Sunil Shah

AGILE DEVELOPMENT AND PAAS USING THE MESOSPHERE DCOS

MESOSPHERE

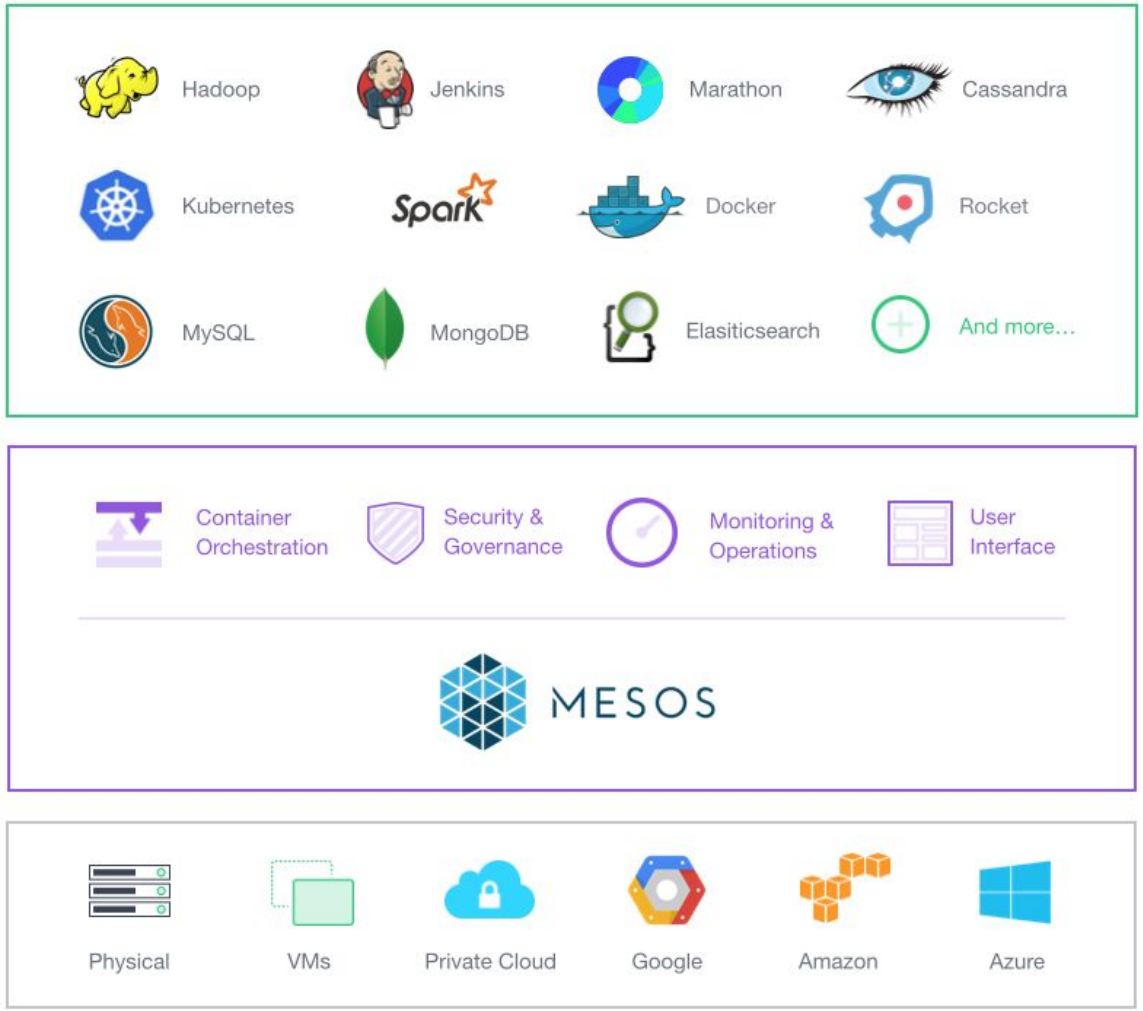
THE DATACENTER OPERATING SYSTEM (DCOS)

DCOS

INTRODUCTION

The Mesosphere Datacenter Operating System (DCOS) is a distributed operating system that spans all of the machines in your datacenter or cloud.

It provides a highly elastic, and highly scalable way of deploying applications, services and big data infrastructure on shared resources.



Services & Containers

Mesosphere DCOS

Existing Infrastructure

DCOS

KEY COMPONENTS

The DCOS is built around some key components:

Apache Mesos

- Datacenter *kernel*

Marathon

- Datacenter *init.d*

Docker

- *Executable format*

Universe

- DCOS *package repository*

DCOS CLI

- *Command line* to your datacenter

All of these components are open source!

DCOS AND PAASTA

Problem	PaaSTA	DCOS
Code containerizer	*Docker*	*Docker*
Scheduling	*Mesos + Marathon*	*Mesos + Marathon*
Service Discovery	SmartStack	Mesos DNS, Marathon-lb
Monitoring	Sensu	DCOS
Workflow	Jenkins or CLI + soa-configs	Jenkins, DCOS CLI

MODERN INFRASTRUCTURE

MODERN INFRASTRUCTURE

CLEAN SEPARATION

Before

- Dan cares about his hardware and Alice's software that runs on it
- Alice cares about her software and what hardware Dan provides

Now

- With DCOS, all the nodes are provisioned exactly the same (but may have heterogenous hardware).
- Dan doesn't care what software is deployed since applications are well encapsulated.
- Alice doesn't care where her software is deployed because it's easy enough to scale up and down.

MODERN INFRASTRUCTURE

NO MORE 3AM WAKE UPS

Before

- Dan had to react every time an application or machine went down.

Now

- Mesos and Marathon monitor running tasks.
- If a task fails or is lost (due to a machine going offline), Mesos communicates that to Marathon.
- Marathon restarts the application.
- Dan gets to sleep peacefully!

MODERN INFRASTRUCTURE

EASY PROGRAMMATIC DEPLOYMENT

Before

- Servers were handcrafted.
- Deploying new or updated software would require oversight and involvement from both Alice and Dan.

Now

- Dan provides Alice with her own instance of Marathon that makes it hard for her to take down someone else's application.
- Running applications are isolated from each other by Mesos.
- Marathon offers a nice API that allows Alice to easily deploy new versions safely.

APACHE MESOS

THE BIRTH OF MESOS

Spring 2009

CS262B

Ben Hindman, Andy Konwinski and Matei Zaharia create “Nexus” as their CS262B class project.

March 2010

TWITTER TECH TALK

The grad students working on Mesos give a tech talk at Twitter.

September 2010

MESOS PUBLISHED

Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center is published as a technical report.

December 2010

APACHE INCUBATION

Mesos enters the Apache Incubator.

INTRODUCTION

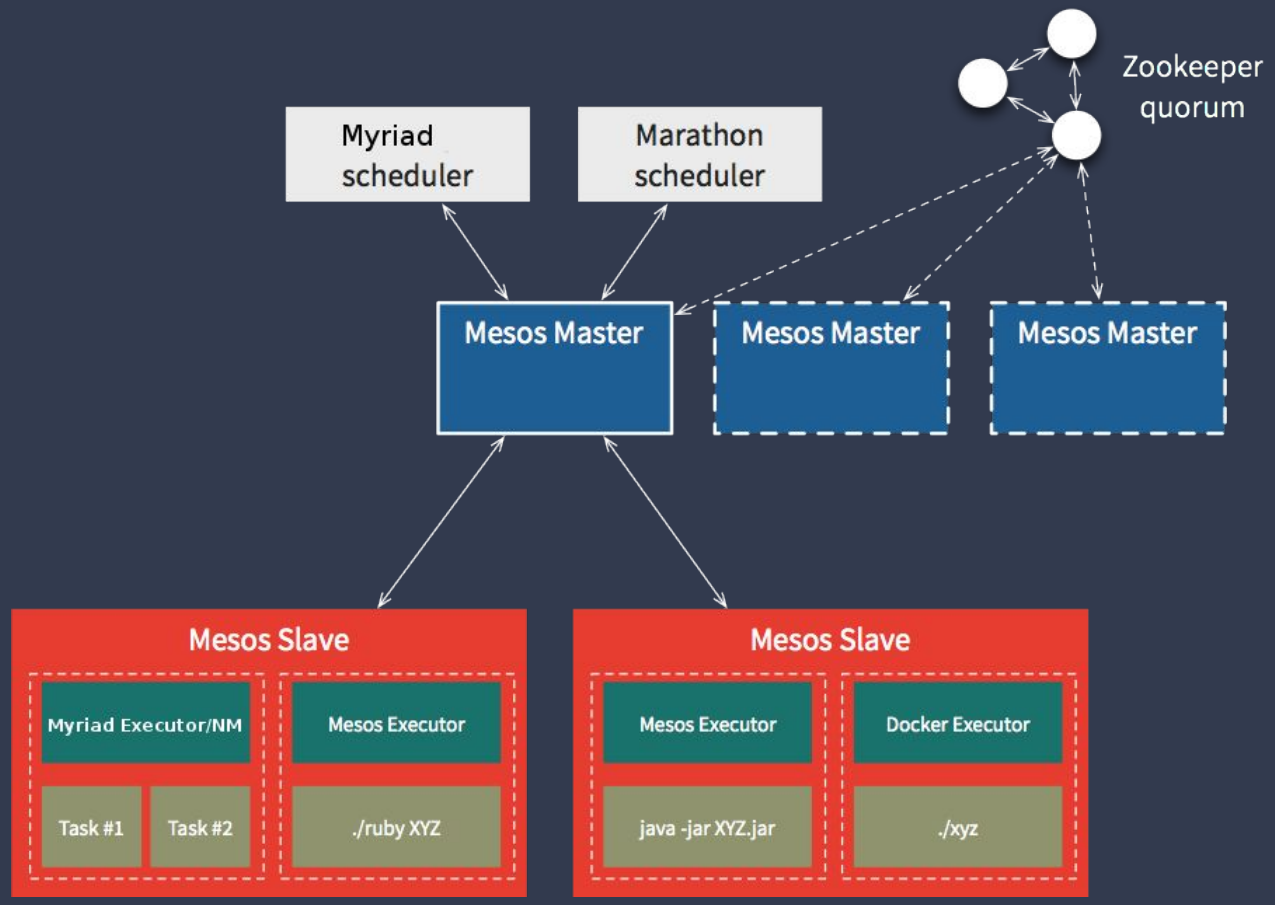
Apache Mesos is a **cluster resource manager**.

It handles:

- **Aggregating resources** and **offering them to schedulers**
- **Launching tasks** (i.e. processes) on those resources
- **Communicating the state of those tasks** back to schedulers
- Tasks can be:
  - Long running services
  - Ephemeral / batch jobs

