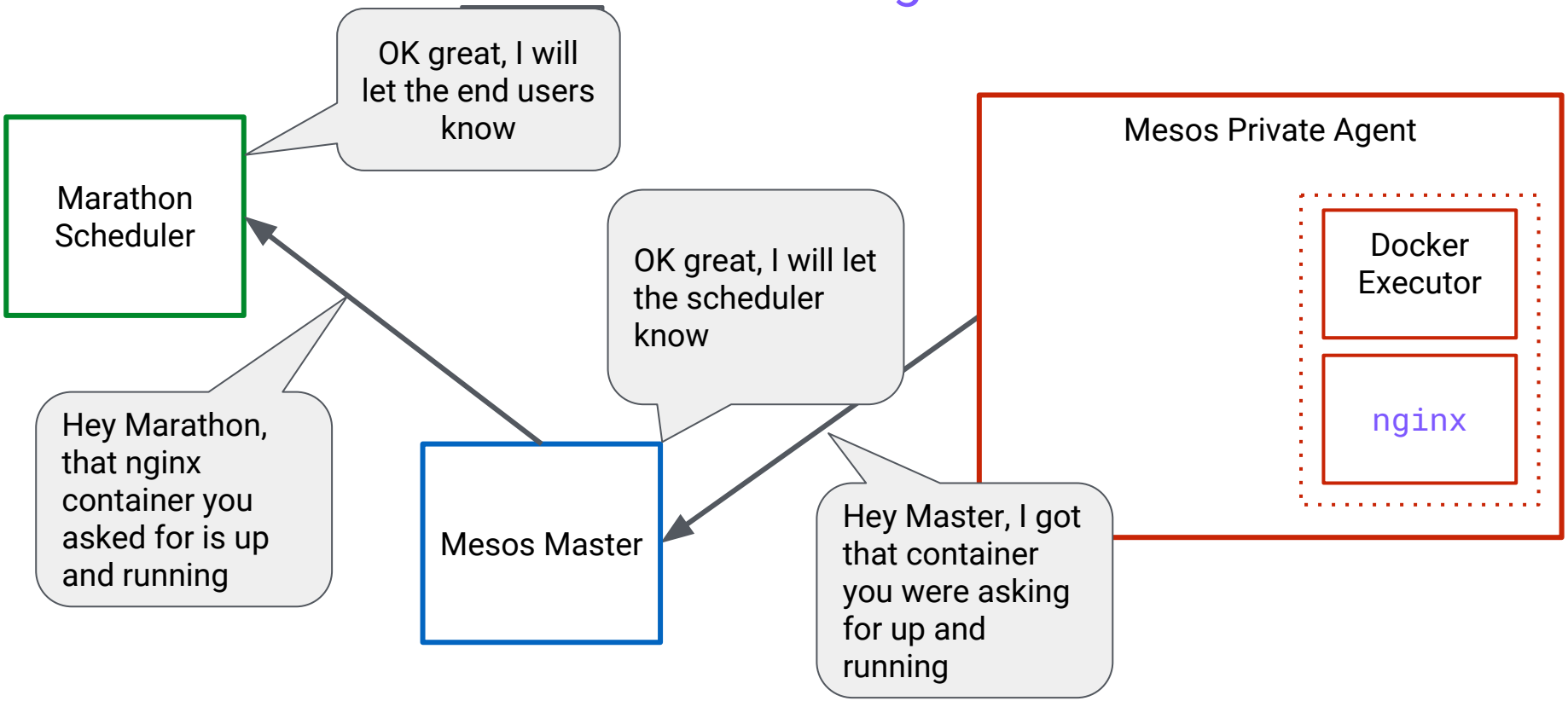
# Mesos in Action - Scheduler Request



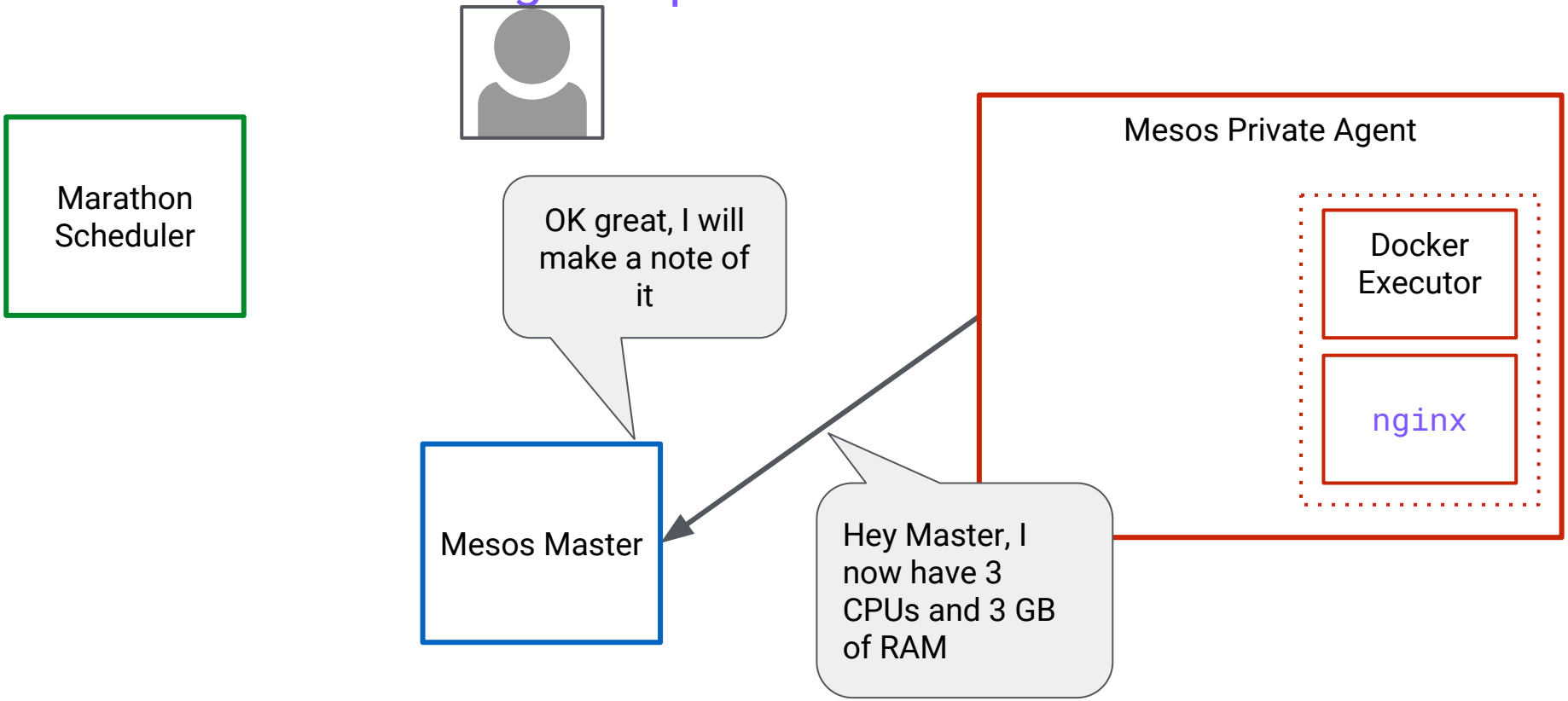
# Mesos in Action - Container Launch



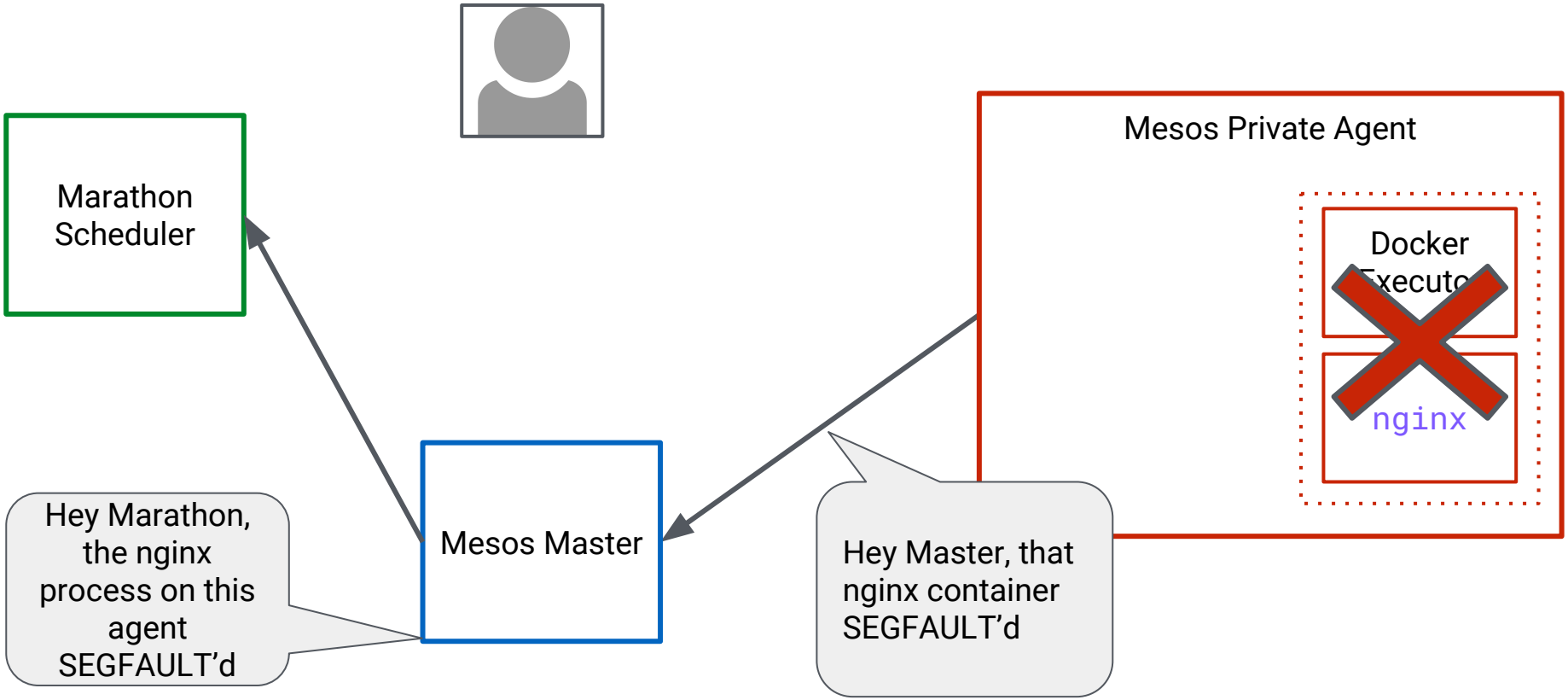
# Mesos in Action - Container Running



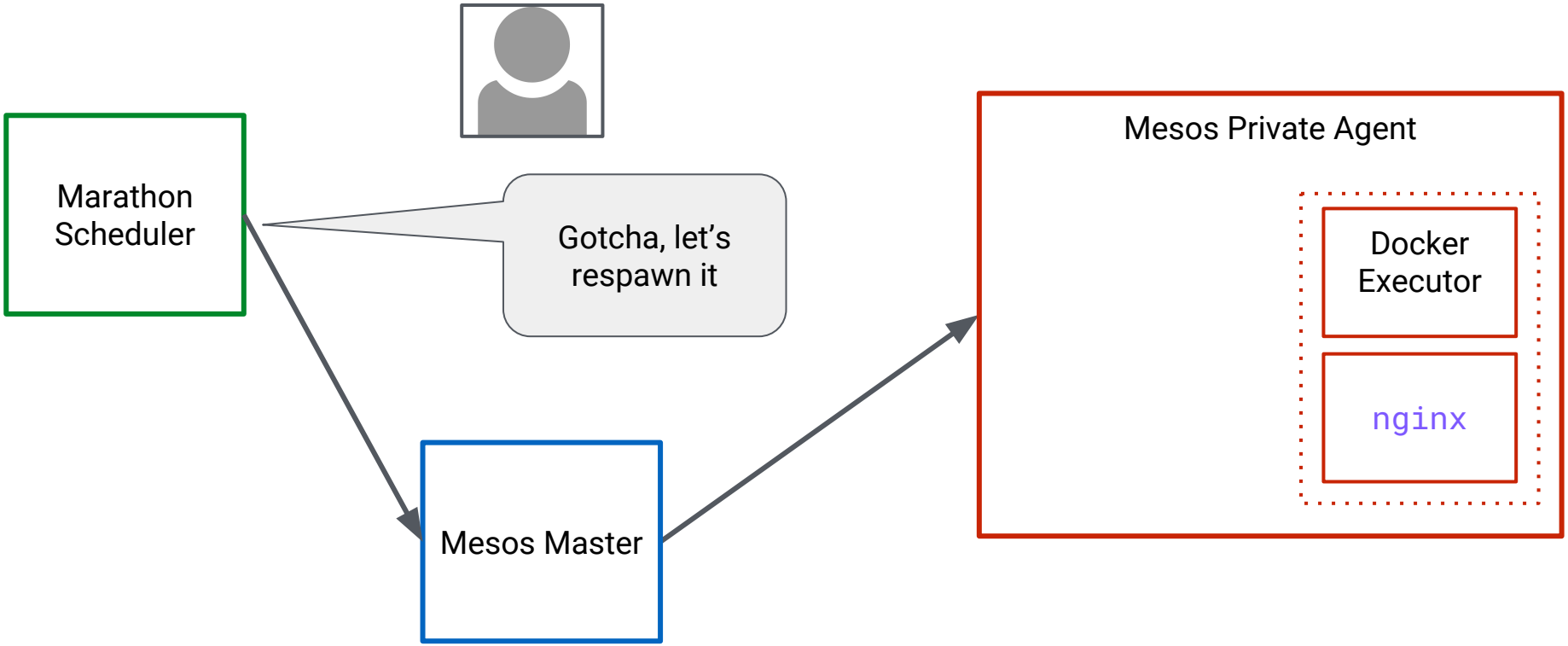
# Mesos in Action - Agent Update



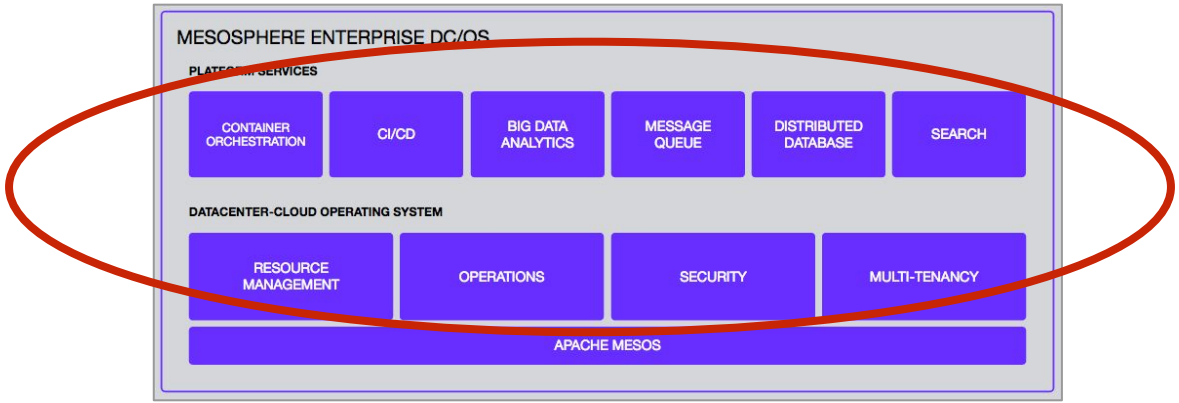
# Mesos in Action - Container Crash



# Mesos in Action - Container Crash



# DC/OS Layer




- DC/OS is built on top of Apache Mesos and provides a lot of “easy buttons” for Mesos
- Built-in container orchestration is provided through two schedulers for Mesos:
  - Marathon: Used to launch and manage long-running services
  - Metronome: Used to launch batch workloads (optionally on a schedule similar to cron)
- Task and networking interface, service discovery, load-balancing built-in
- GUI for ease of monitoring
- CLI for ease of automation
- Robust security and access control capabilities (Enterprise Edition only)

