

---

Sunil Shah

# BUILDING A CONTAINERIZED CONTINUOUS DEPLOYMENT PIPELINE

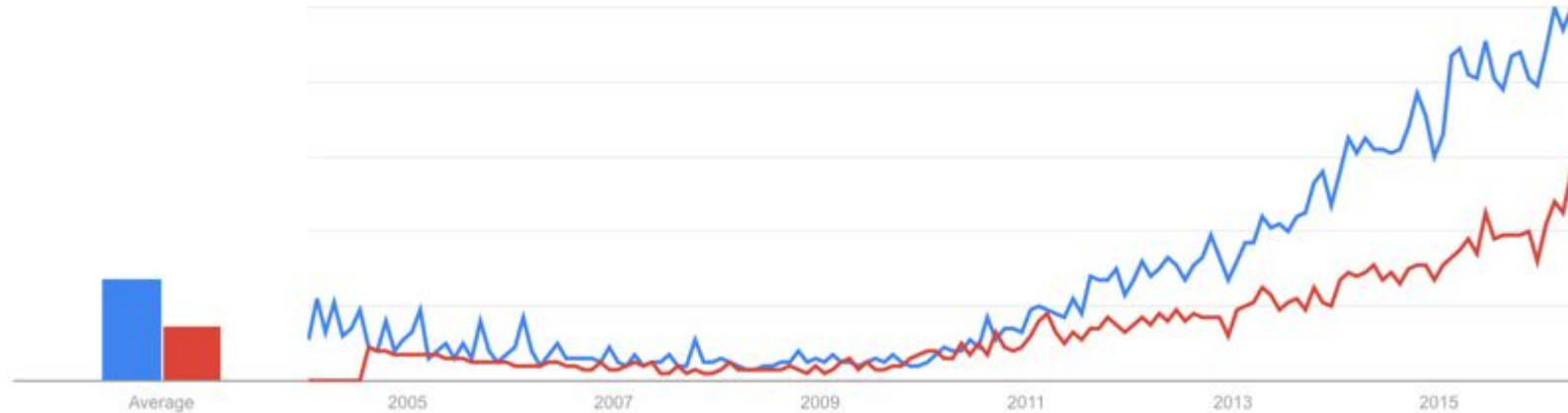
(using Apache Mesos, Jenkins and Marathon)



MESOSPHERE

# WHY BOTHER?

Why is continuous deployment interesting now?



Google Trends for continuous delivery (blue) and continuous deployment (red)

# WHY BOTHER?

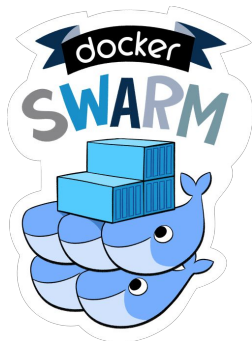
## 1. It's much easier to get compute resources nowadays!

- Doesn't cost much (sometimes it's even free - just ask a graduate student)
- **EVERY** platform and their dog has an API



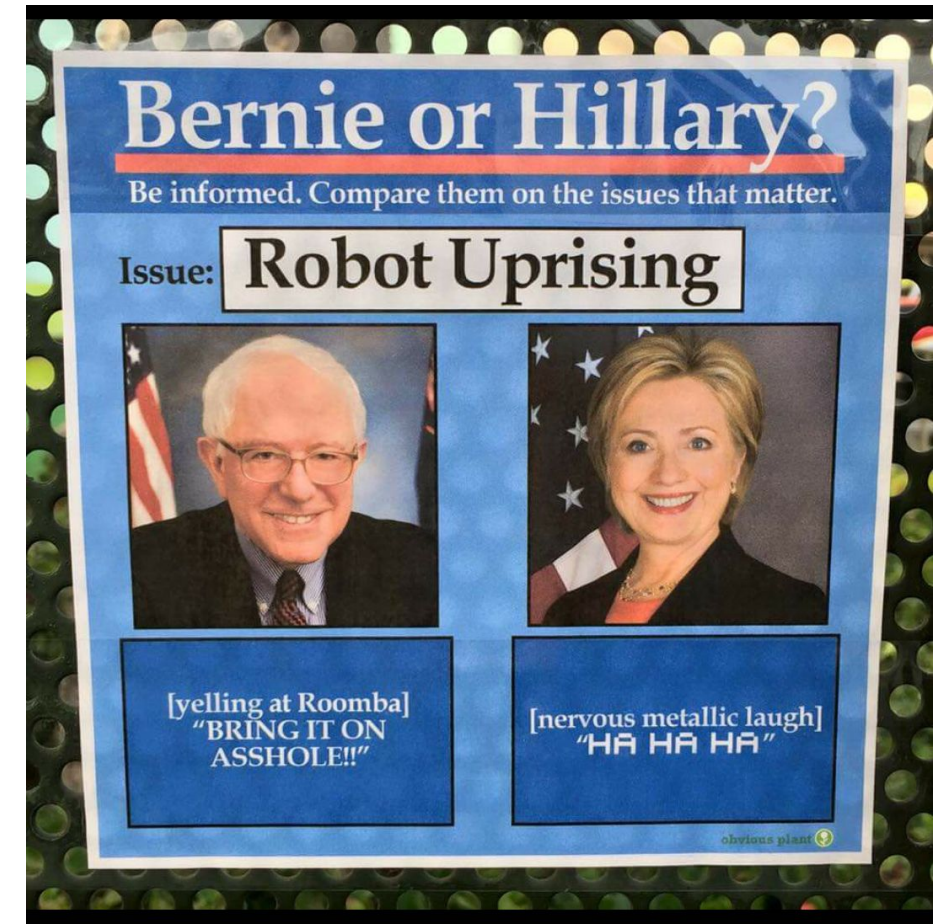
# WHY BOTHER?

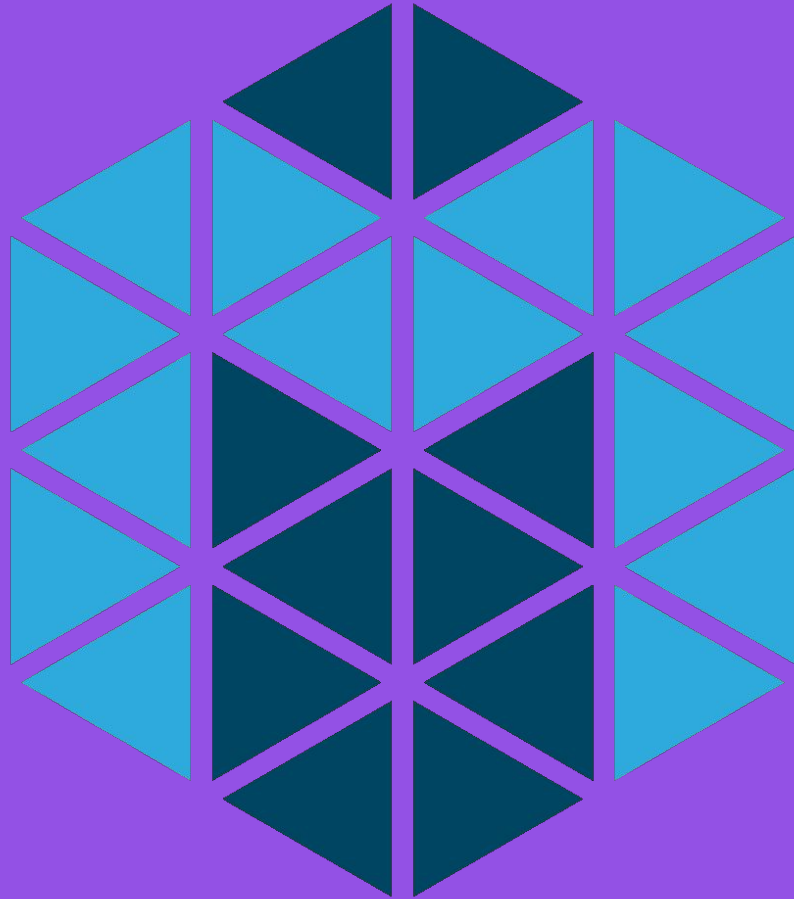
1. It's much easier to get compute resources nowadays!
2. **It turns out getting sleep is good for you!**
  - The National Sleep Foundation recommends 7-9 hours of sleep per night
  - Container orchestrators take the manual pain out of waking up and rebooting an application (to varying degrees of success)
  - Let your devs dev and ops sleep!



# WHY BOTHER?

1. It's much easier to get compute resources nowadays!
2. It turns out getting sleep is good for you!
3. **Containers mean you can!**
  - No need for humans to ssh in and ``apt-get package install python-mylibrary123``.
  - Is this the beginning of the robot uprising?

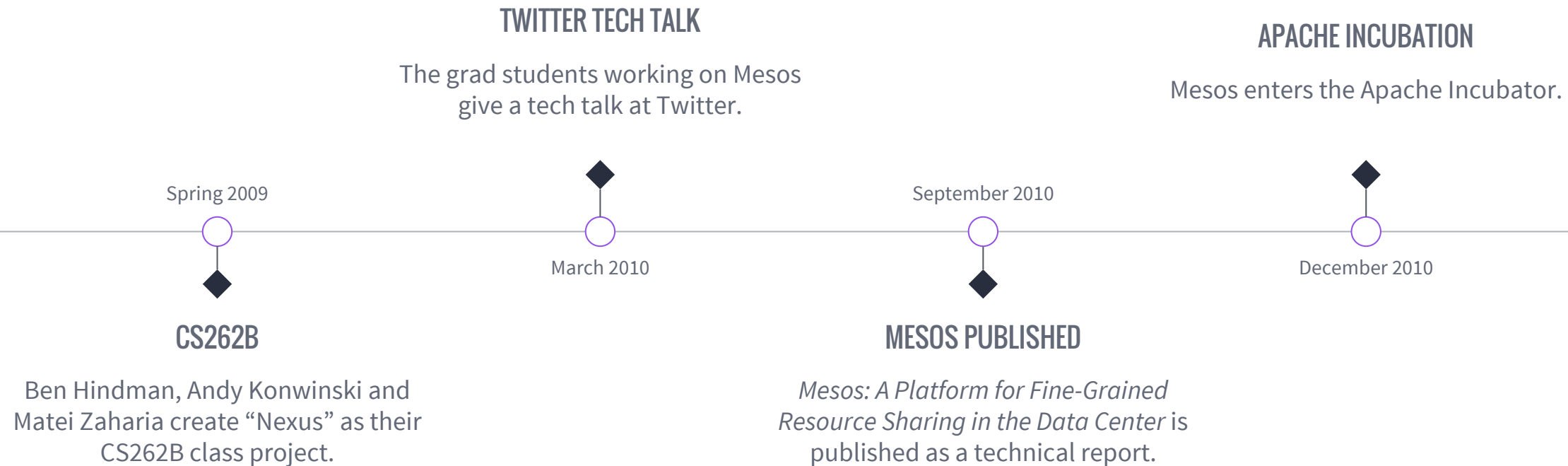




---

# APACHE MESOS: THE STORY OF

# THE BIRTH OF MESOS



# GRAD STUDENTS LEARNED HOW TO SHARE

## 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*

Sharing resources between batch processing frameworks:

- Hadoop
- MPI
- Spark

## The Datacenter Needs an Operating System

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

What does an operating system provide?