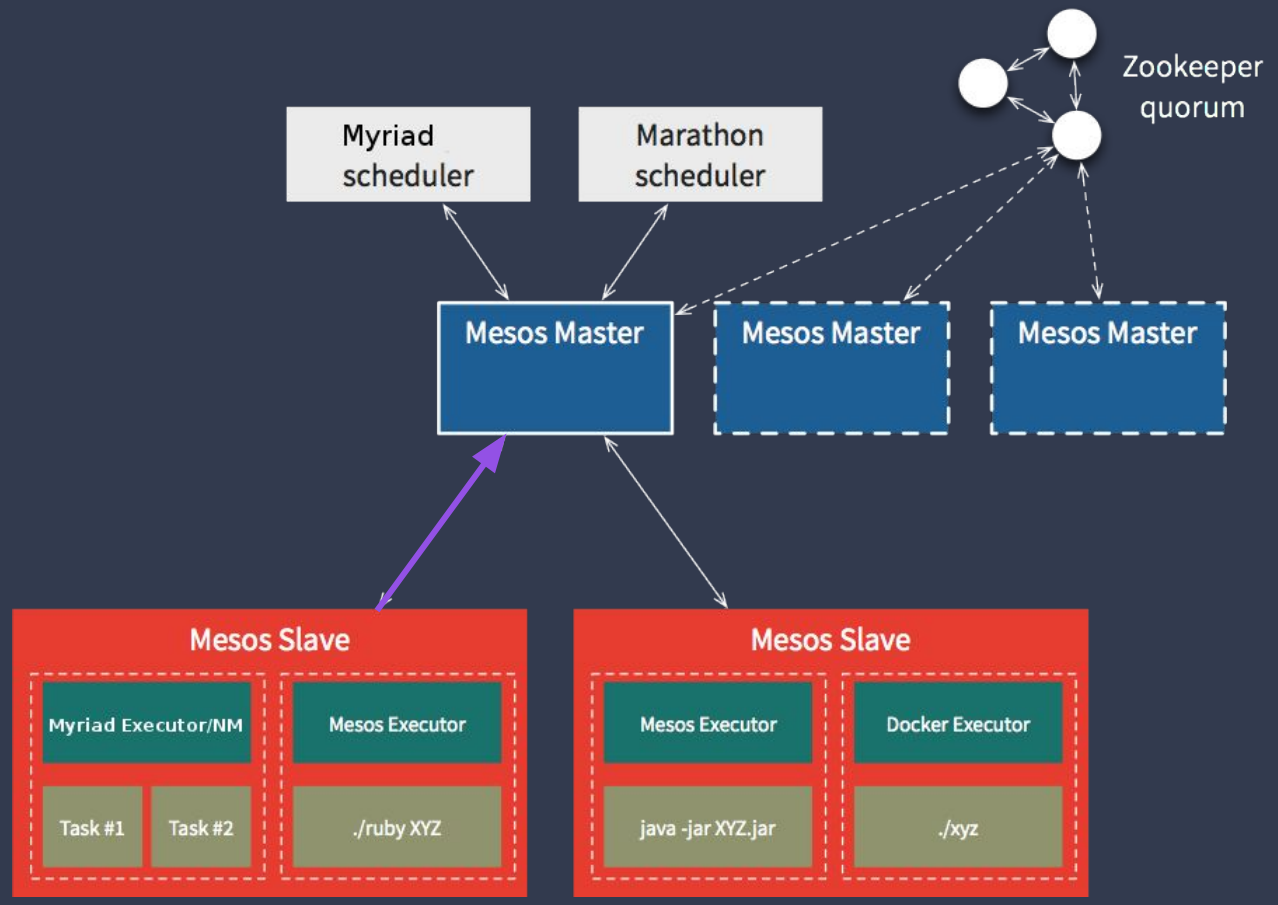
MESOS FUNDAMENTALS

ARCHITECTURE



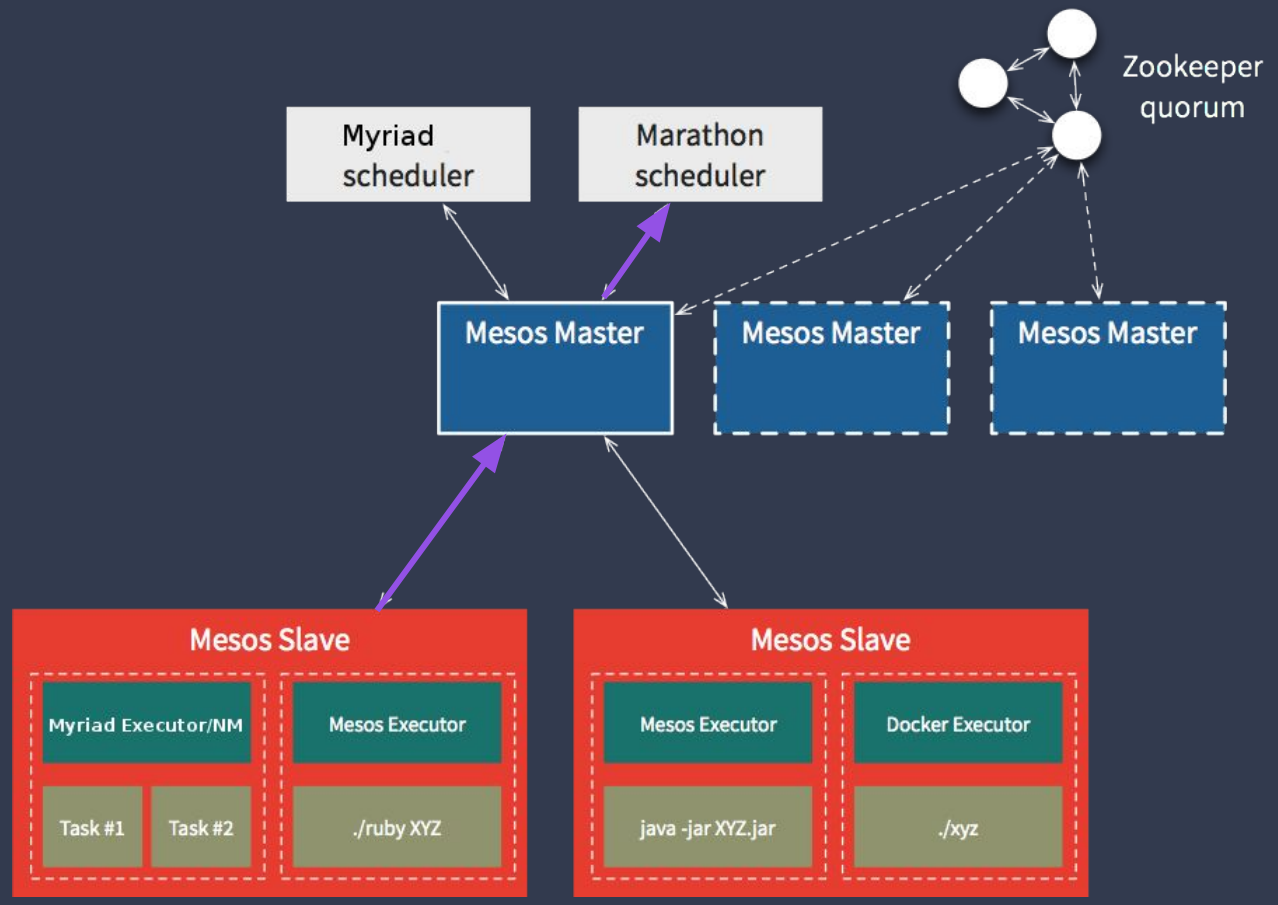
ARCHITECTURE

- Agents advertise resources to Master
- Master offers resources to Scheduler
- Scheduler rejects/uses resources
- Agents report task status to Master



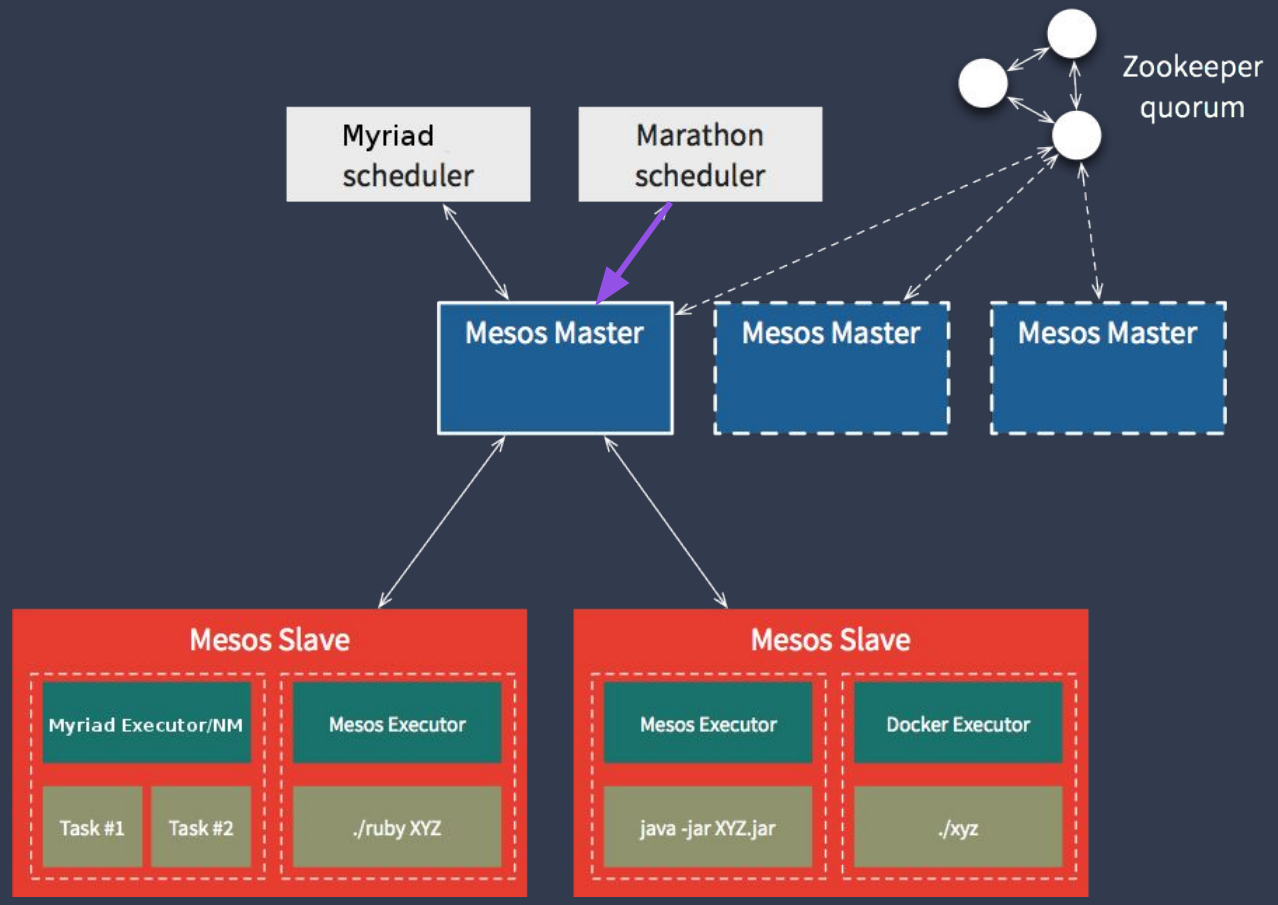
ARCHITECTURE

- Agents advertise resources to Master
- Master offers resources to Scheduler
- Scheduler rejects/uses resources
- Agents report task status to Master



ARCHITECTURE

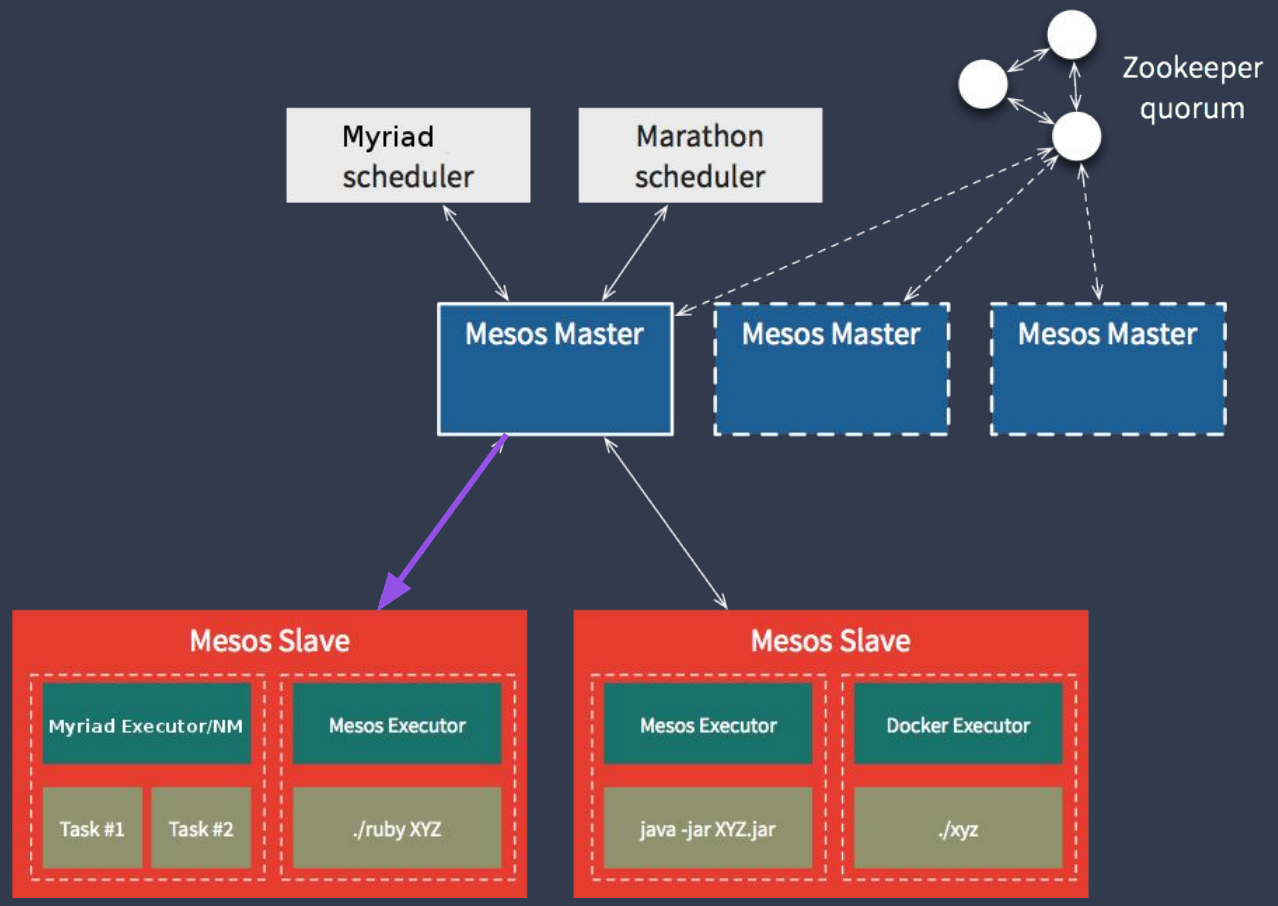
- Agents advertise resources to Master
- Master offers resources to Scheduler
- Scheduler rejects/uses resources
- Agents report task status to Master