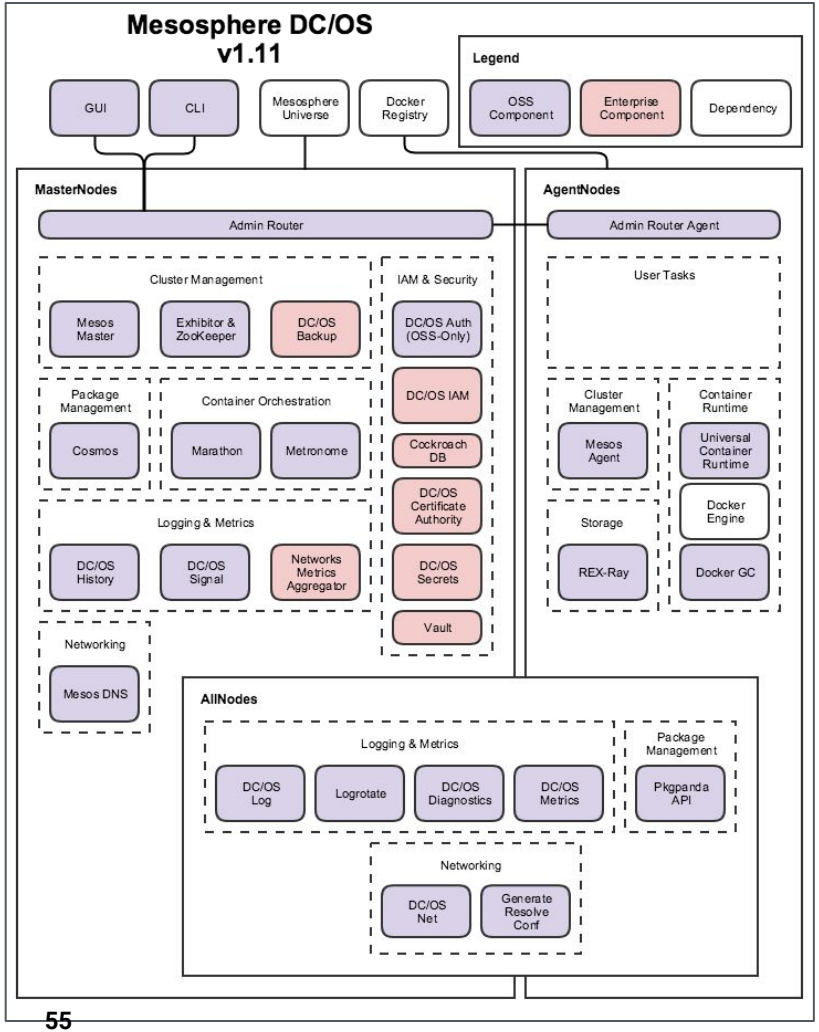

**DC/OS (OSS and Enterprise)**

- Resource Management
- Universal Container Runtime (UCR)
- Container Orchestration
- Pluggable Schedulers
- Support for Both Batch Jobs & Long Running Services
- Distributed System Services
- Virtual Networking and CNI
- Package Management
- Service Catalog & Application Ecosystem

**DC/OS (Enterprise Only)**

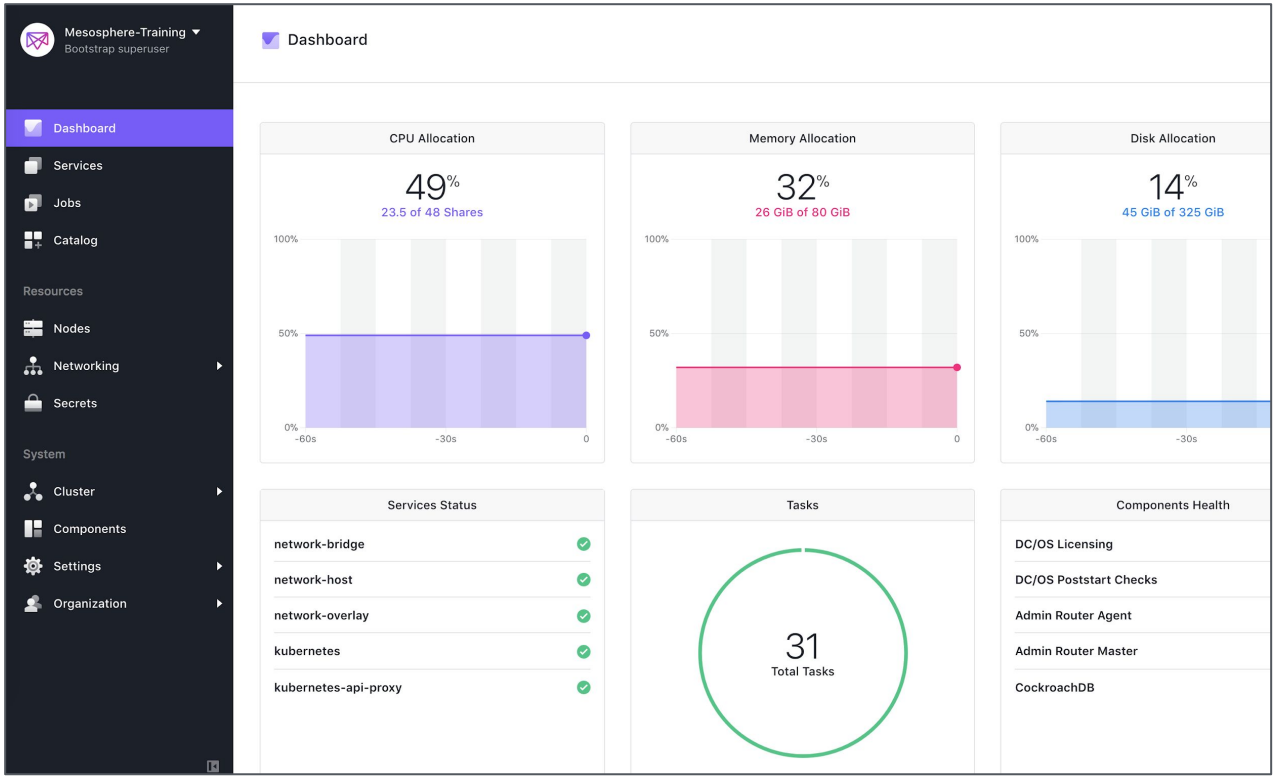
- E2E Encryption
- Identity & Access Management
- Secrets Management
- Support