- Resource management
- Programming abstractions
- Security
- Monitoring, debugging, logging



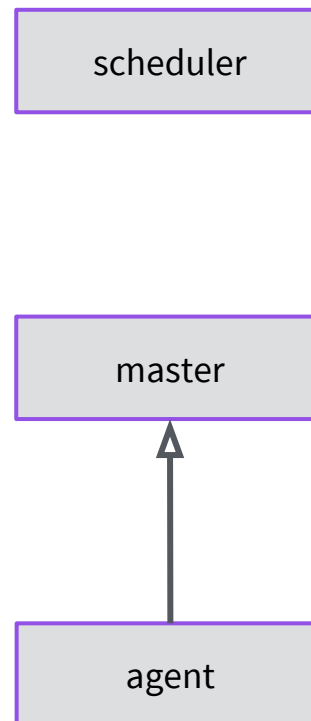
# CLUSTERING YOUR RESOURCES FOR YOU

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

It handles:

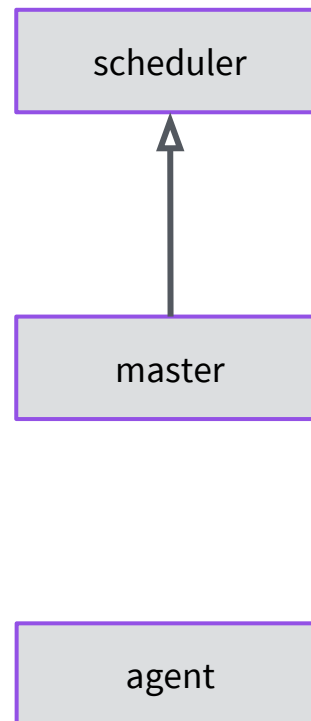
- **Aggregating resources** and **offering them to schedulers**
- **Launching tasks** (i.e. processes) on those resources
- **Communicating the state of those tasks** back to schedulers
- Tasks can be:
  - Long running services
  - Ephemeral / batch jobs

# A SIMPLE MESOS CLUSTER ON MONDAY MORNING



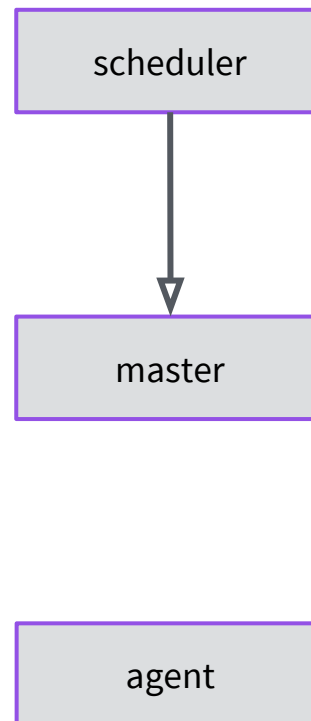
“Sir, I have some spare resources:  
4 CPUs, 8 GB of memory and 1 TB of  
disk.”

# A SIMPLE MESOS CLUSTER ON MONDAY MORNING



“Hey, scheduler, would you like some compute resources?”

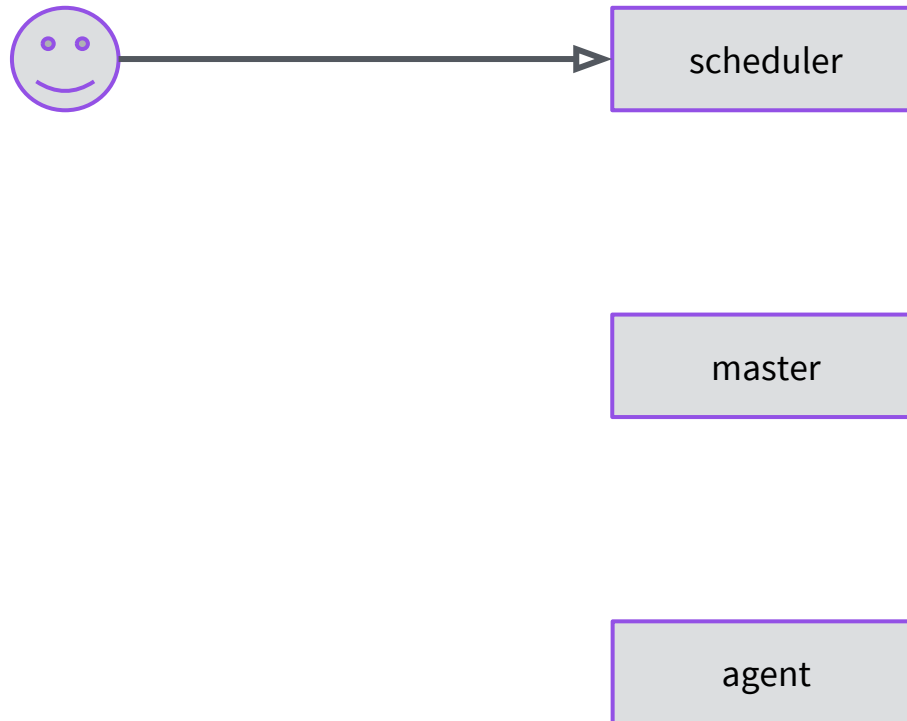
# A SIMPLE MESOS CLUSTER ON MONDAY MORNING



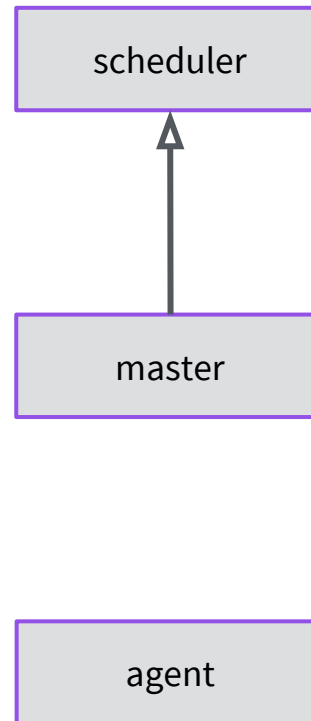
“Not right now, but thanks!”

# A SIMPLE MESOS CLUSTER ON MONDAY MORNING

“Happy Monday! Here’s a bunch of work.”

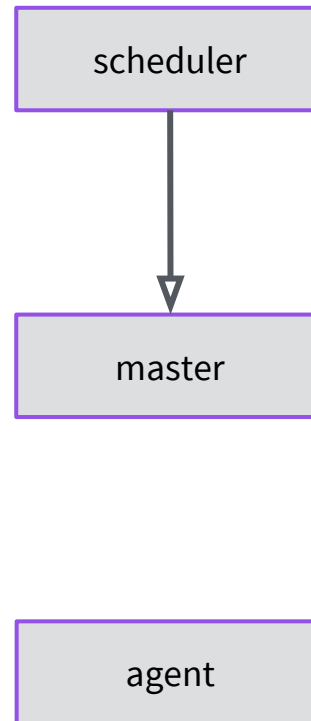


# A SIMPLE MESOS CLUSTER ON MONDAY MORNING



“Changed your mind?”

# A SIMPLE MESOS CLUSTER ON MONDAY MORNING



“How’d you know? Mind running this for me please?”

# A SIMPLE MESOS CLUSTER ON MONDAY MORNING



scheduler

master

agent

“Still got those spare resources? This task wants to run on them. Let me know how it goes.”



# A SIMPLE MESOS CLUSTER ON MONDAY MORNING



scheduler

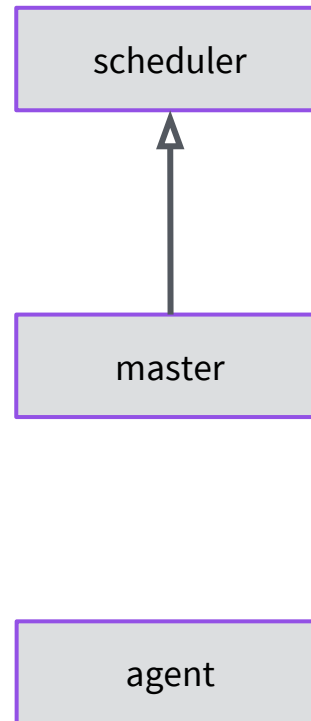
master

agent



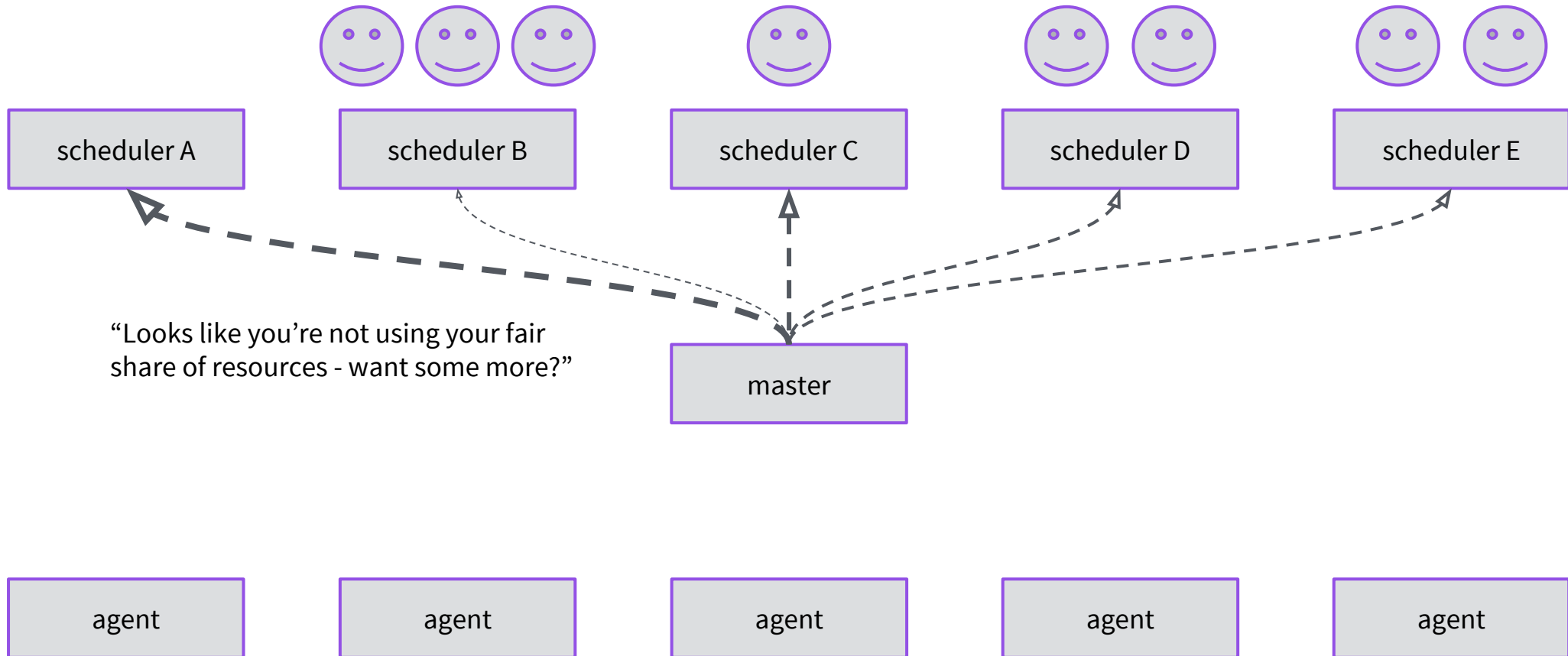
“Sure thing, it’s running.”

# A SIMPLE MESOS CLUSTER ON MONDAY MORNING

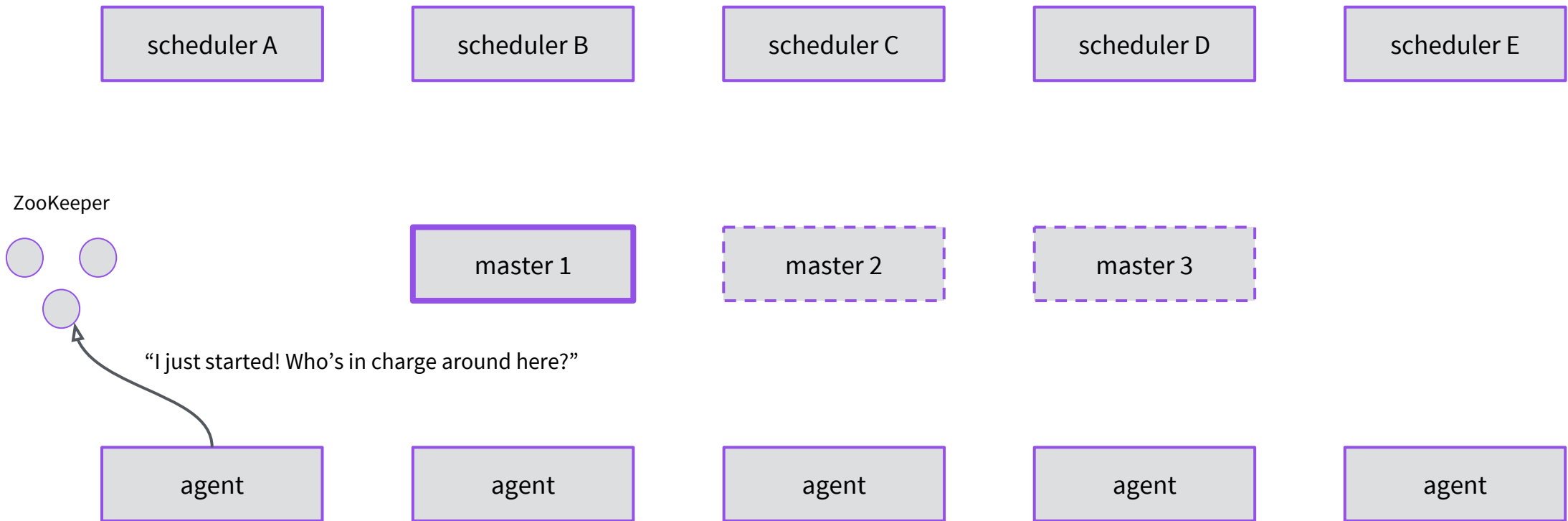


“Your task is running! I’ll let you know if that changes.”

