

Summary of Mesos investigations

Giulio Eulisse

See also:

This is a summarised version of the **O2 CWG10 talk** I gave in the (Configuration and Control) context: <http://cern.ch/go/hT6Q>

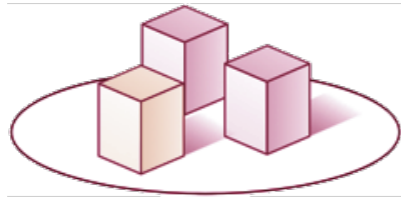
CHEP2016 presentation by Dario: <http://cern.ch/go/X9bC>

Our investigations so far

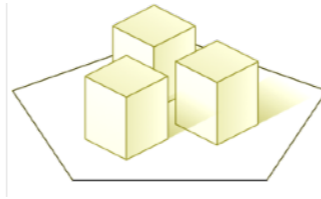
- ☑ Run the build infrastructure on top of Mesos
 - ☑ Understand how to install it
 - ☑ Understand how to operate it
- ☑ Understand how to extend Mesos to fit our custom use-cases:
 - ☑ Prototype of a Mesos DDS plugin
 - ☑ mesos-workqueue
- ☑ Understand how much we can rely on pre-existing solutions on top of Mesos:
 - ☑ Evaluate Mesosphere DC/OS
 - ☑ Evaluate Cisco Mantl



QA & analytics



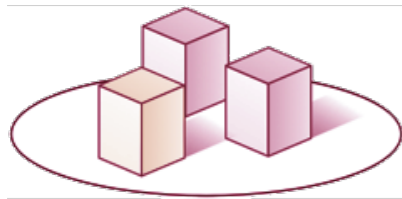
build cluster



release validation

Architectural shift from
statically partitioned
silos...

VAF



ALICE

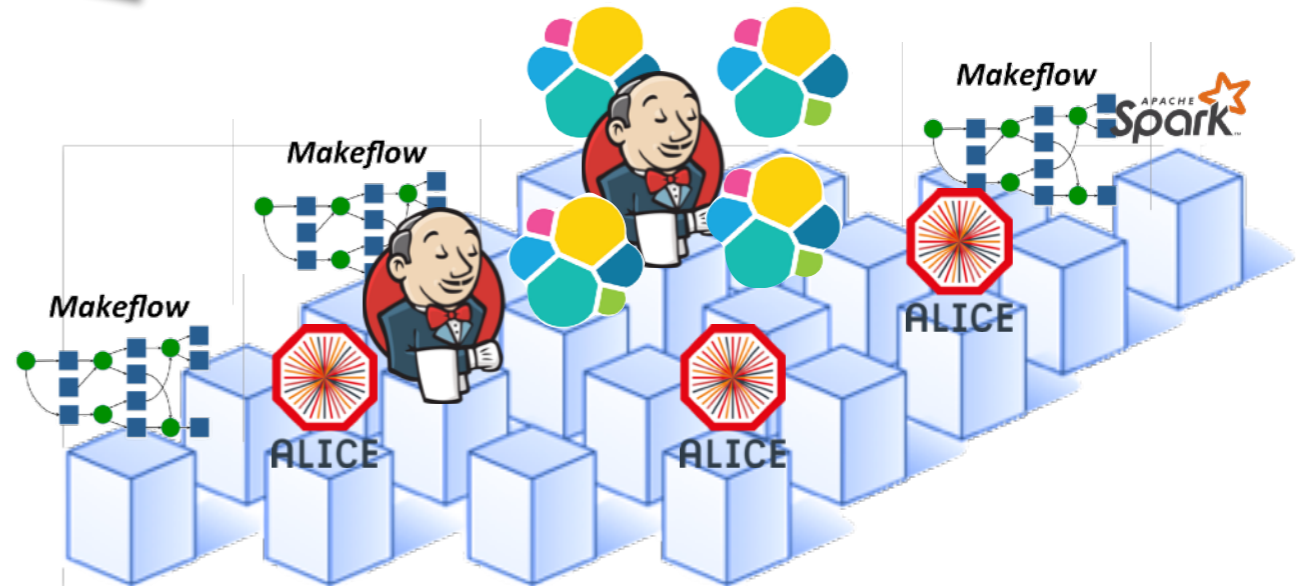
DAQ



ALICE



...to an **undifferentiated** set
of resources which
can be **reassigned**
dynamically depending on
load (**horizontal scalability**)
and failures (**high**
availability).