# Architecture



# DC/OS GUI

# Main GUI Interface



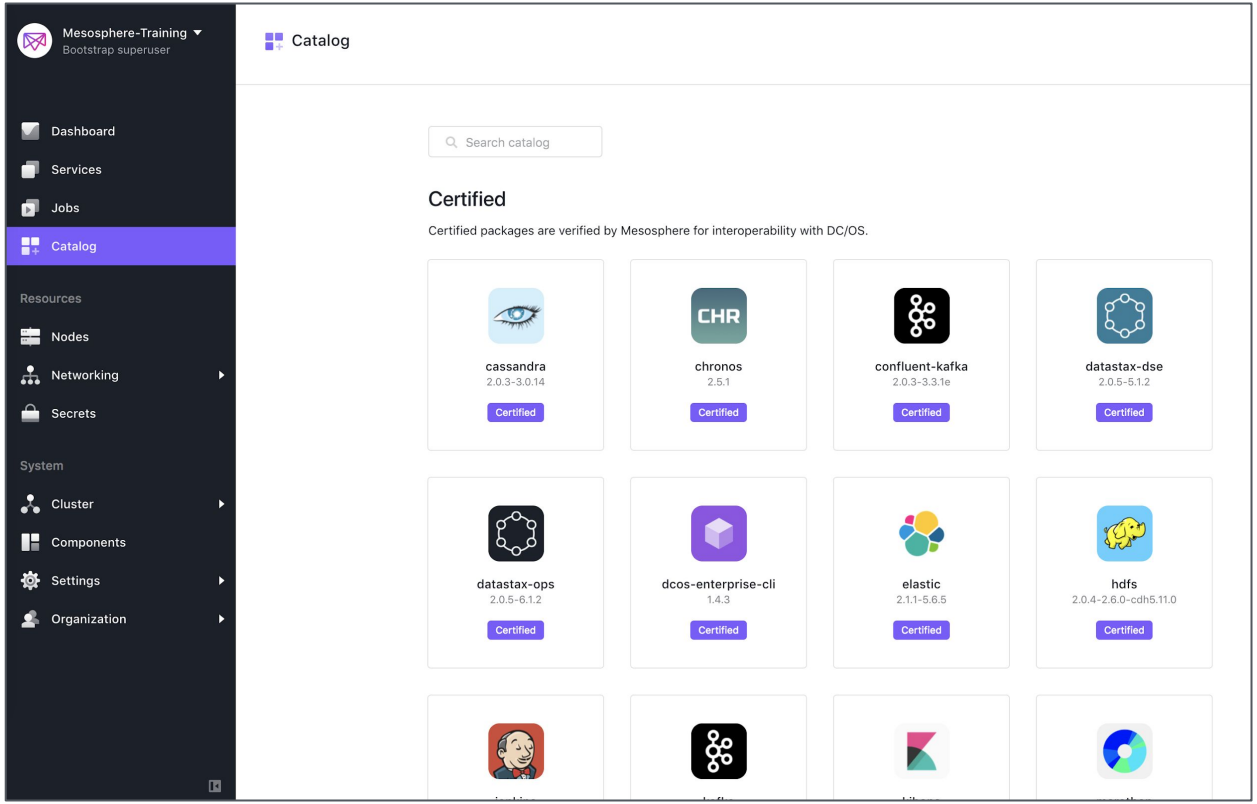
# Demo of DC/OS GUI

# Lab 2

## DC/OS GUI

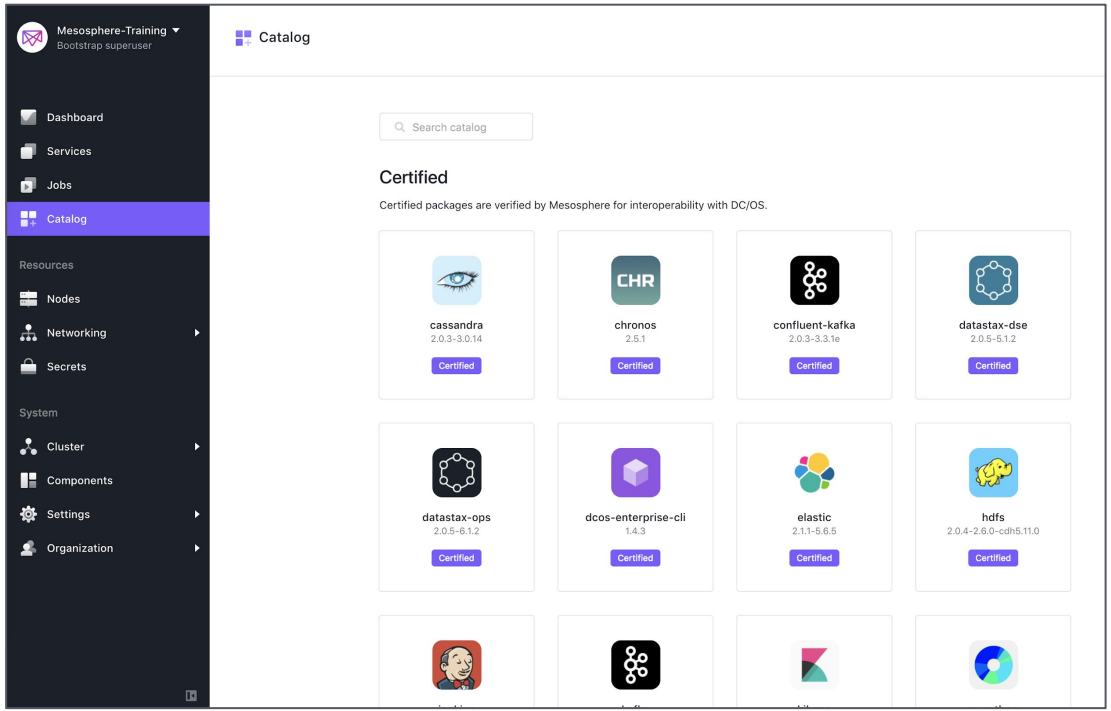
# Package Catalog

# Package Catalog

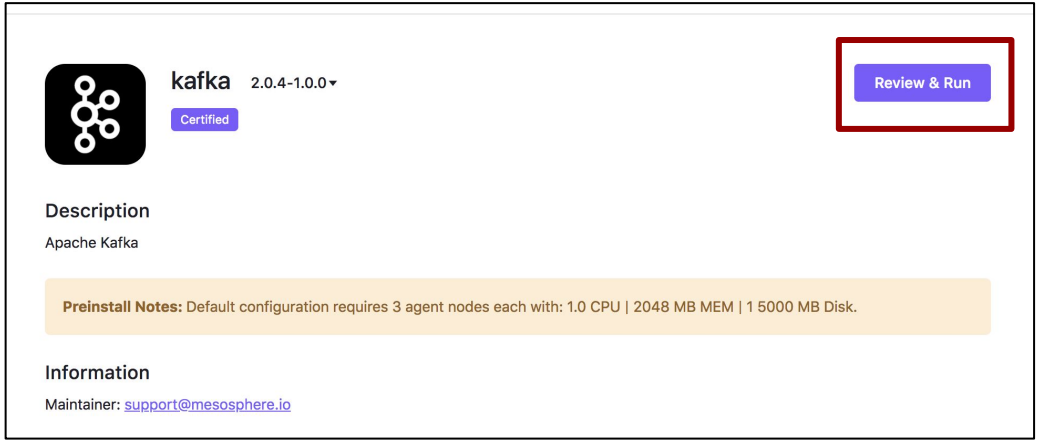