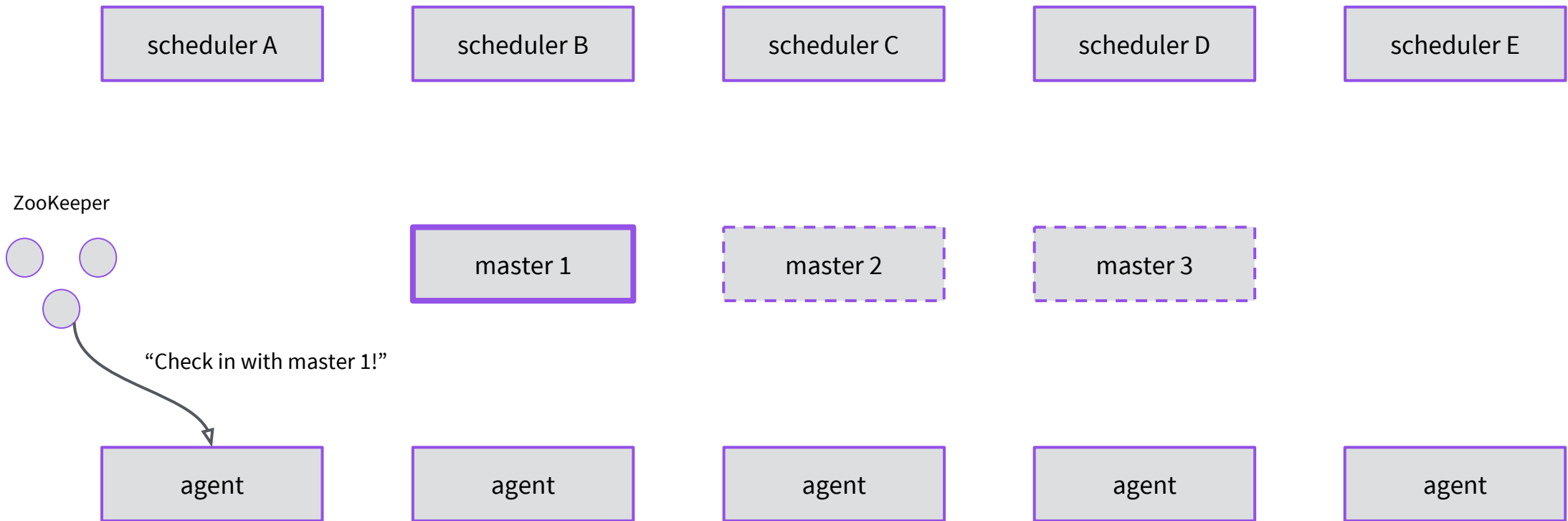
# FAIRNESS FOR ALL SCHEDULERS



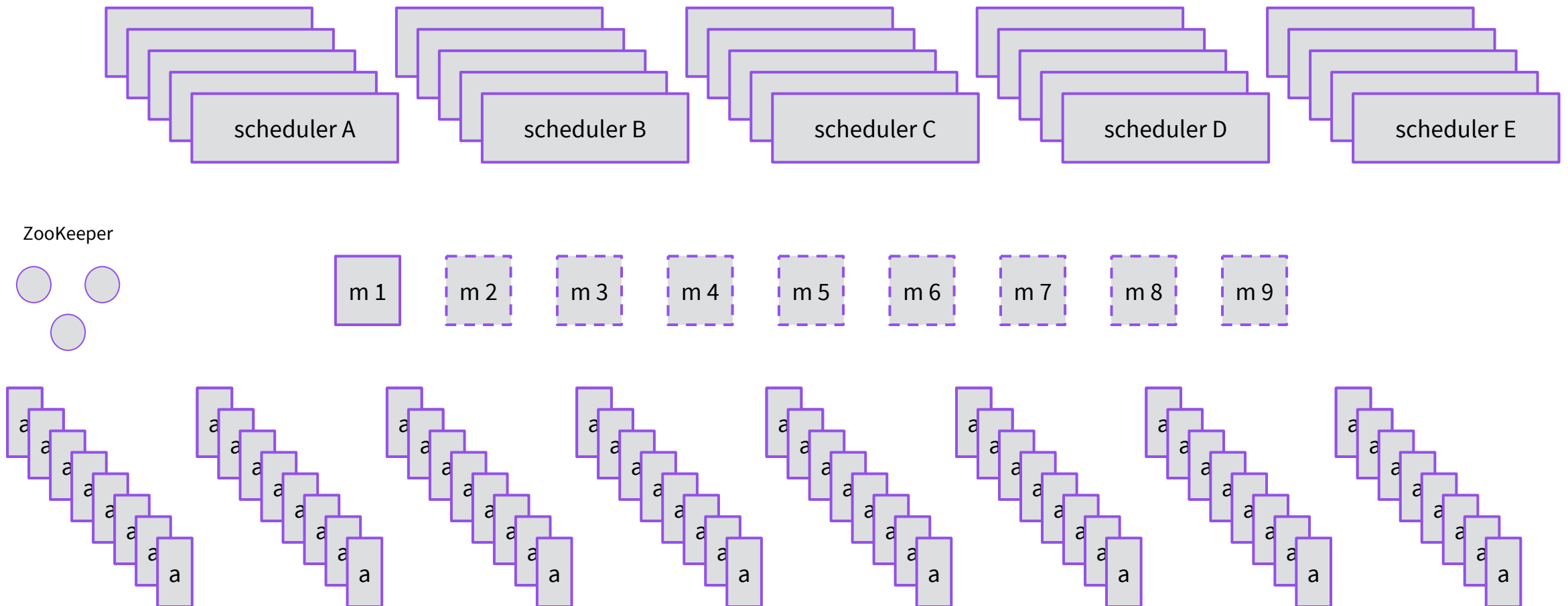
# HELPING YOUR OPERATOR SLEEP WELL



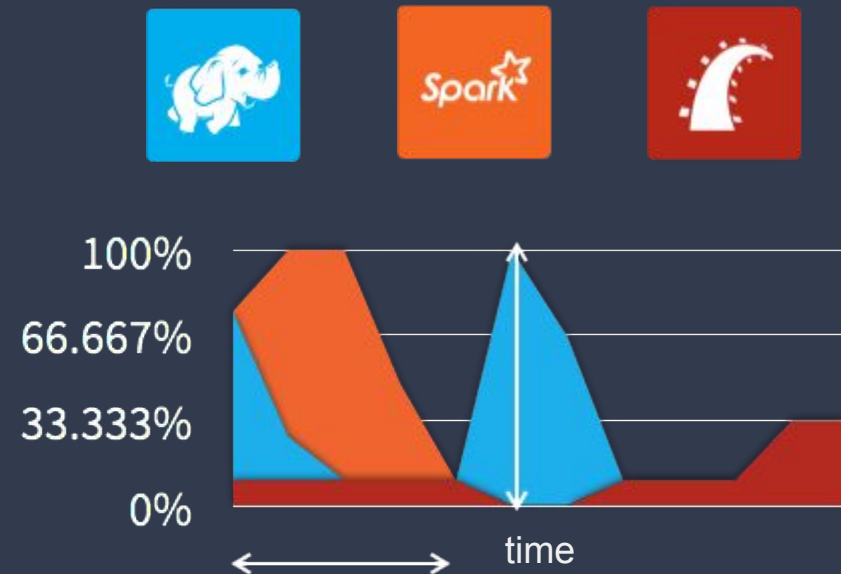
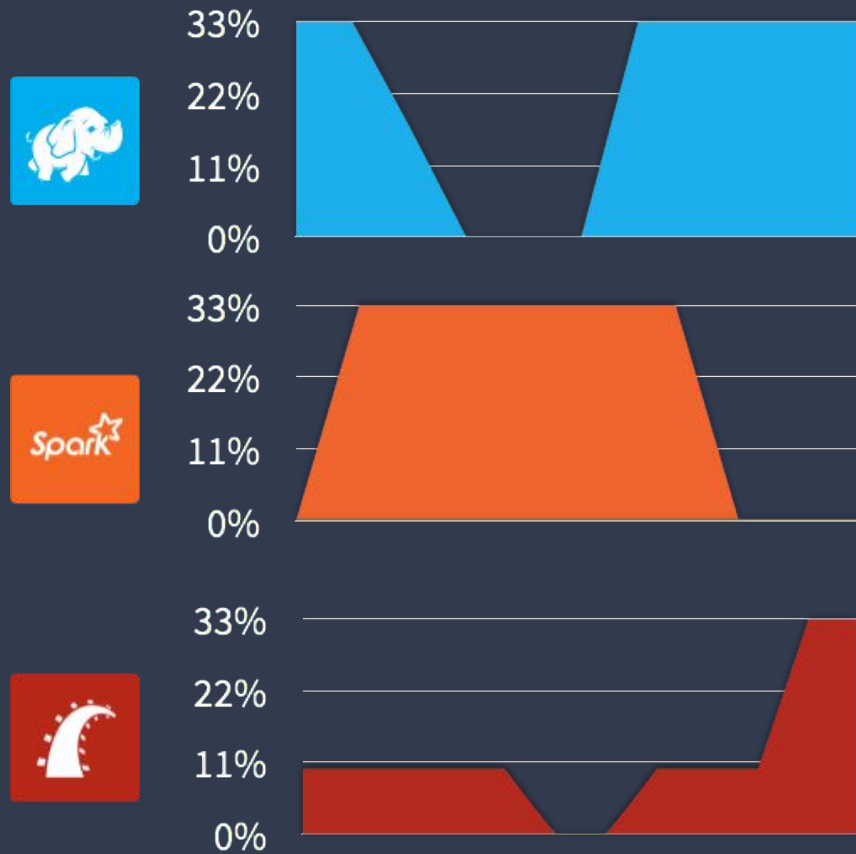
# HELPING YOUR OPERATOR SLEEP WELL