MESOS FUNDAMENTALS

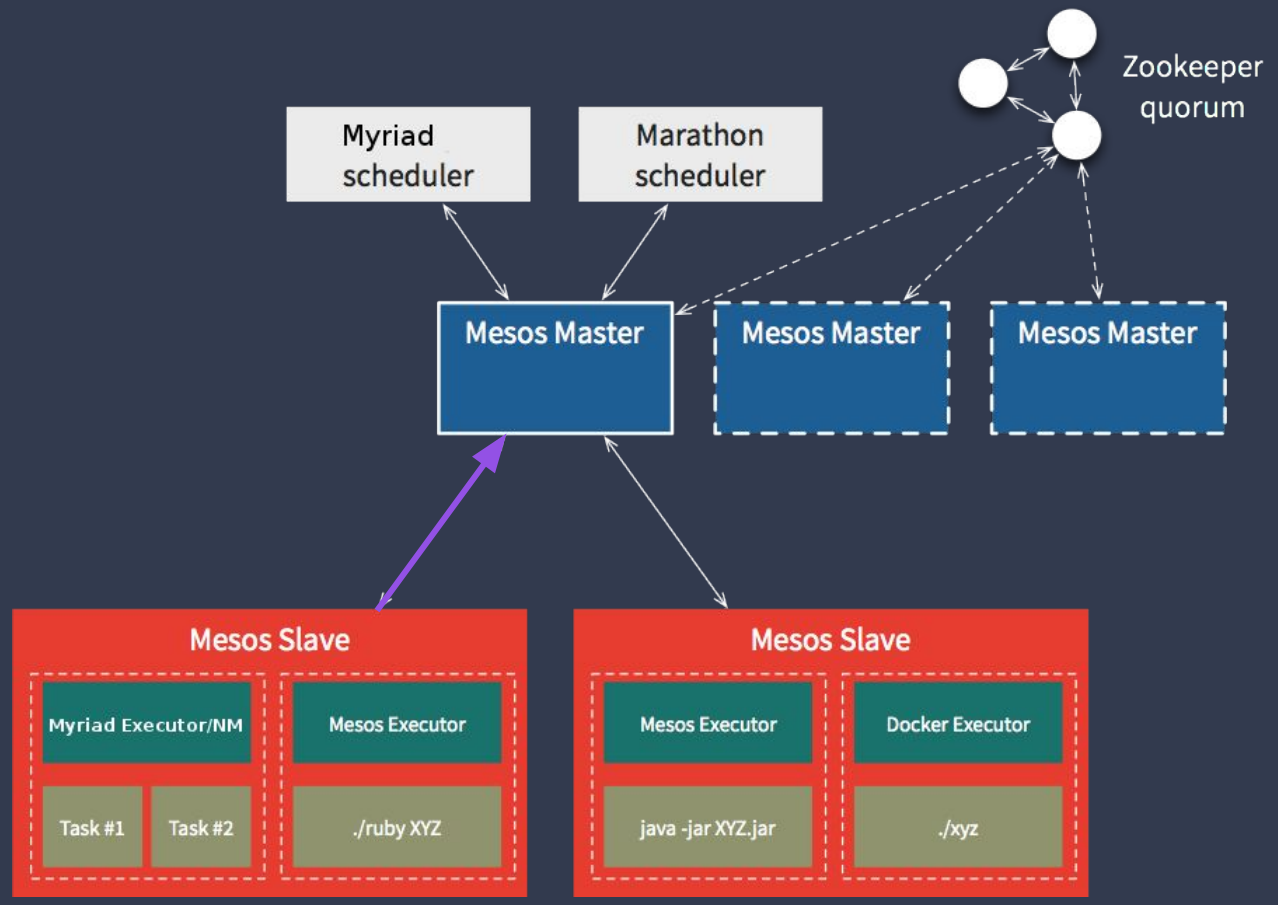
ARCHITECTURE

- Agents advertise resources to Master
- Master offers resources to Scheduler
- Scheduler rejects/uses resources
- Agents report task status to Master

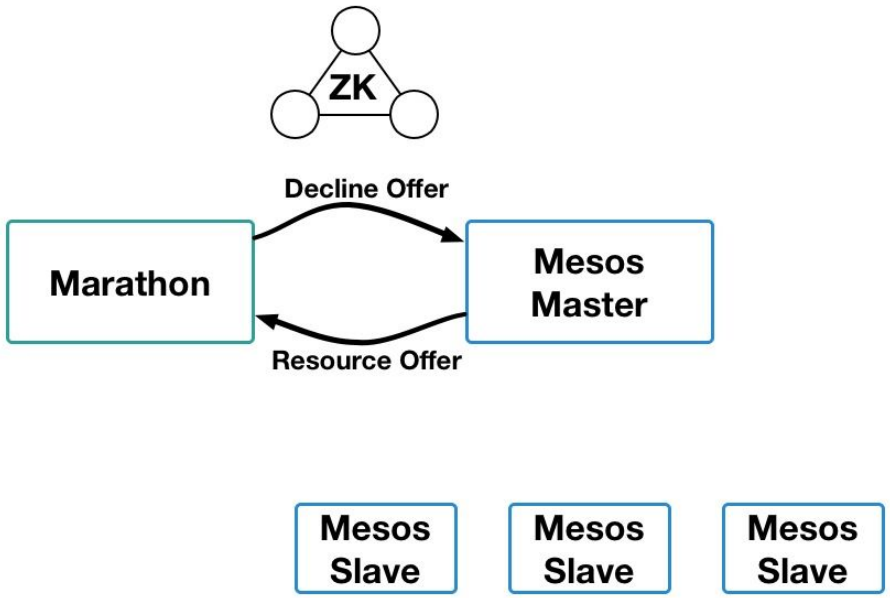


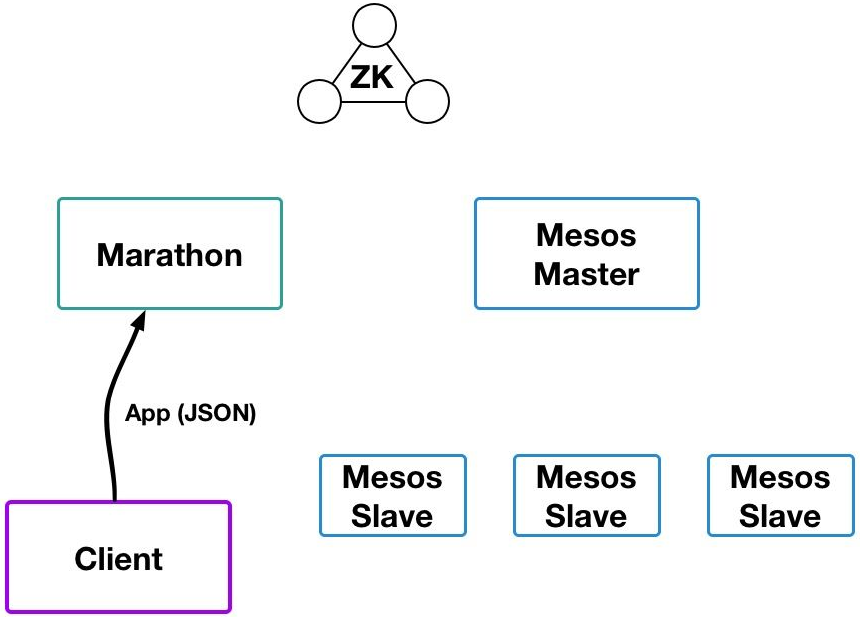
ARCHITECTURE

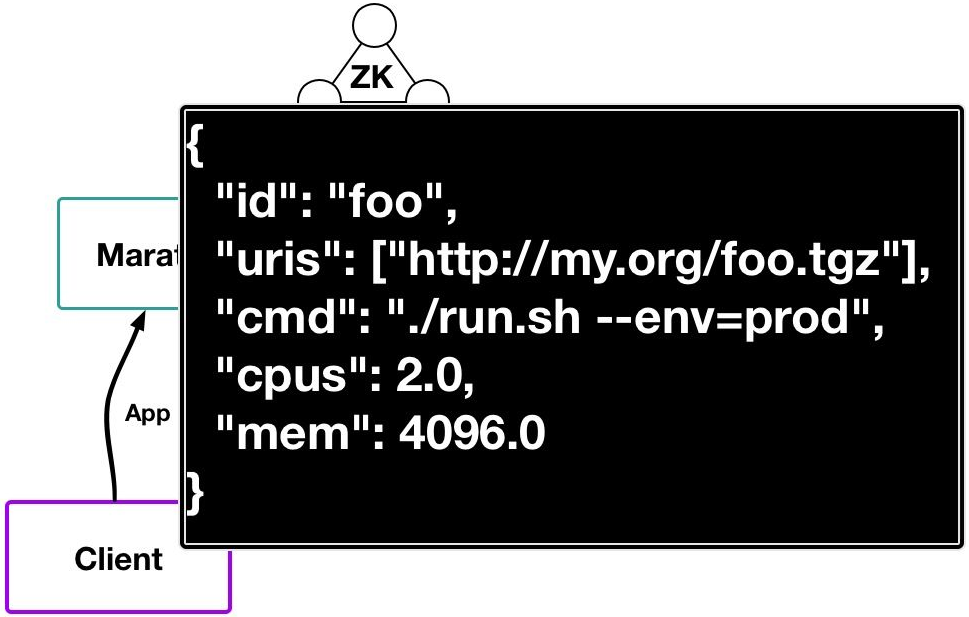
- Agents advertise resources to Master
- Master offers resources to Scheduler
- Scheduler rejects/uses resources
- Agents report task status to Master