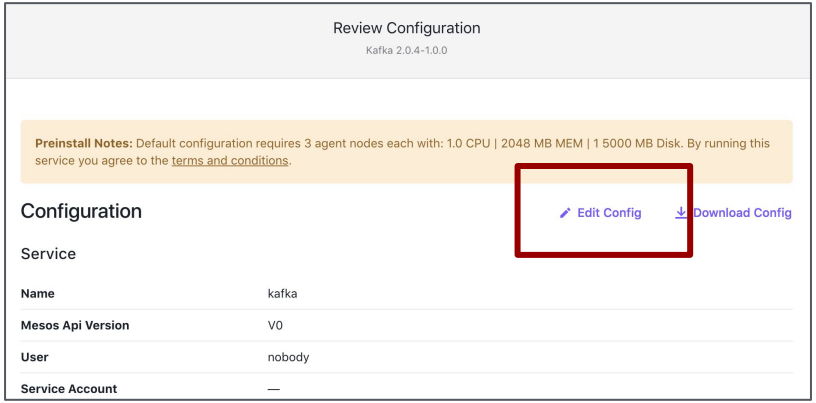
# Certified Catalog Packages

- Managed by Mesosphere
- Mesosphere maintained documentation
- Based on Mesosphere's SDK
- Additional, built-in features