# MESOS CLUSTERS CAN BE REALLY, REALLY LARGE



# NOT USING MESOS IS EXPENSIVE!



Apache Mesos: The Story Of

# PRODUCTION MESOS USERS



Bloomberg



NETFLIX







MARATHON

---

# MARATHON

## (OR, HOW TO RUN MICROSERVICES ON MESOS)

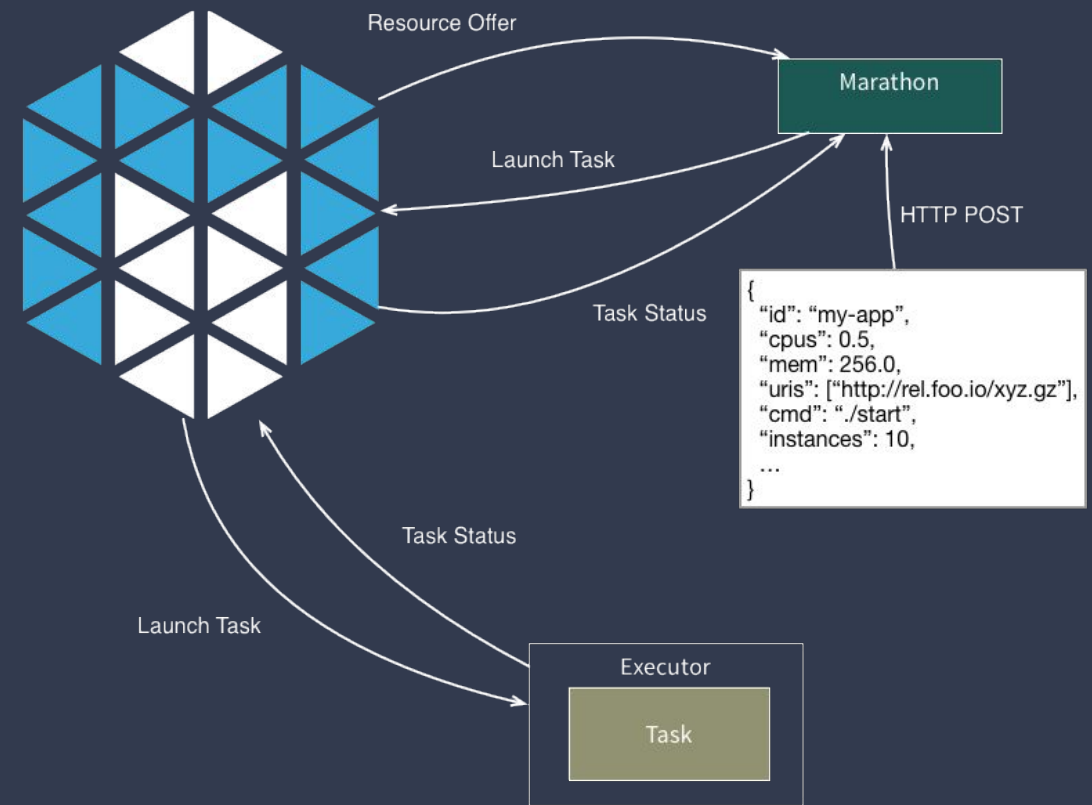
Marathon

# MARATHON TALKS TO MESOS

Mesos can't run applications on its own (!)

That's where schedulers like Apache Aurora and Marathon come in.

Marathon keeps your application running!  
A bit like a distributed "init.d".



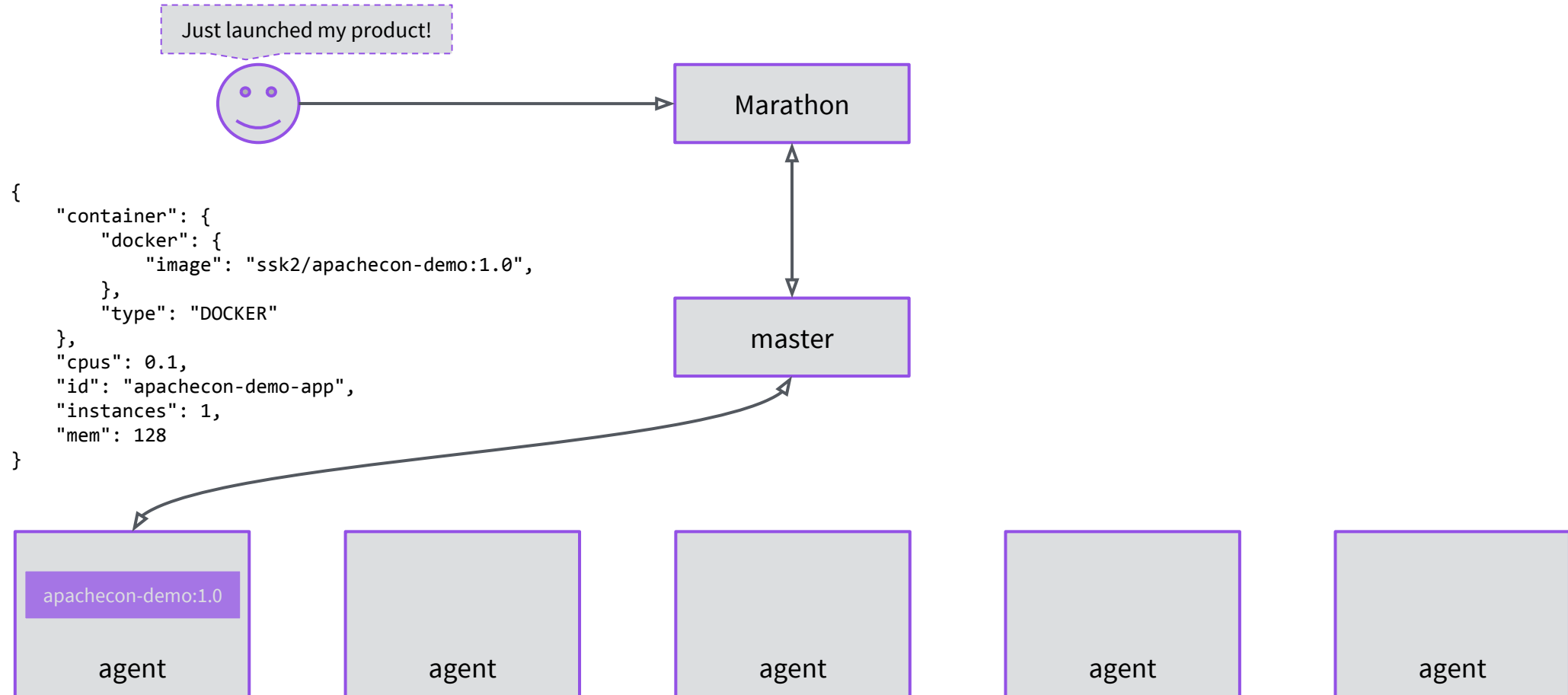
Marathon

# A SELF SERVE INTERFACE TO YOUR CLUSTER

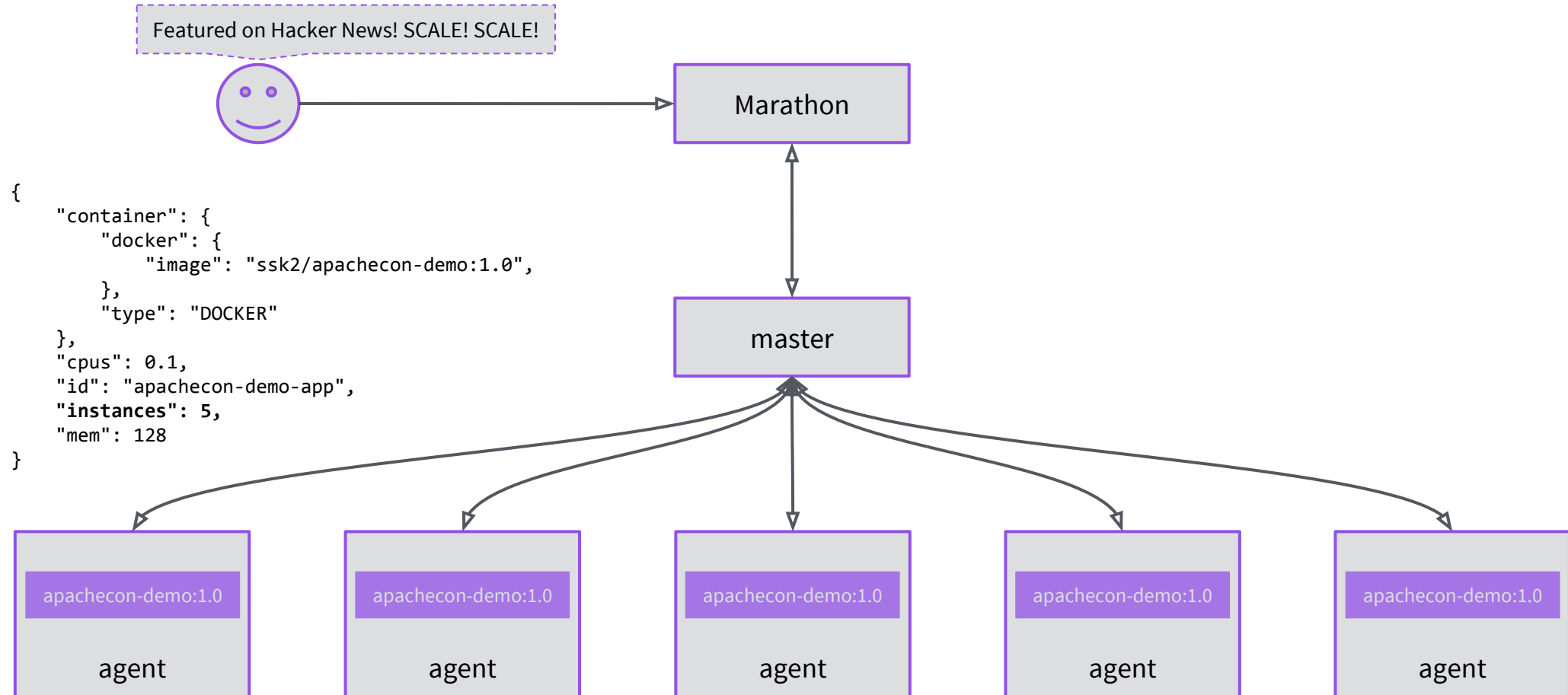
The screenshot displays the Marathon web interface. The top navigation bar includes the Marathon logo, tabs for 'Applications' and 'Deployments', a search bar labeled 'Search all applications', a help icon, and the user 'Bootstrap superuser'. The left sidebar contains filters for 'STATUS' (Running, Deploying, Suspended, Delayed, Waiting), 'HEALTH' (Healthy, Unhealthy, Unknown), 'LABEL' (a 'Select' dropdown), and 'RESOURCES' (Volumes). The main content area, titled 'Applications', shows a table of running applications. The table has columns for Name, CPU, Memory, Status, Running Instances, and Health. Three applications are listed: 'apachecon-demo-app' (0.1 CPU, 128 MiB), 'jenkins' (1.0 CPU, 1 GiB), and 'marathon-lb' (2.0 CPU, 1 GiB). All are in a 'Running' state with 1 of 1 instances. A message 'Error fetching apps. Refresh to try again.' is displayed above the table rows.

Name	CPU	Memory	Status	Running Instances	Health
apachecon-demo-app	0.1	128 MiB	Running	1 of 1	Healthy
jenkins	1.0	1 GiB	Running	1 of 1	Healthy
marathon-lb	2.0	1 GiB	Running	1 of 1	Healthy

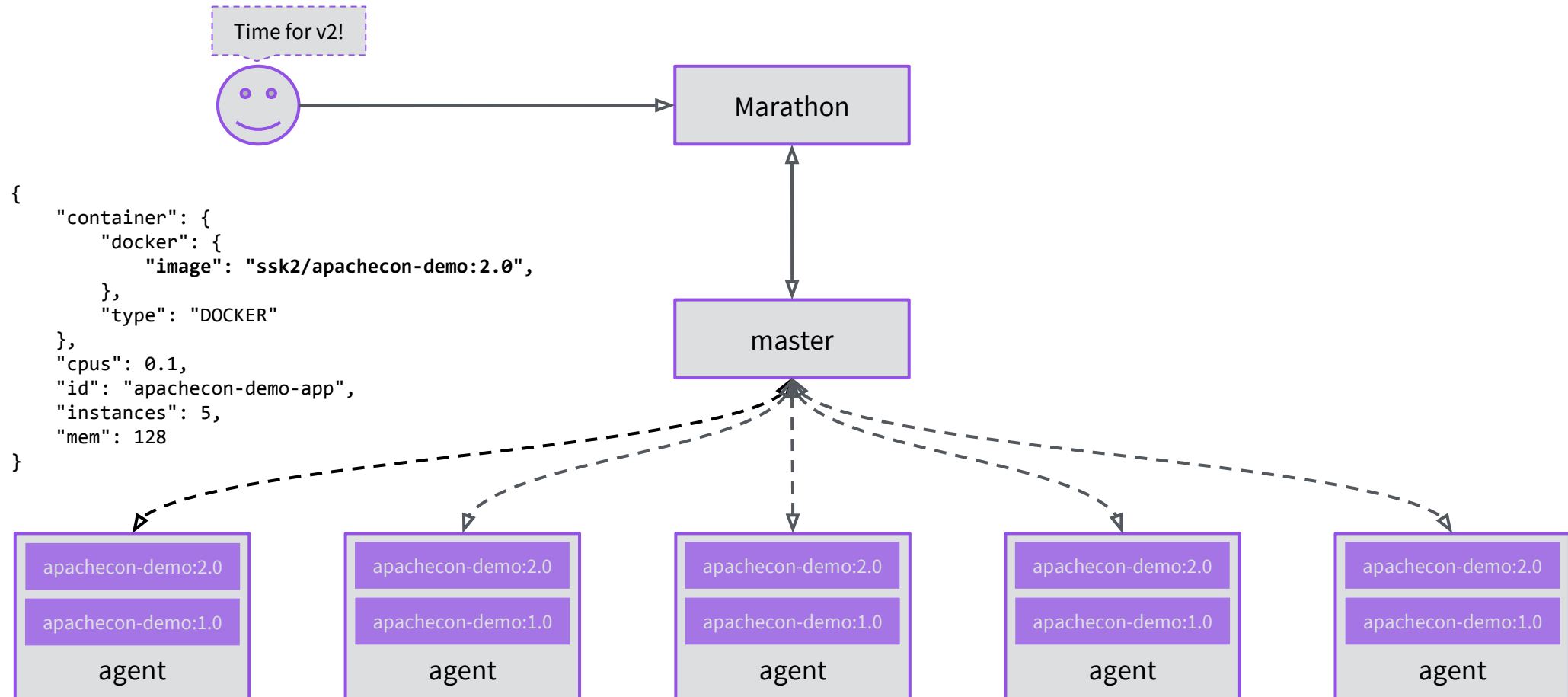
# MARATHON MANAGES THE APPLICATION LIFECYCLE