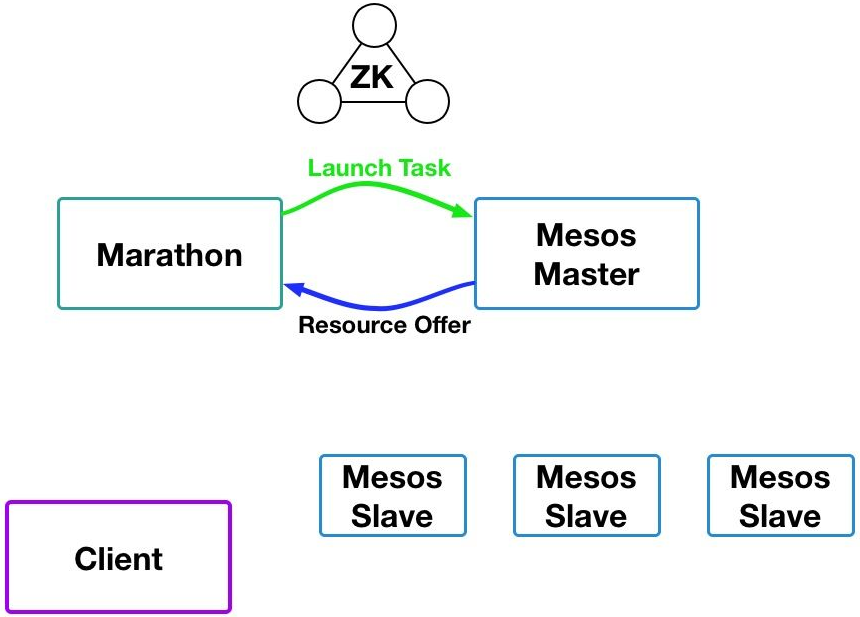


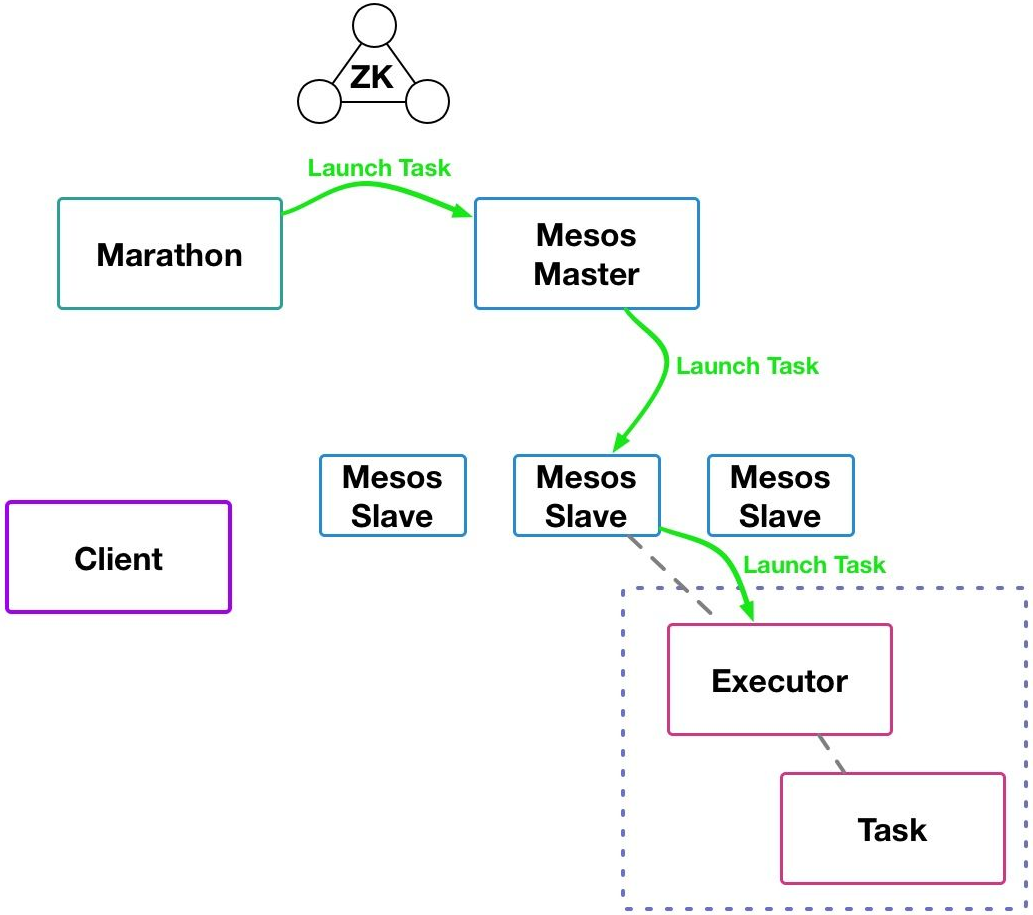
MESOS AND MARATHON EXAMPLE

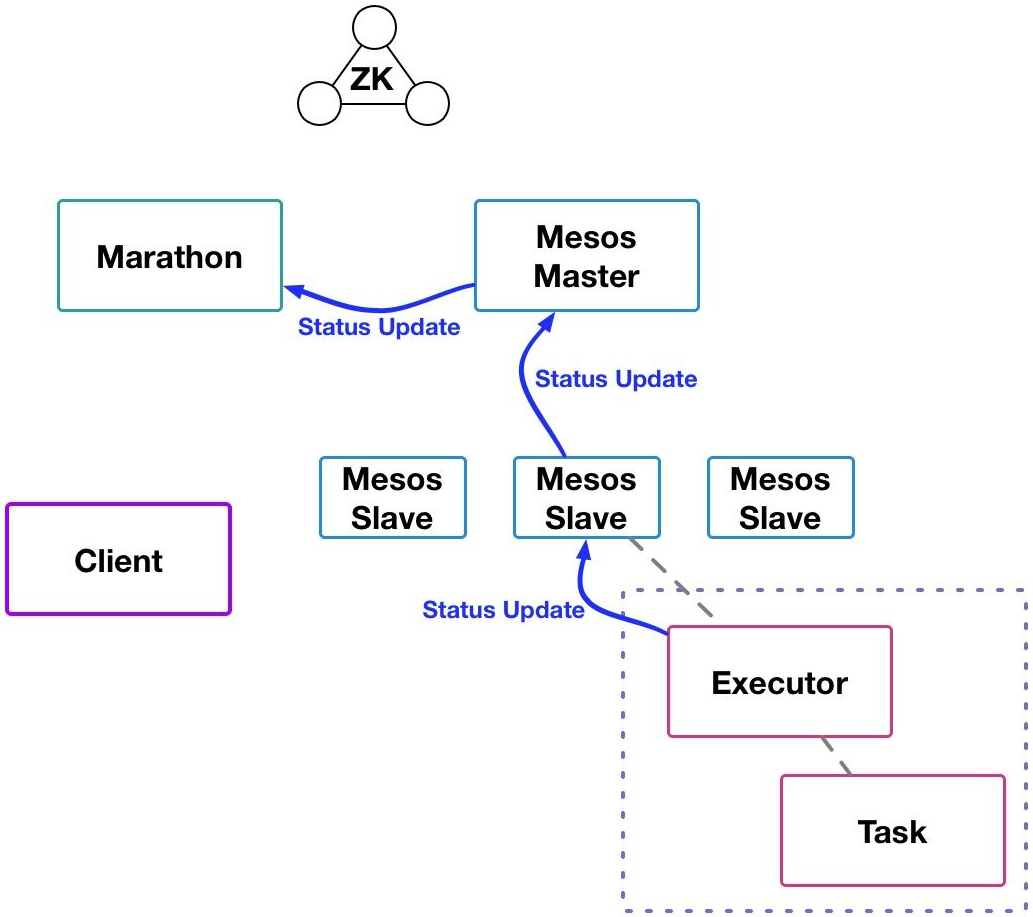


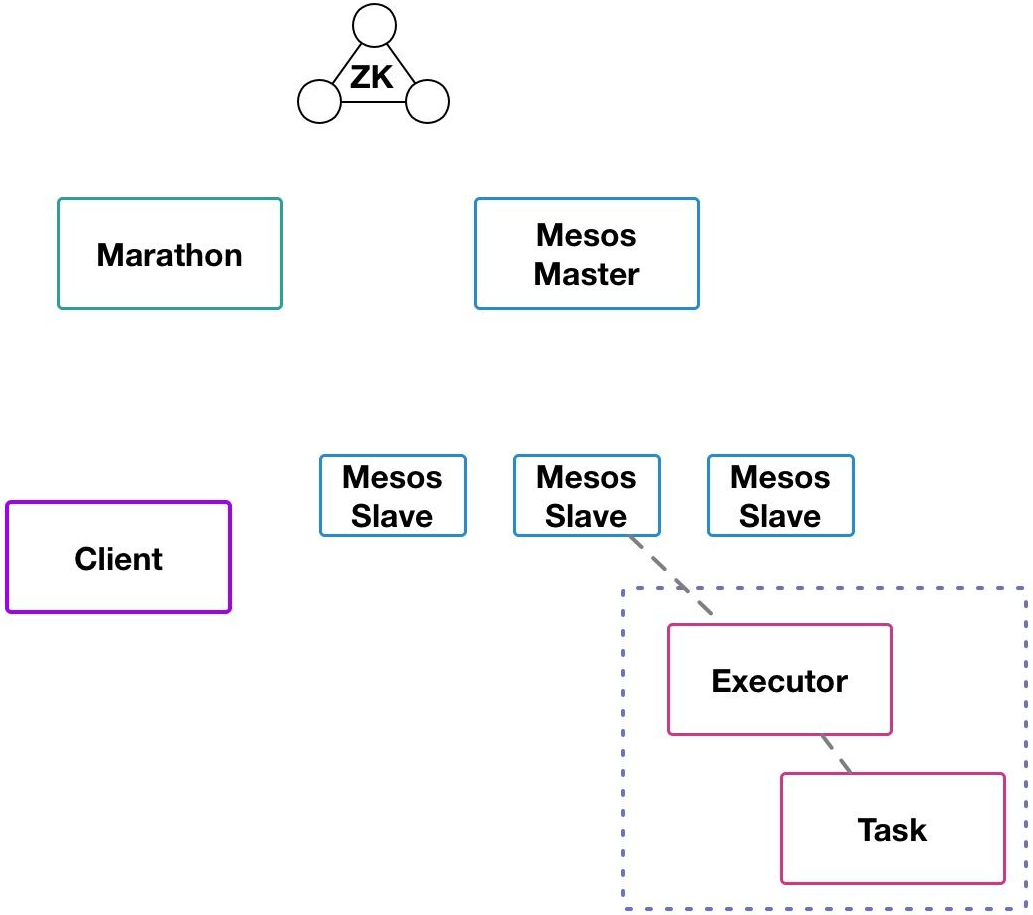






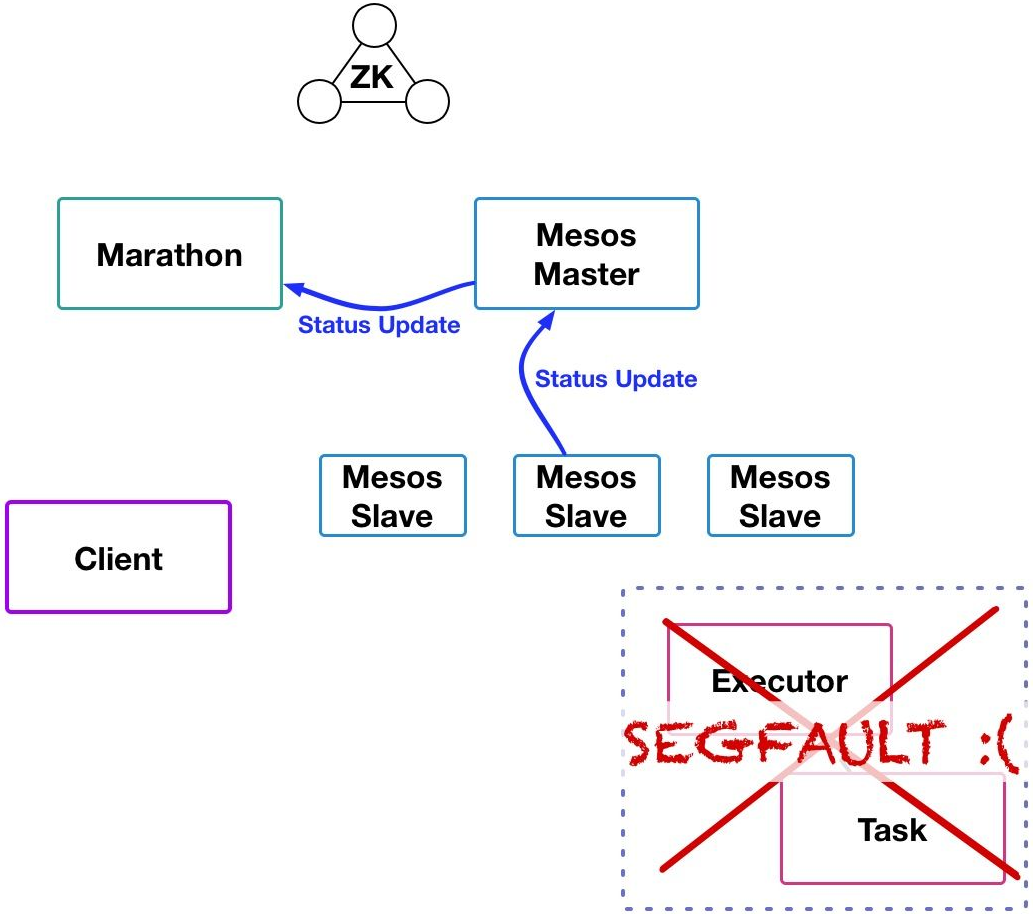


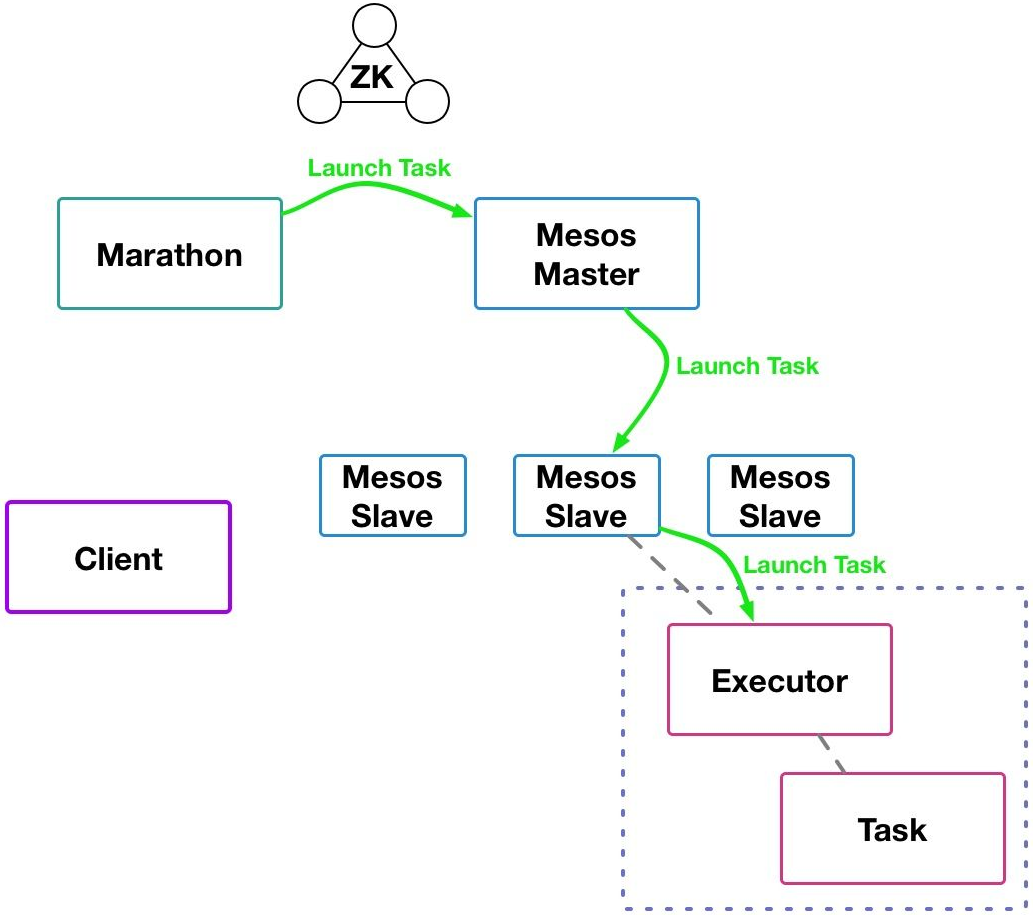


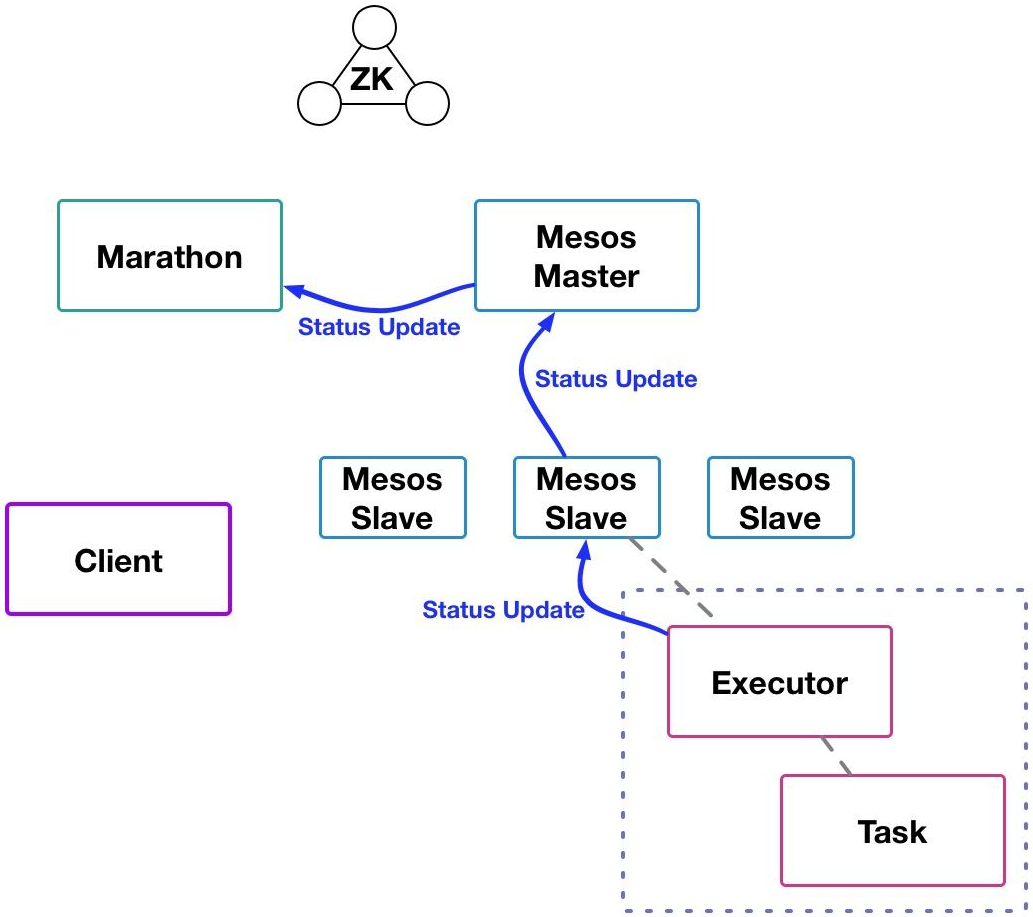


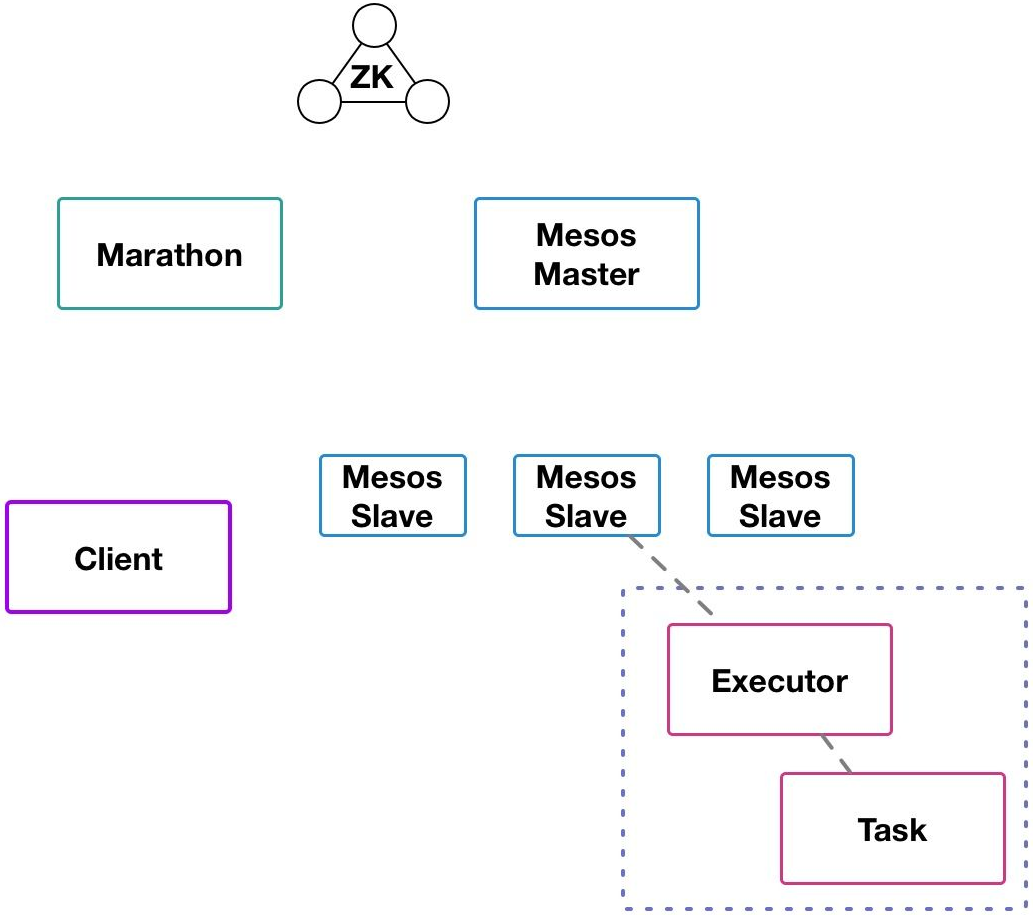
MESOS AND MARATHON EXAMPLE