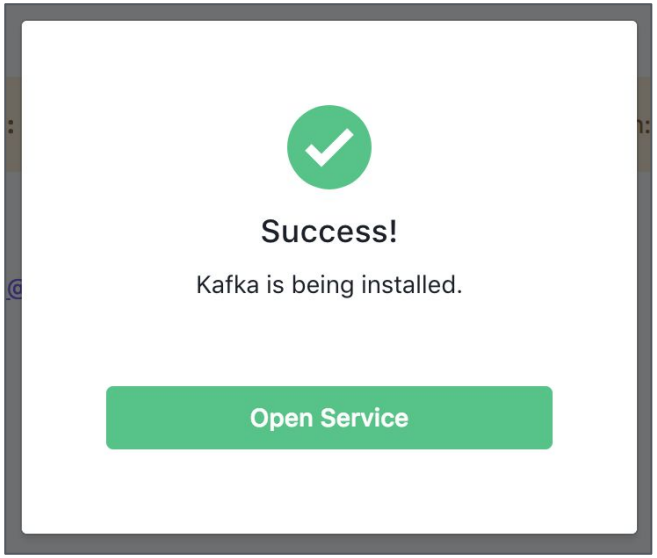


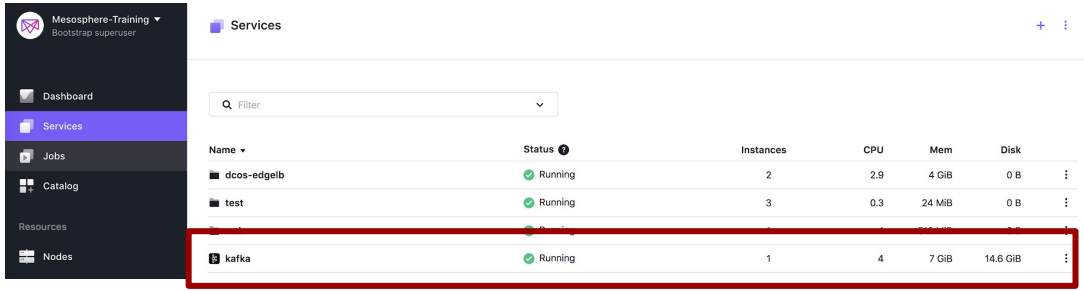
# Catalog - Installing and Configuring a Package



# Catalog - Package Installed



# Catalog - Custom/Private Repository

- It is possible to create and maintain a private Catalog for your DC/OS cluster
- Can be a copy of the entire public Catalog, or a subset thereof
- Possible to build a Catalog made up of packages that use a set of proprietary container images
- Documentation

# Lab 3

Install Packages from the DC/OS GUI

# DC/OS CLI

# DC/OS CLI

```
$ dcos

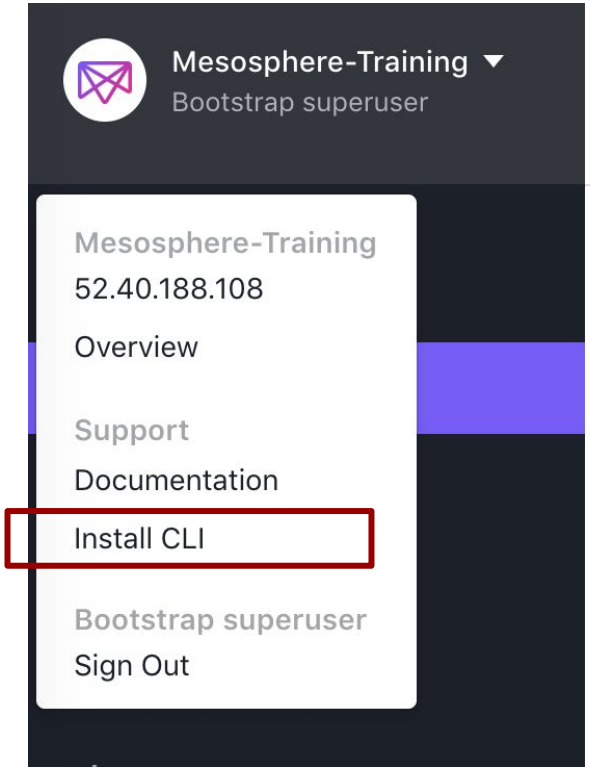
Command line utility for the Mesosphere Datacenter Operating
System (DC/OS). The Mesosphere DC/OS is a distributed operating
system built around Apache Mesos. This utility provides tools for
easy management of a DC/OS installation.

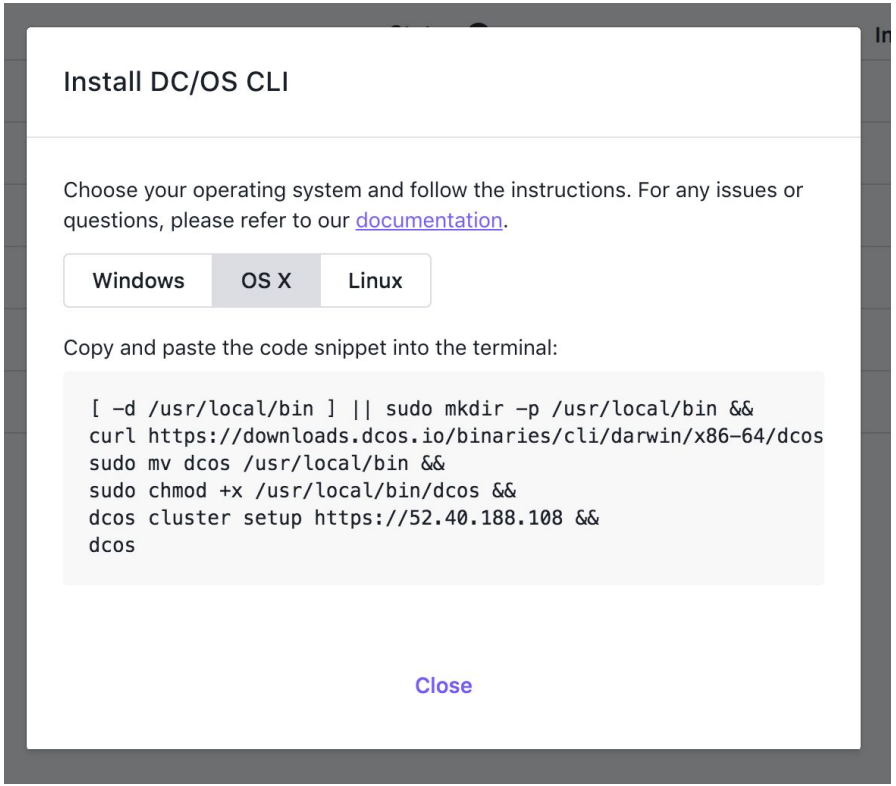
Available DC/OS commands:

      auth            Authenticate to DC/OS cluster
      cluster         Manage your DC/OS clusters
      config          Manage the DC/OS configuration file
      help            Display help information about DC/OS
      job             Deploy and manage jobs in DC/OS
      license         Manage your DC/OS licenses
      marathon        Deploy and manage applications to DC/OS
      node            View DC/OS node information
      package         Install and manage DC/OS software packages
      service         Manage DC/OS services
      task            Manage DC/OS tasks

Get detailed command description with 'dcos <command> --help'.
```