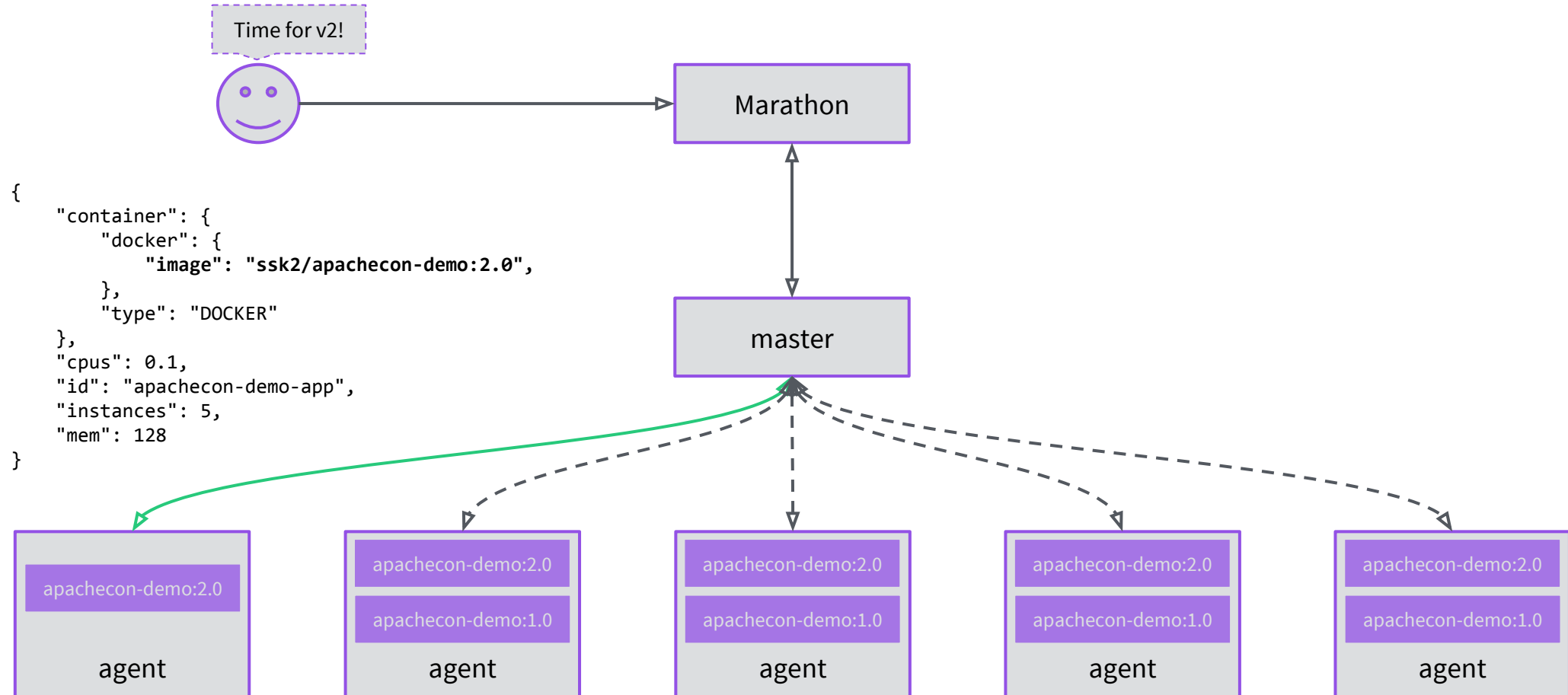
# MARATHON MANAGES THE APPLICATION LIFECYCLE



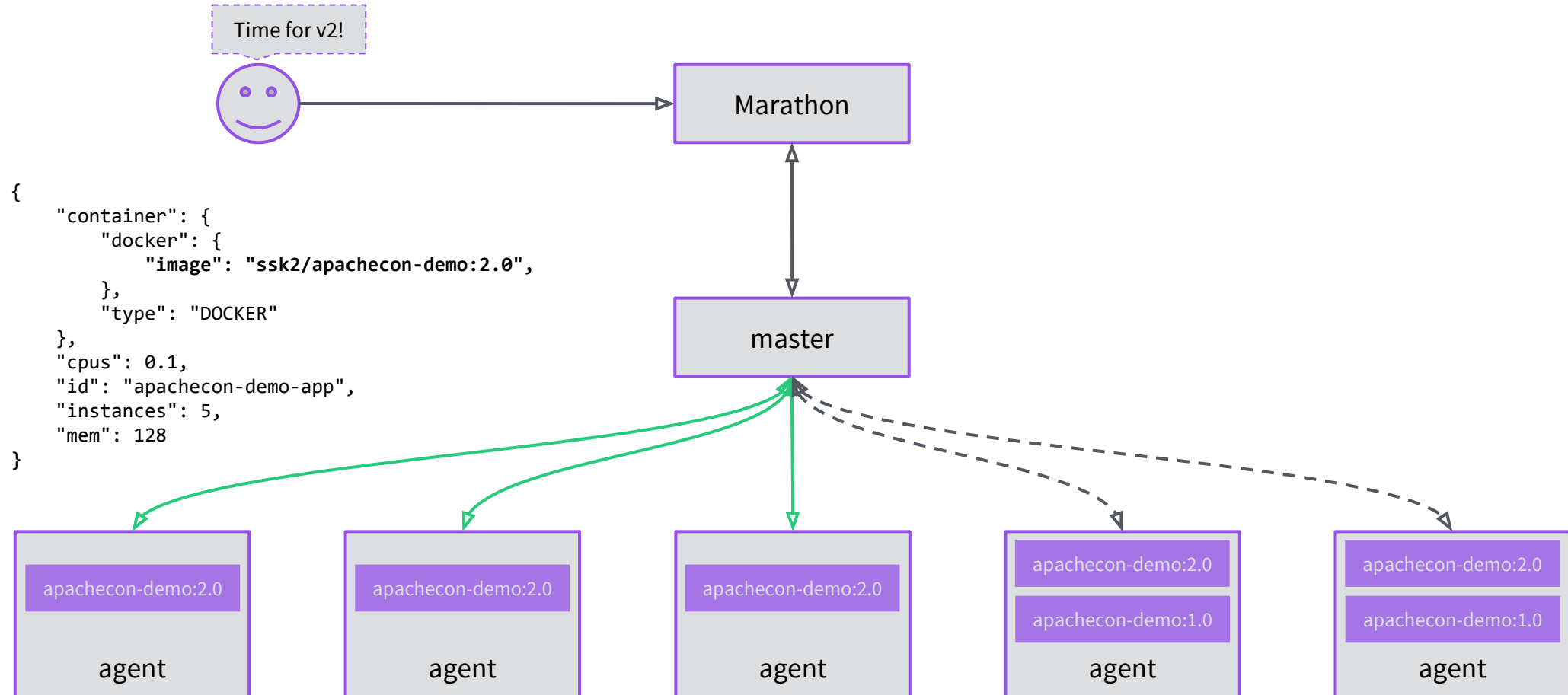
# MARATHON MANAGES THE APPLICATION LIFECYCLE



# MARATHON MANAGES THE APPLICATION LIFECYCLE

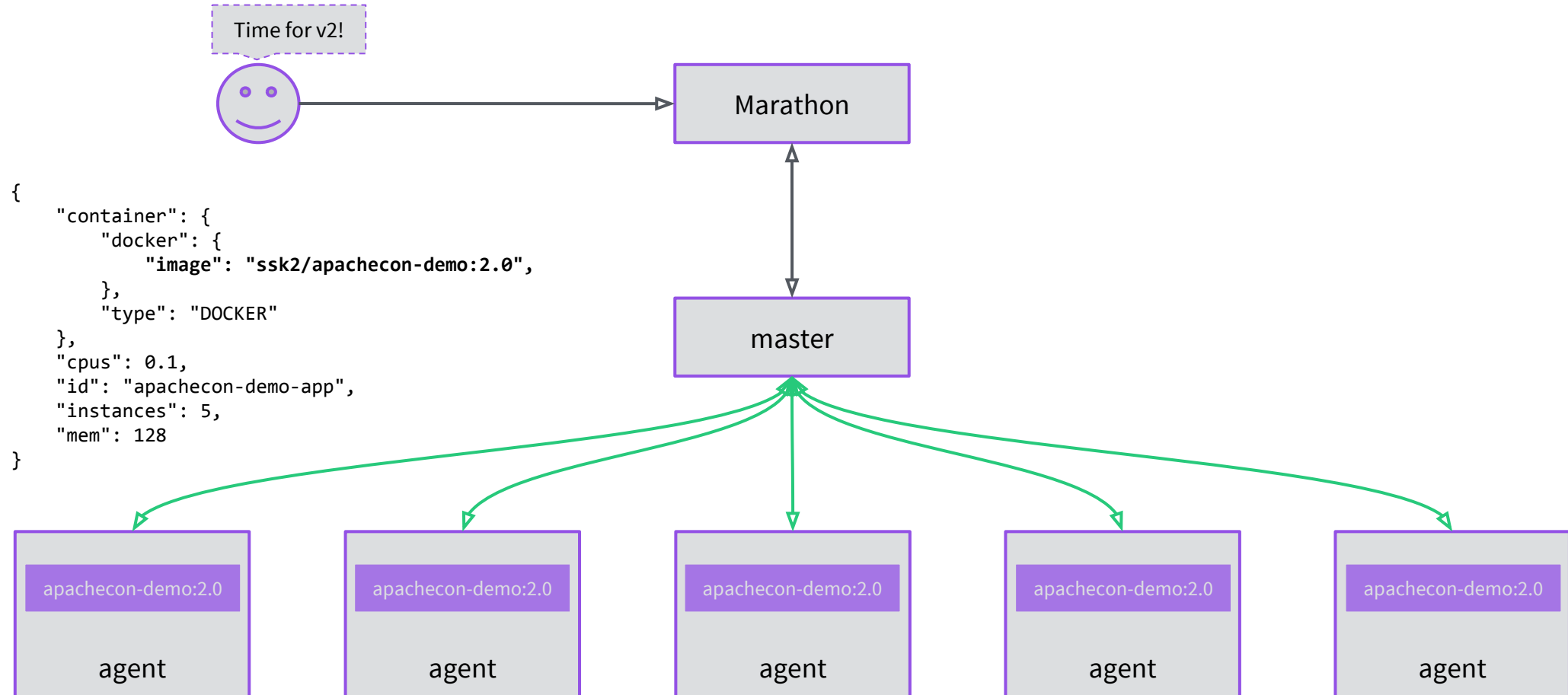


# MARATHON MANAGES THE APPLICATION LIFECYCLE





# MARATHON MANAGES THE APPLICATION LIFECYCLE





---

# JENKINS ON MESOS

(AND WHY YOU  
SHOULD PROBABLY BE  
RUNNING IT LIKE THIS)