TASK FAILURE





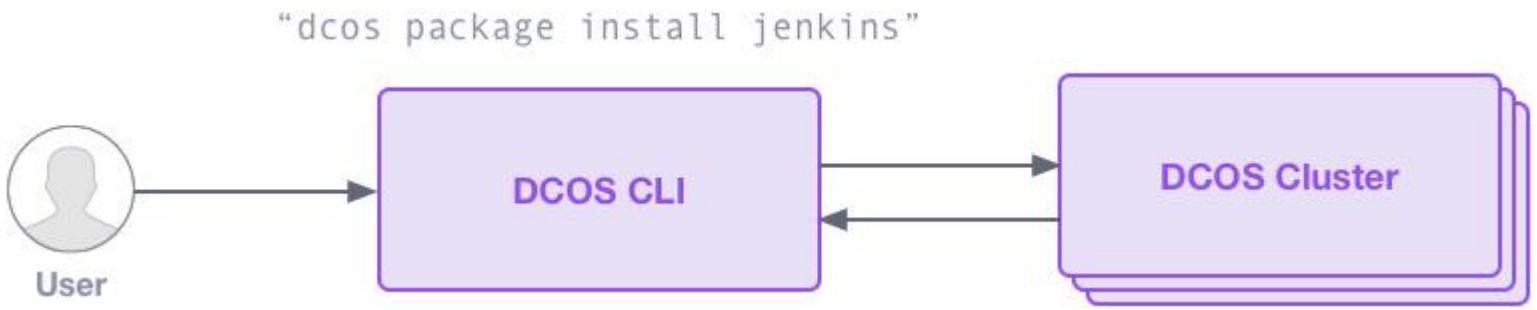




DEPLOYING A PAAS USING DCOS

DEPLOYING A PAAS USING DCOS

DCOS CLI AND THE UNIVERSE



MARATHON

1. Create a JSON file:

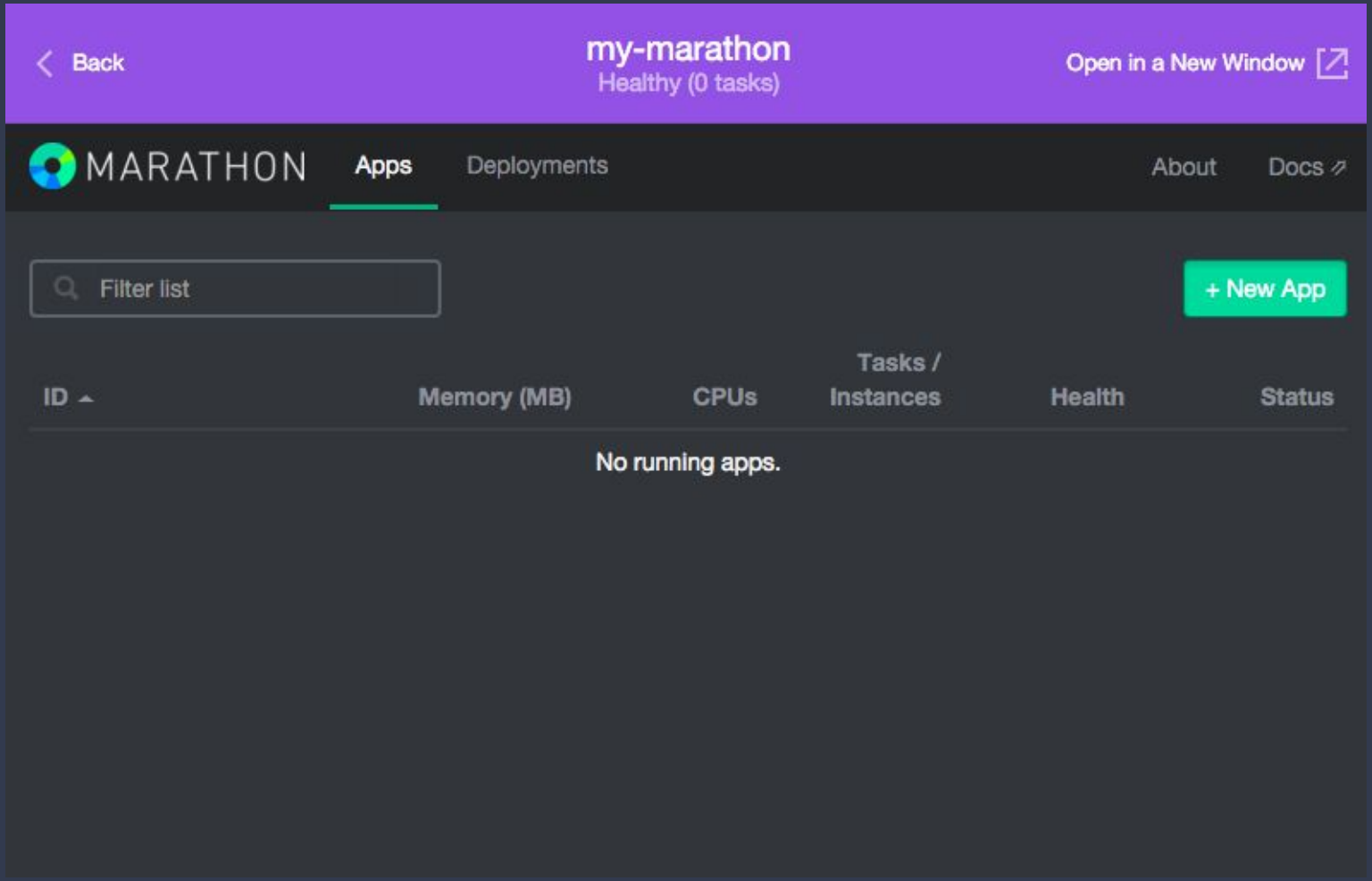
```
{"marathon": {"framework-name": "my-marathon" }}
```

2. Use the CLI to install it:

```
$ dcos package install --options=my-marathon-config.json marathon
```

3. ????
4. Profit!