# Installation



# DC/OS CLI: Login to Enterprise

```
$ dcos cluster setup http(s)://<master_ip>
10.0.0.50's username: <user>
<user>@10.0.0.50's password: <password>
Login successful!
```

```
$ dcos auth login
10.0.0.50's username: <user>
<user>@10.0.0.50's password: <password>
Login successful!
```

# DC/OS CLI: Usage

```
$ dcos package install mysql
By Deploying, you agree to the Terms and Conditions
https://mesosphere.com/catalog-terms-conditions/#commun
ity-services

This DC/OS Service is currently in preview. There may
be bugs, incomplete features, incorrect documentation,
or other discrepancies.


```Advanced Installation options notes```

storage / *persistence*: create local persistent
volumes for internal storage files to survive across
restarts or failures.
```
```

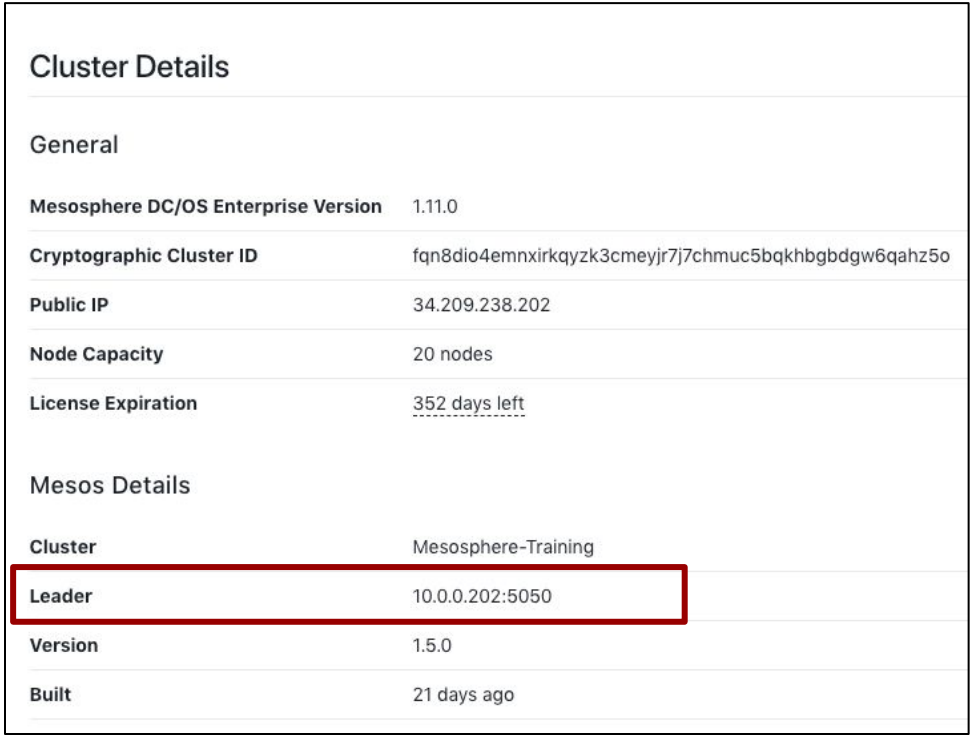
# Lab 4

The DC/OS CLI

# SSH Connectivity

# SSH Connectivity

- Sometimes you need direct access to your cluster nodes (masters, private agents, public agents)
- SSH in to the leading Mesos master node - you can then use that as a jumpbox to the other nodes in your cluster
- You can figure out which system is your leading Mesos master node from within the DC/OS GUI



# SSH Connectivity with `dcos` CLI

The `dcos` CLI has built-in capabilities to simplify how to connect to a cluster node over SSH

Examples:

1. To SSH to your leading master node:
   `dcos node ssh --leader --master-proxy --user=<user>`
2. To SSH to a private agent, first retrieve its ID: `dcos node`
3. Initiate SSH connection:
   `dcos node ssh --master-proxy --user=<user> --mesos-id=<agent_id>`

# SSH Connectivity from a Linux System

Follow instructions in the [documentation]
High-level overview:

```
$ ssh-copy-id <user>@<host>
$ chmod 600 <private_key>
$ ssh-add <private_key>
$ dcos node ssh --master-proxy --leader --user <user>
$ dcos node
$ dcos node ssh --master-proxy --user <user> --mesos-id <agent_id>
```

# SSH Connectivity from a Windows System

- Requires installation of an SSH client (e.g. PuTTY)
- Load your private key into the SSH client
  - For PuTTY: Convert your `.pem` file to a `.ppk` by using PuTTYgen
- To SSH in to a master node:
  - Specify hostname or IP address
  - Set auth to use `.ppk`
  - From the master, you can SSH in to agents
- Documentation

# Summary: Day One

**In this module we covered:**

- DC/OS Overview
- DC/OS Installation
- Accessing your cluster with the DC/OS GUI
- Accessing your cluster with the DC/OS CLI
- Connecting to your cluster nodes with SSH

MESOSPHERE