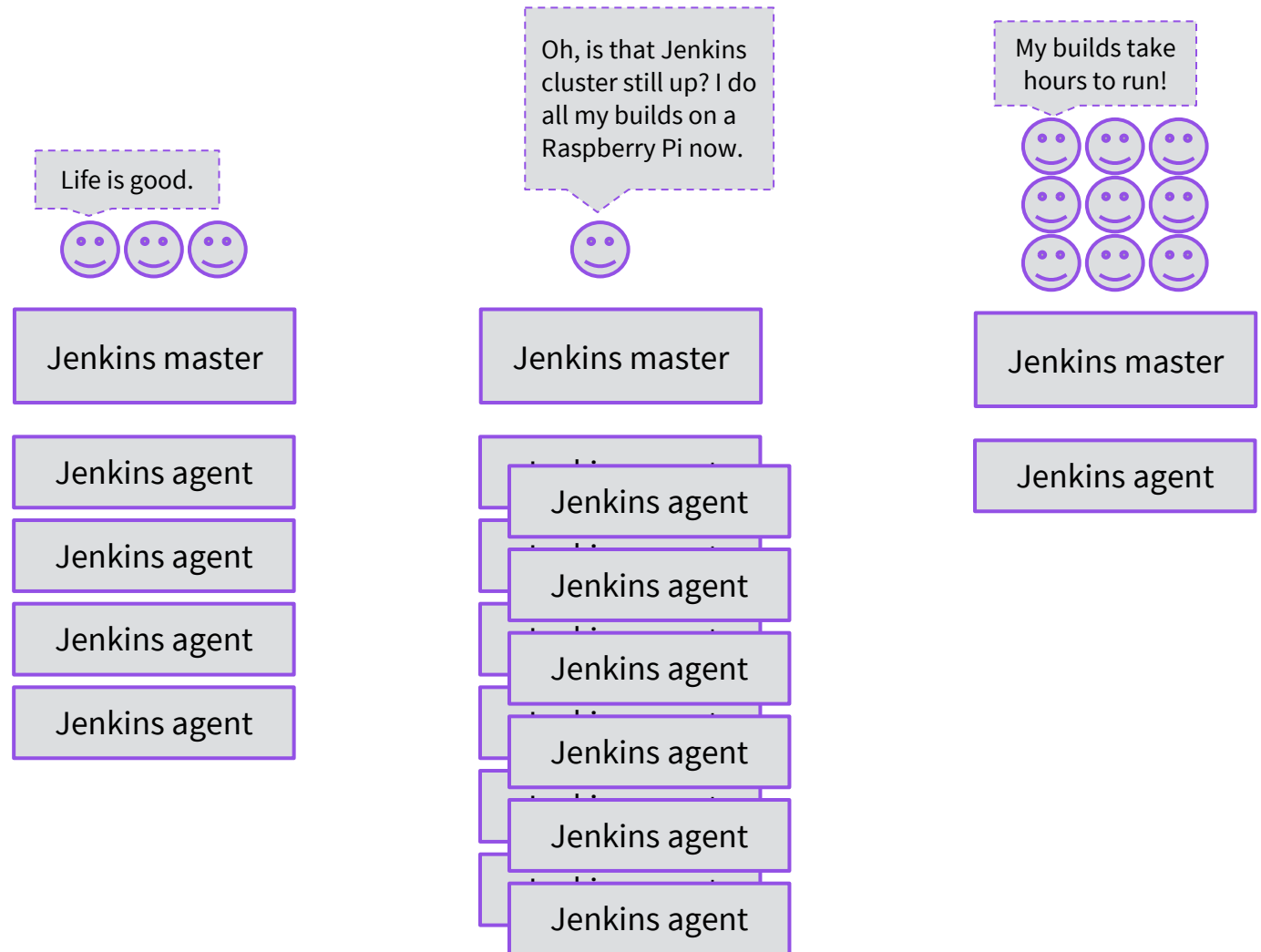
# WHEN IT BEGAN

Continuous  
Integration is soooo  
futuristic and this  
interface is beautiful.