MARATHON



KUBERNETES

1. Add the Multiverse:

```
$ dcos config prepend package.sources https://github.com/mesosphere/multiverse/archive/version-1.x.zip
$ dcos package update
```

2. Use the CLI to install etcd:

```
$ dcos package install etcd
```

3. Create a JSON file to configure Kubernetes:

```
{ "kubernetes": { "etcd-mesos-framework-name": "etcd" } }
```

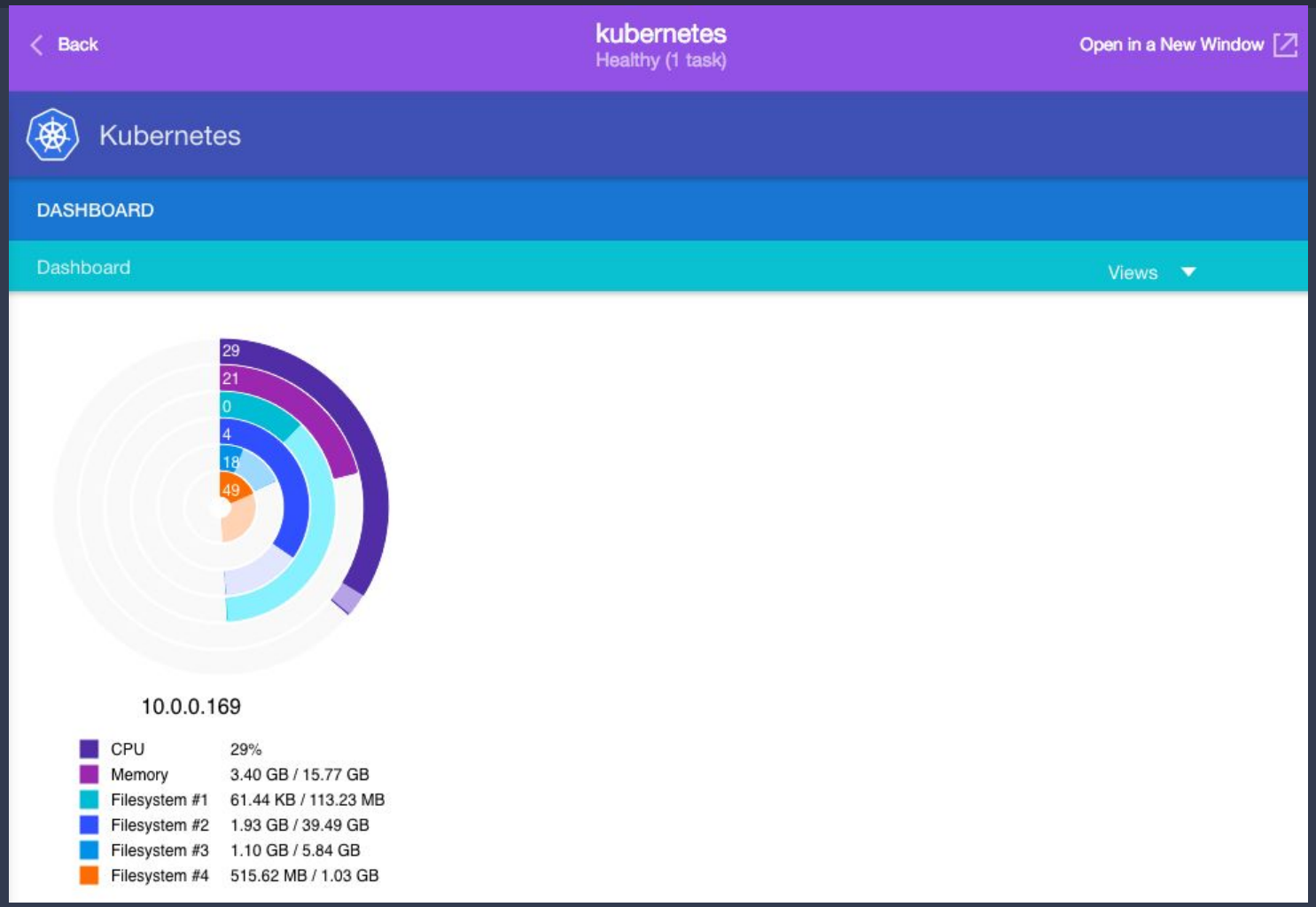
4. Use the CLI to install Kubernetes:

```
$ dcos package install --options=my-kubernetes-config.json kubernetes
```

5. ????
6. Profit!

DEPLOYING A PAAS USING DCOS

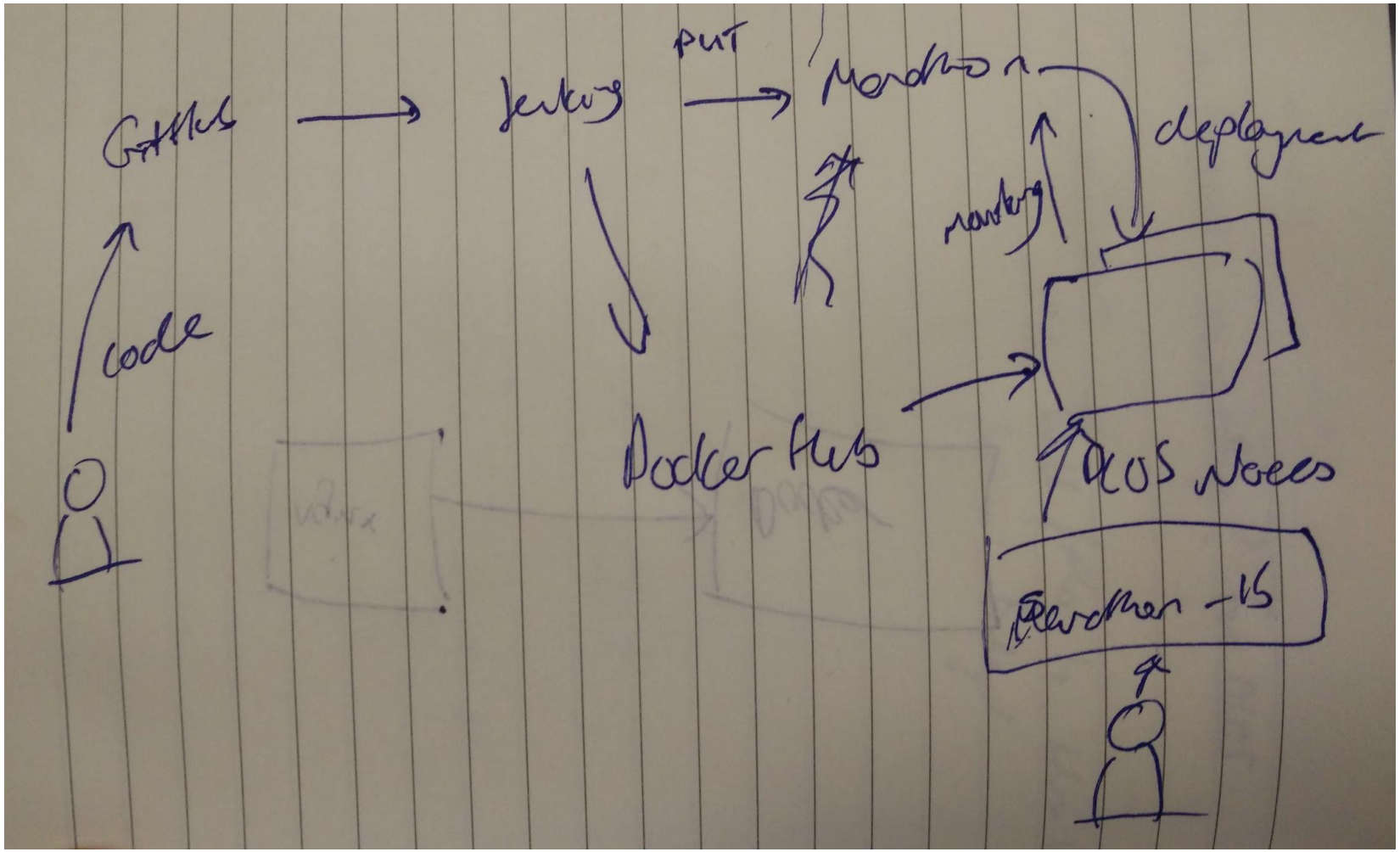
KUBERNETES

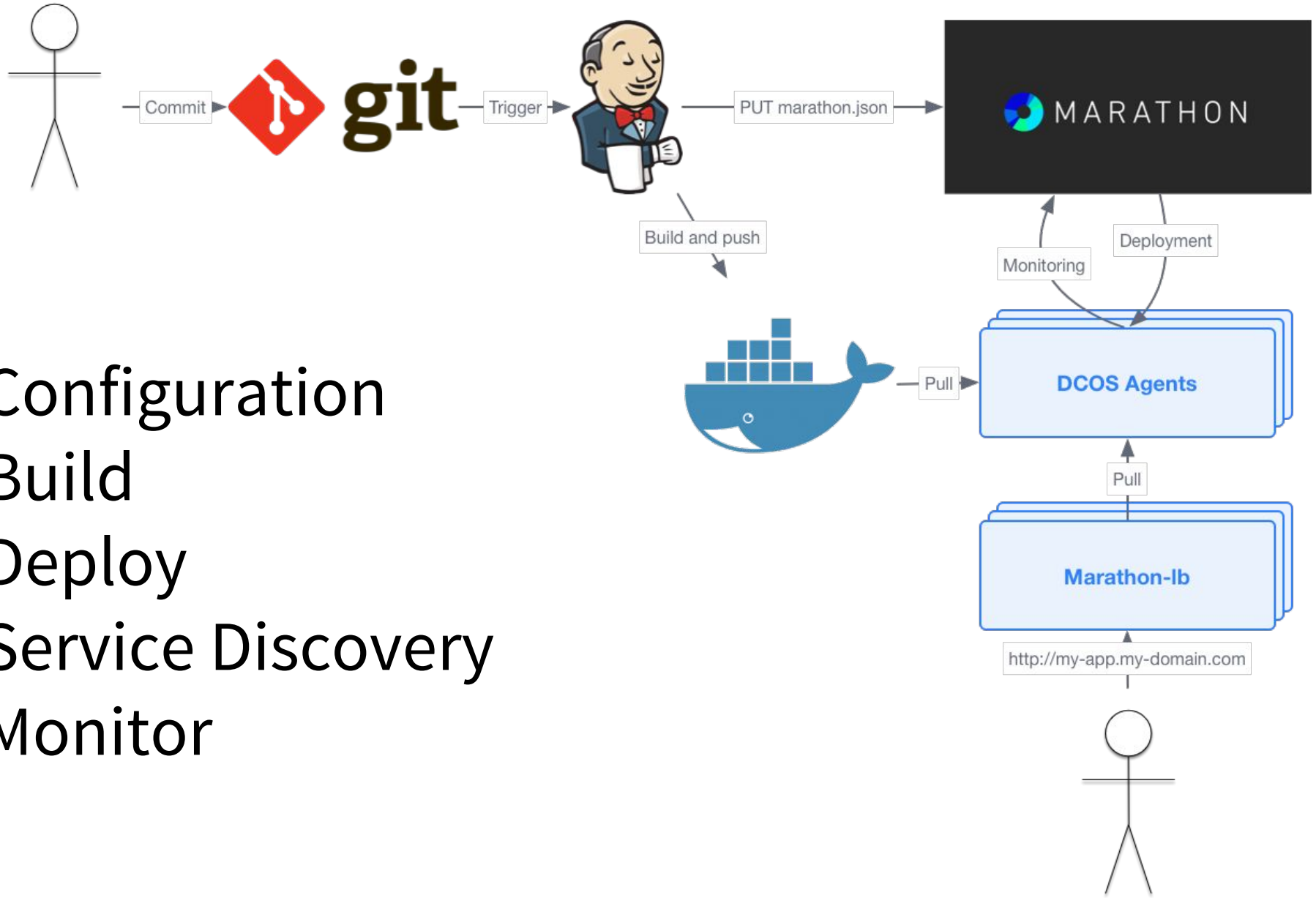


CONTINUOUS DELIVERY ON DCOS

CD ON DCOS

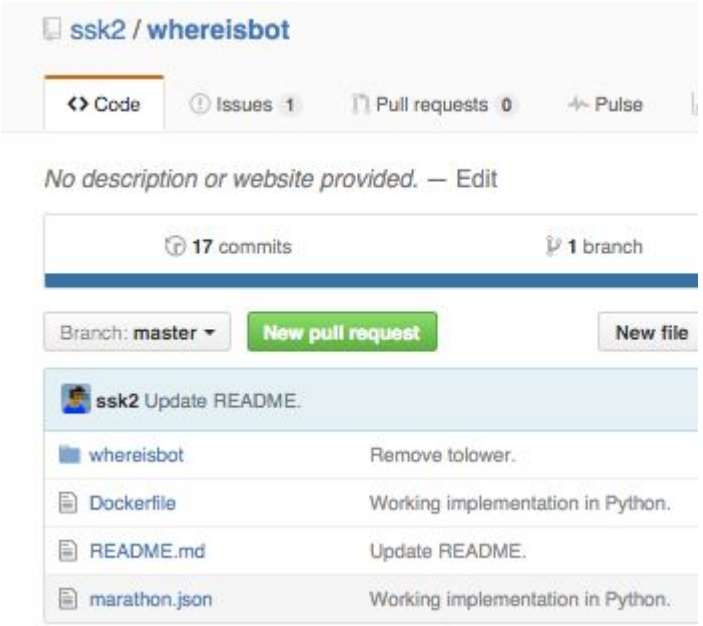
CONTINUOUS DELIVERY PIPELINE





1. Configuration
2. Build
3. Deploy
4. Service Discovery
5. Monitor

1. CONFIGURATION



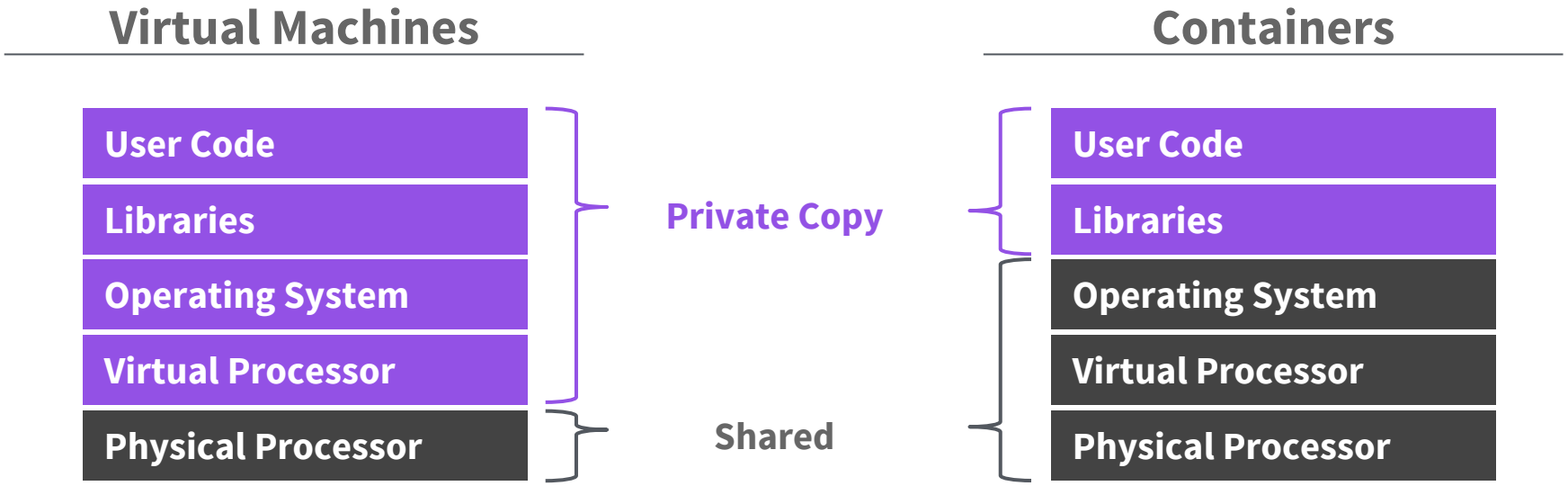
Building a CD pipeline requires configuration in a couple of places:

1. Docker and Marathon files in your repo
2. Build configuration in Jenkins*

*in the future, you'll be able to check in your build configuration alongside your repository too!

1. CONFIGURATION

DEPENDENCY MANAGEMENT



	Virtual Machines	Containers
Start time	30-45 seconds	< 50 ms
Stop time	5-10 seconds	< 50 ms
Workload density	1x	10 - 100x

1. CONFIGURATION

DEPENDENCY MANAGEMENT

Docker is becoming the de-facto container format for packaging applications:

- Encapsulates dependencies
- Runs on your laptop
- Runs on your cluster

Mesos and Marathon have native support for Docker.

Just stick a Dockerfile (or two) in the root of your repository!

1. CONFIGURATION

DEPENDENCY MANAGEMENT

```
FROM orchardup/python:2.7
RUN pip install Flask uwsgi requests
ADD . /code
WORKDIR /code
CMD uwsgi --http 0.0.0.0:8080 --wsgi-
file whereisbot/whereisbot.py --
callable app --master --no-default-app
```

1. CONFIGURATION

APPLICATION CONFIGURATION

Marathon application definitions are JSON files that describe:

- resources required
- how many instances to run
- what command to run
- how to check your application is healthy

marathon.json should live in the root of your project repository.

MARATHON

1. CONFIGURATION

APPLICATION CONFIGURATION

```
{ "id": "/ssk2/whereisbot",
  "cmd": "uwsgi --http 0.0.0.0:8080 --wsgi-file
whereisbot/whereisbot.py --callable app --master --no-default-app",
  "cpus": 0.5,
  "mem": 64,
  "instances": 1,
  "acceptedResourceRoles": [
    "slave_public"
  ],
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "ssk2/whereisbot:<build_tag>",
      "network": "BRIDGE",
      "portMappings": [
        {
          "containerPort": 8080,
          "hostPort": 0
        }
]}},
```

MARATHON

1. CONFIGURATION

APPLICATION CONFIGURATION

```
  "healthChecks": [
    {
      "protocol": "HTTP",
      "portIndex": 0,
      "path": "/",
      "gracePeriodSeconds": 5,
      "intervalSeconds": 20,
      "maxConsecutiveFailures": 3
    }
  ],
  "env": {
    "VIRTUAL_HOST": "whereis.mesosphere.com",
    "SOURCE_JSON": "https://path/to/dates.json"
  }
}
```

MARATHON

2. BUILDING

It's trivial to install Jenkins on DCOS:

1. Create a JSON file:

```
{"jenkins": {"framework-name": "my-jenkins" }}
```

2. Install:

```
$ dcos package install --options=my-jenkins-config.json jenkins:
```

3. ???
4. Profit!

2. BUILDING

Now, set up your build:

1. Set up a GitHub webhook to trigger Jenkins builds

2. Set up build to run tests

3. Set up triggered build to build and push Docker image

```
docker build . -t ssk2/whereisbot:$(GIT_BRANCH)

docker push ssk2/whereisbot:$(GIT_BRANCH)
```

4. Set up triggered build to update marathon.json using jq and PUT to Marathon

```
http PUT https://dcos/service/my-marathon/v2/app/ssk2/whereisbot < marathon.json
```

3. DEPLOYING

MARATHON

When you PUT to Marathon's API, you trigger a deployment.

```
http PUT https://dcos/service/my-marathon/v2/app/ssk2/whereisbot < marathon.json
```

Marathon attempts to scale application to desired state by:

- Launching new instances
  - By default try to launch 100% of instances requested at once
- Killing old instances when new instances are healthy

4. SERVICE DISCOVERY

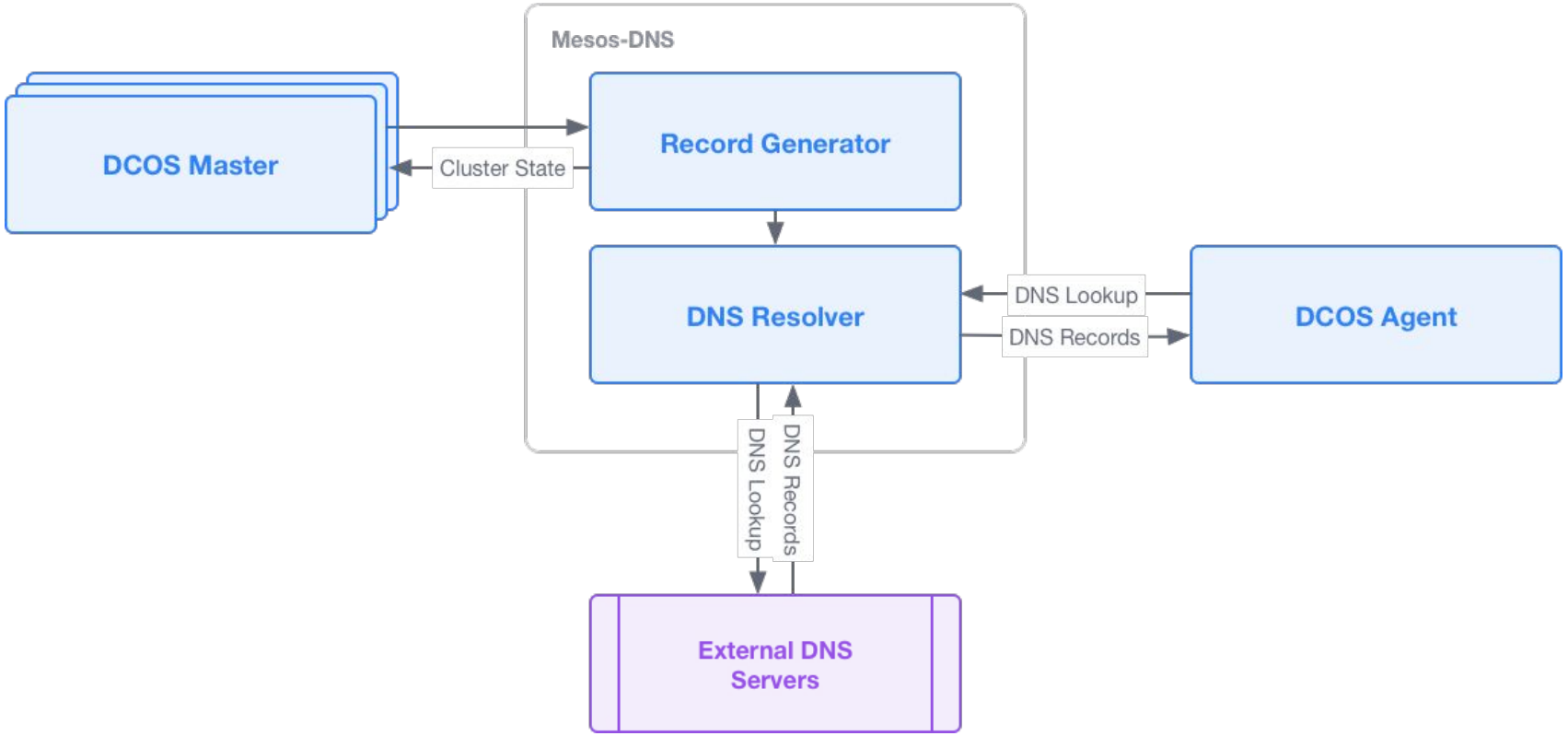
Two main service discovery mechanisms are provided with DCOS:

1. DNS based (Mesos-DNS)
2. HAProxy based (Marathon-lb)

4. SERVICE DISCOVERY

MESOS-DNS

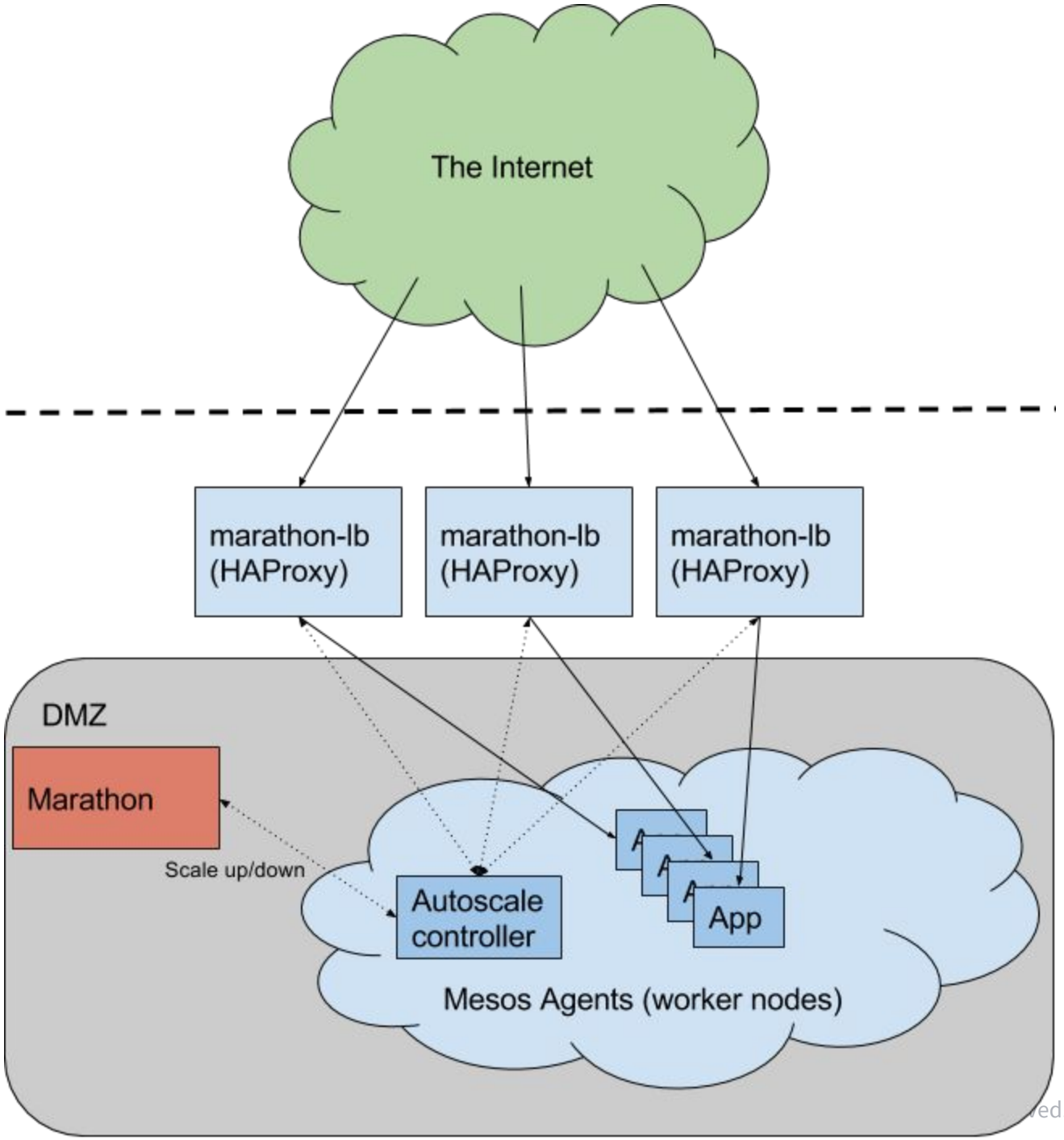
- Ingests cluster state periodically.
- Uses cluster state to generate DNS records for all running Mesos tasks.
- Services query DNS server to discover IP address and port of other services.
- Primarily used for internal service discovery.
- No extra configuration required!