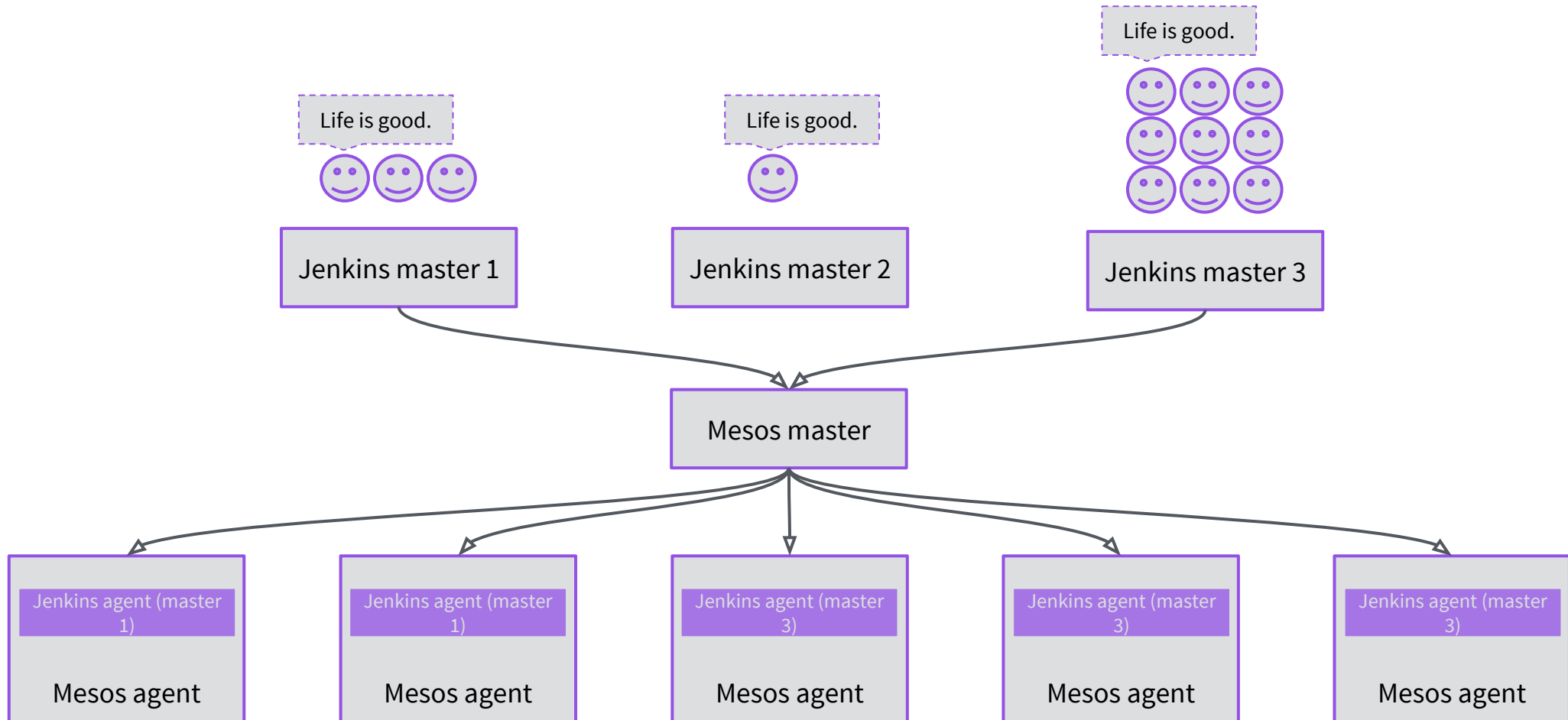


Jenkins master

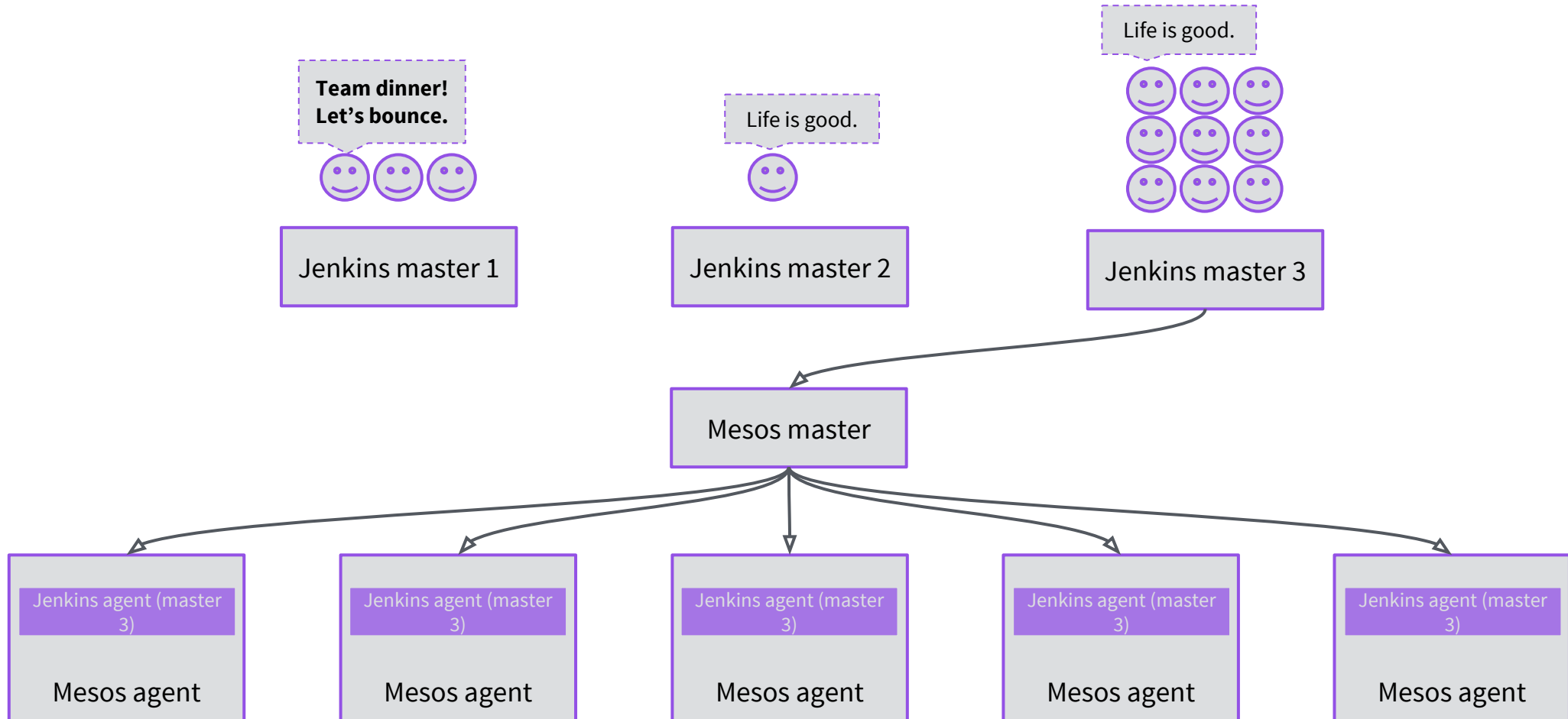
# THE OLD WORLD



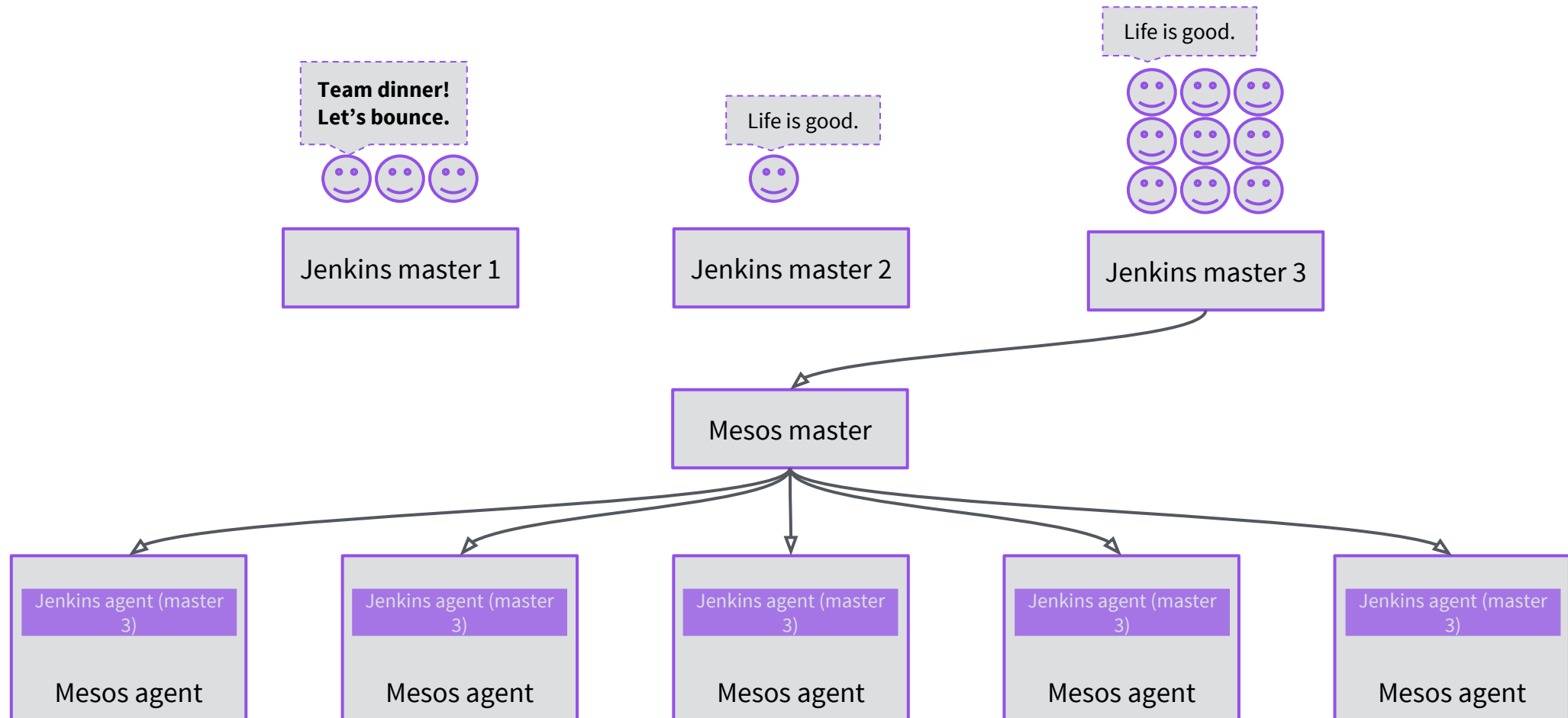
# JUST USE WHAT YOU NEED, WHEN YOU NEED IT AND SHARE THE LOVE RESOURCES



# JUST USE WHAT YOU NEED, WHEN YOU NEED IT AND SHARE THE LOVE RESOURCES

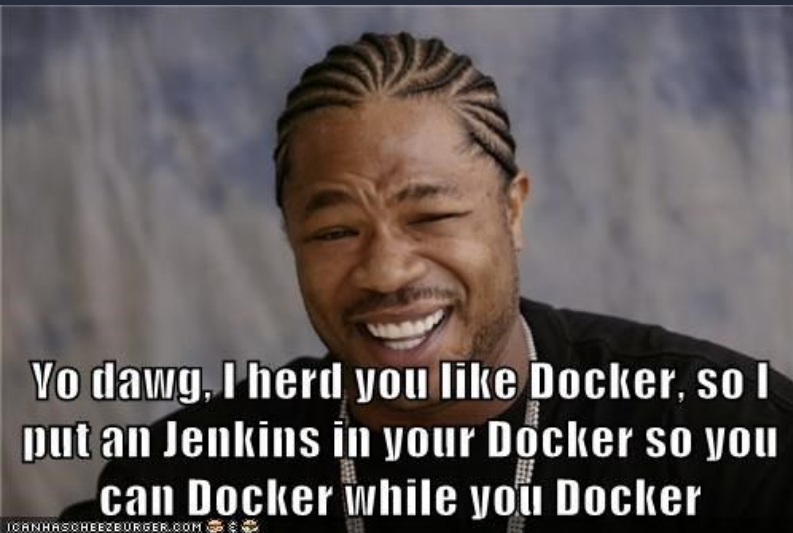


# JUST USE WHAT YOU NEED, WHEN YOU NEED IT AND SHARE THE LOVE RESOURCES



Jenkins on Mesos

# BUILDING DOCKER IN DOCKER: ONE WEIRD TRICK



This brave new world of containers running in containers has a bit of a whale and whale egg\* problem.

We run everything inside a container to make it easy to bundle dependencies and to isolate it from other processes.

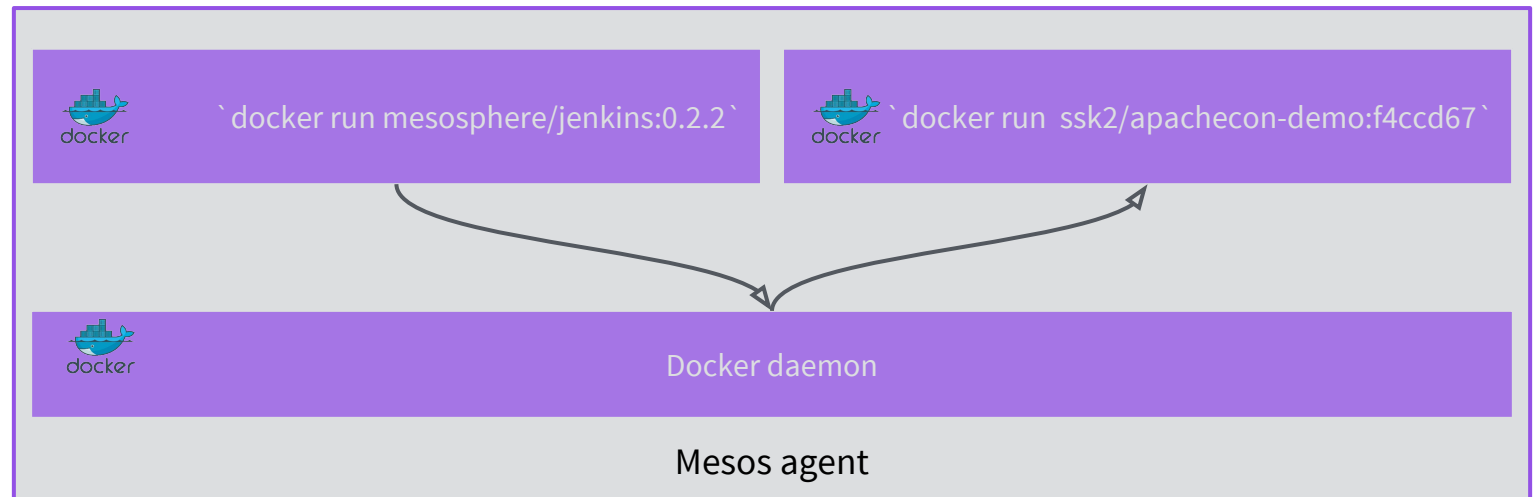
But when the thing that's running wants to build a container, what do you do?

\*Yes, I know.



# BUILDING DOCKER IN DOCKER: ONE WEIRD TRICK

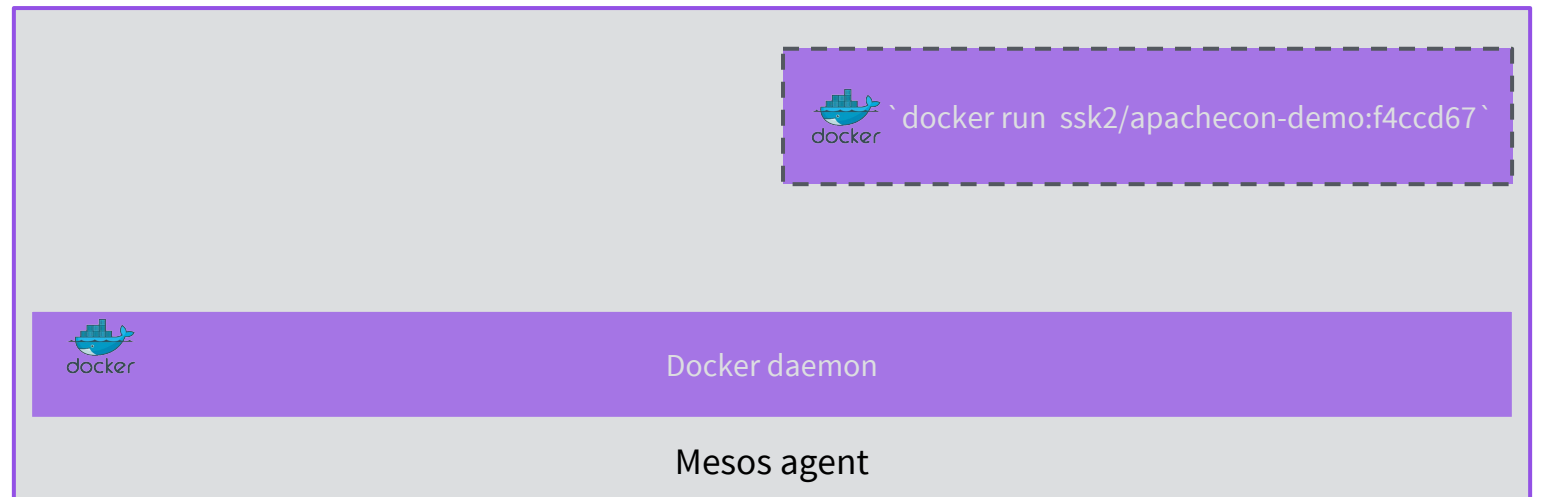
One recommended approach is to *bind mount* in the host system's Docker daemon.



# BUILDING DOCKER IN DOCKER: ONE WEIRD TRICK

This doesn't work for Mesos though! It doesn't track containers that it doesn't launch.

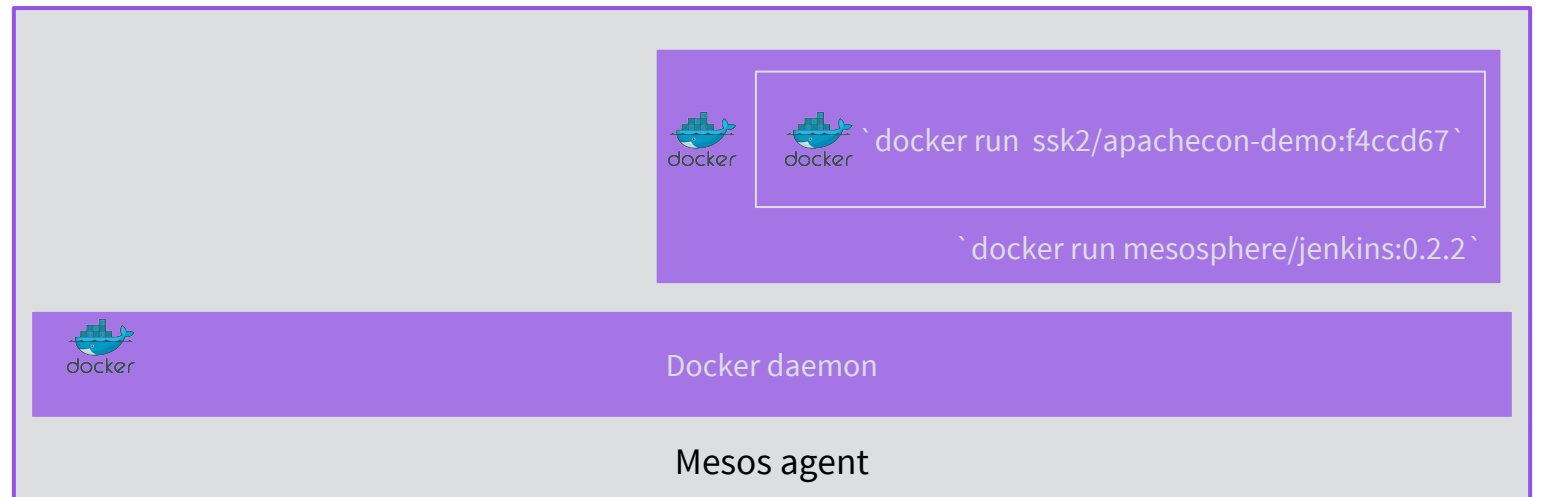
The sibling container becomes orphaned and runs forever.