4. SERVICE DISCOVERY

MARATHON-LB

- Ingests state of running Marathon applications.
- Regenerates HAProxy configuration.
- Supports virtual hosts!
- Can be used for both internal and external service discovery.
- Must add HAPROXY_GROUP and HAPROXY_0_VHOST variables to your marathon.json.



CD ON DCOS

5. MONITORING

Marathon takes care of monitoring your application using the health checks you specified earlier!

chronos	DCOS_PACKAGE_IS_FRAMEWORK:true ...	0.5	512 MiB	Running	1 of 1
dd		0.0	0 B	Suspended	0 of 0
dispatch		0.5	128 MiB	Running	1 of 1
frontend-foosball		0.0	0 B	Suspended	0 of 0
history		0.5	256 MiB	Running	1 of 1
ie-app		0.1	16 MiB	Running	1 of 1
kdc		1.0	1 GiB	Running	1 of 1
kinit		1.0	1 GiB	Running	1 of 1
marathon-hands-on		0.0	0 B	Suspended	0 of 0
marathon-on-marathon		12.0	24 GiB	Running	3 of 3
marathon-scale-test		2.0	2 GiB	Running	1 of 1

MARATHON

COMING SOON

COMING SOON

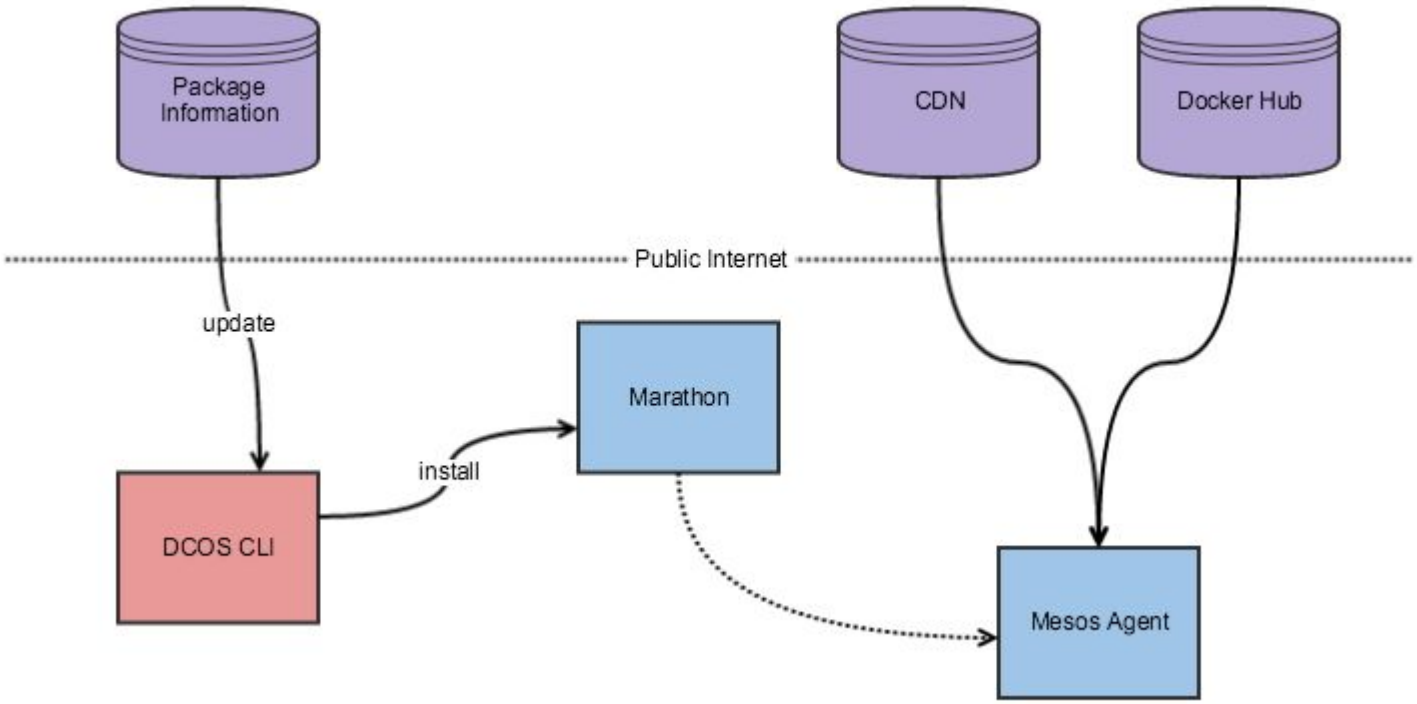
LOCAL UNIVERSE

The Universe is the public package repository for DCOS.

It lives on GitHub, assets are stored on DockerHub or Amazon S3.

Often, you don't want your application to call out to the world.

Alternatively you only want to publish packages locally.



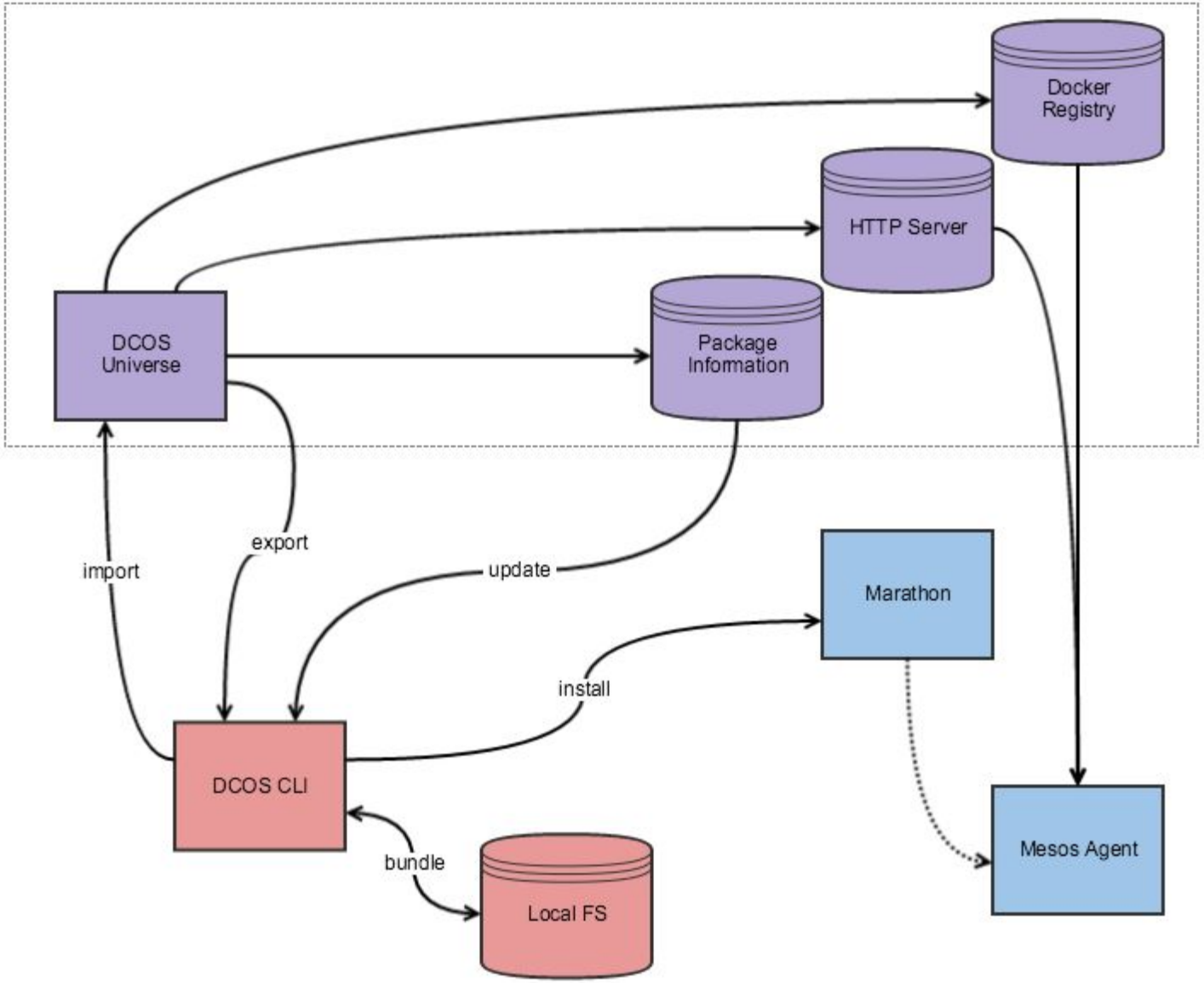
COMING SOON

LOCAL UNIVERSE

Command line features to bundle your own packages and push them up to an internally hosted registries and file servers.

Easily build this into your own pipeline and use the same commands to install your own packages as you do for public packages!

Still very much being built!

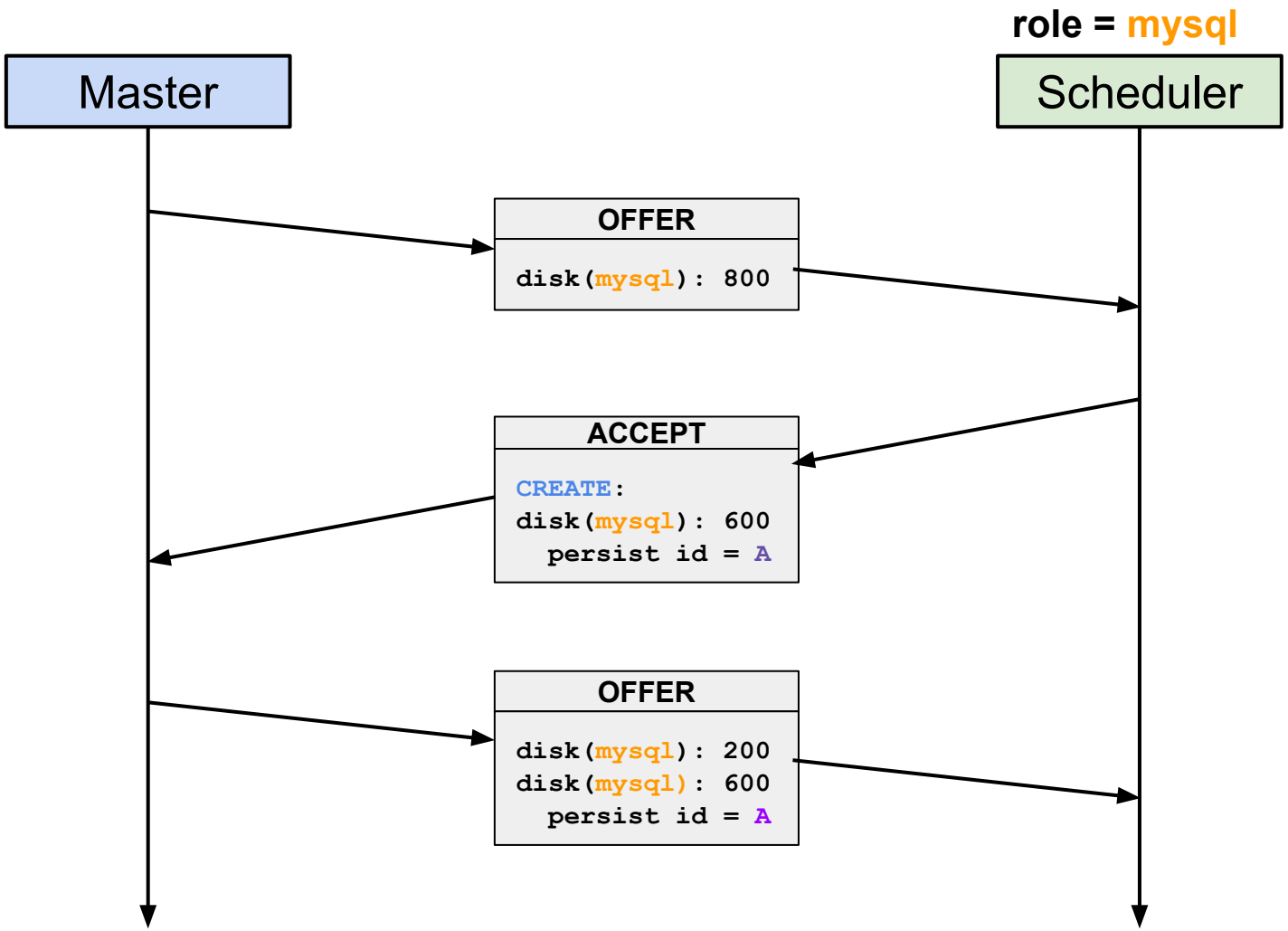


COMING SOON

PERSISTENT VOLUMES

As of 0.23.0 Mesos now offers primitives to allow schedulers to create volumes.

Marathon will soon have support for these - your application will be able to re-attach to its data!



THANK YOU!

Come and talk to us!

- Email me at sunil@mesosphere.io
- Slides will be up at http://mesosphere.github.io/presentations