# BUILDING DOCKER IN DOCKER: ONE WEIRD TRICK

Our solution is to use a customised Docker-in-Docker container.

This is a little slower but Mesos takes care of the resources!

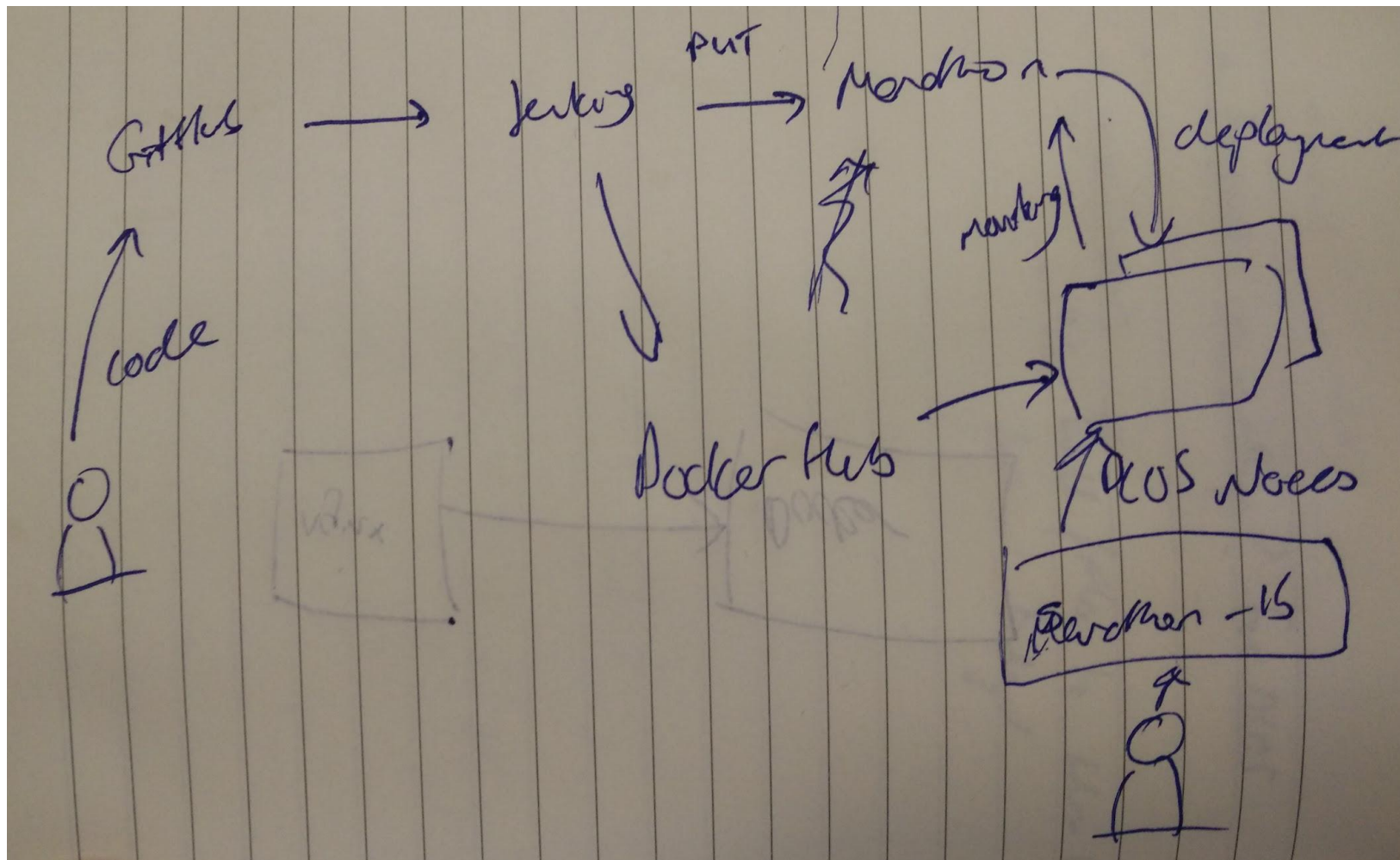


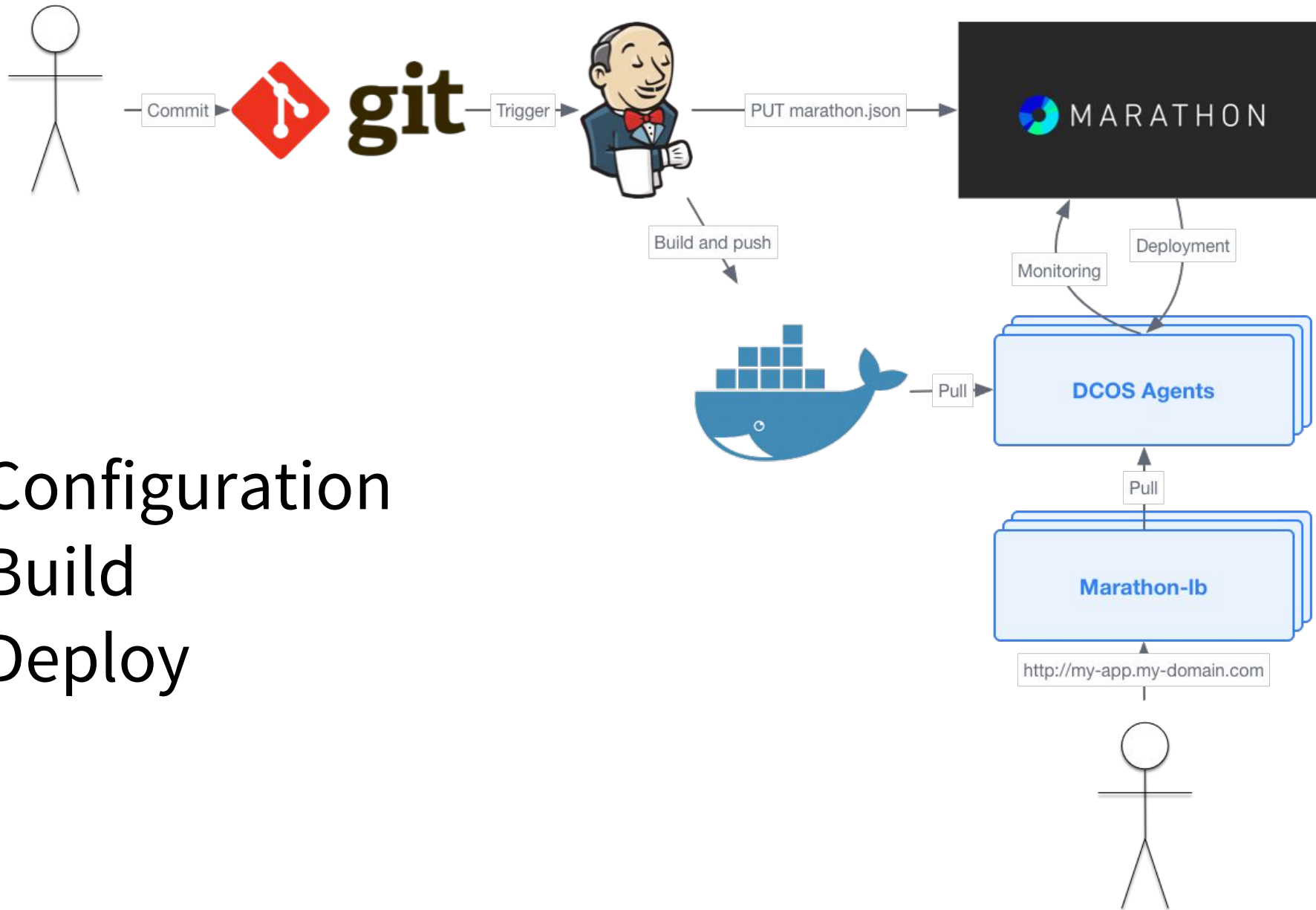
---

# CONTINUOUS DEPLOYMENT



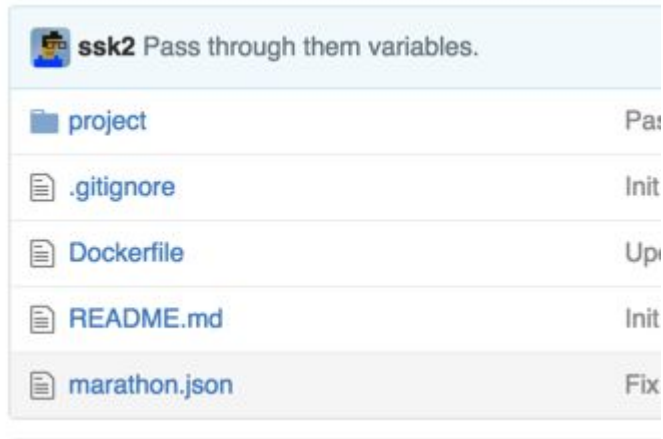
# PIPELINE: A FIRST PASS











1. Configuration
2. Build
3. Deploy

# 1. CONFIGURATION



 <b>ssk2</b> Pass through them variables.	
 project	Pa
 .gitignore	Init
 Dockerfile	Up
 README.md	Init
 marathon.json	Fix

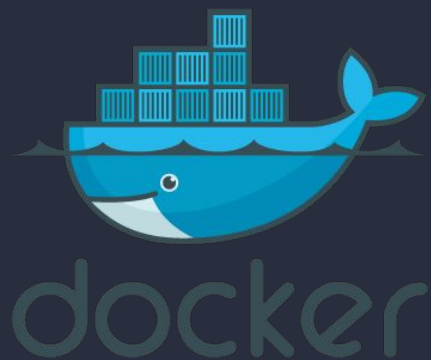
Building a CD pipeline requires configuration in a couple of places:

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



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!

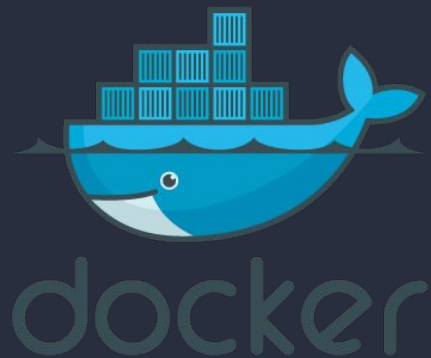


---

1. CONFIGURATION

# DEPENDENCY MANAGEMENT

```
FROM jekyll/jekyll  
ADD site /srv/jekyll
```



## 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.



## 1. CONFIGURATION

# APPLICATION CONFIGURATION

```
{
  "id": "apachecon-demo",
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "ssk2/apachecon-demo:latest",
      "network": "BRIDGE",
      "portMappings": [{
        "containerPort": 80,
        "protocol": "tcp"
      }]
    }
  },
  "labels": {
    "HAPROXY_0_VHOST": "sunil-889-publics1-781ifozh3z-1399492298.us-west-2.elb.amazonaws.com",
    "HAPROXY_GROUP": "external"
  },
  "instances": 1,
  "cpus": 0.1,
  "mem": 128
}
```



## 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 Jenkins:

1. Install the [Marathon plugin](#)
2. Save your Docker Hub credentials
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
```

## 2. BUILDING

Next, let's create a build:

1. Set up a build that polls GitHub periodically to build and push Docker image

```
docker login -u ${DOCKER_HUB_USERNAME} -p ${DOCKER_HUB_PASSWORD} -e sunil@mesosphere.com
```

```
docker build -t ssk2/apachecon-demo:${GIT_COMMIT} .
```

```
docker push ssk2/apachecon-demo:${GIT_COMMIT}
```

2. Add a Marathon post deploy step pointing to the DC/OS Marathon:
  - Set any variables you'd like to override.

## 3. DEPLOYING



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

# THANK YOU!

Come and talk to us!

- Email me at [sunil@mesosphere.io](mailto:sunil@mesosphere.io)
- Slides will be up at <http://mesosphere.github.io/presentations>
- Check out <https://dcos.io>

Other Mesosphere talks:

- 11:15am Elastic Hadoop Clusters on Mesos with Apache Myriad
- 2:30pm Stateful Services on Apache Mesos