What is Mesos?

"Program against your datacenter like it's a single pool of resources"

A data-center kernel

Mesos is to the datacenter what the kernel is to the desktop: resource management and scheduler at large.

Apache Foundation project

Initially developed by AMPLab at UC Berkley [1], now under the Apache Foundation umbrella since a few years. Shares commonalities (and authors) with Google's known architecture papers [2].

Used in production by big players

Twitter, Apple, Netflix, AirBnB, Uber, NASA JPL and many others use Mesos in production [3]. Claimed to scale linearly to 10'000s of machines.

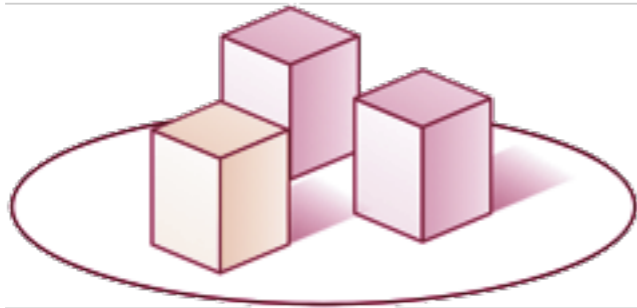
Large ecosystem

Thanks to the efforts of companies like Mesosphere, Cisco, Rancher, and many others a large ecosystem of pre-packaged solutions are available for most common "big data" tools (e.g. Spark, Elasticsearch).

Mesos Components

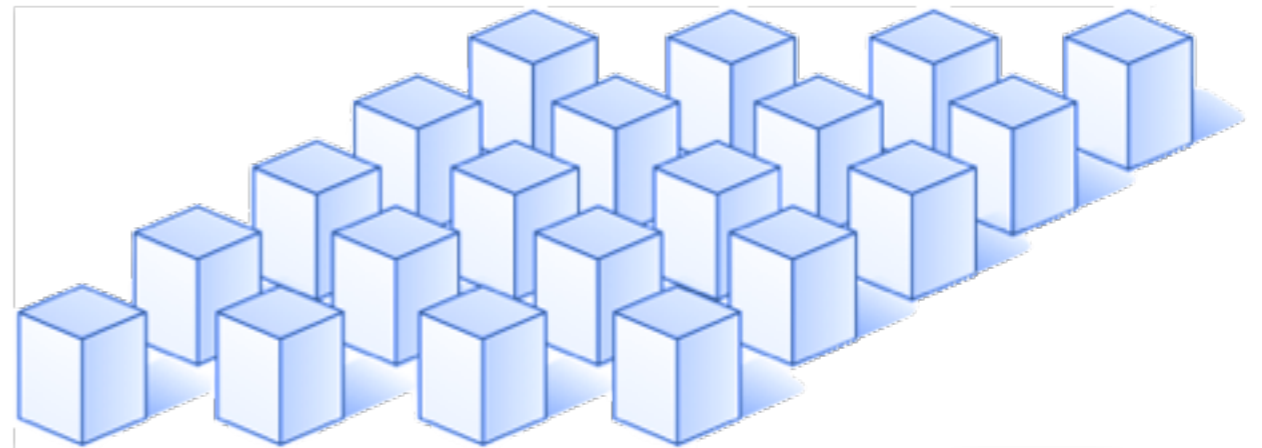
Masters

- Track application state
- Distribute resource offers
- Provide High Availability
- Accounting and authorisation
- Service discovery
- Monitoring / Debugging GUI



Agents

- "Cattle" like.
- Provide computational / storage resources
- Run (optionally containerised) applications
- Provide isolation



Frontends

- Routing
- Authentication



Mesos Features

High Availability

Zookeeper based ensemble which allows you to lose $(N-1)/2$ Masters before scheduler goes down. Agents can be restarted (or die) and can reconcile and adopt orphaned tasks.

Resource Management

Allocate CPU, memory, GPU, disk space to given tasks.

Resource Isolation

First class isolation support for CPU, memory, disk, ports, GPU, and modules for custom resource isolation.

APIs

Native C++ or HTTP REST based API for scheduling, operating the clusters (e.g. scheduled downtimes), monitoring.

Fine grained authentication and authorisation

E.g. only operators can delete volumes, only application X can access certain resources, application X has at least Y resources associated to it.

Containers

(Optional) native support for running containers. Can run Docker or AppC (without the need for their runtime).

How do I use Mesos?

Mesos by itself only fulfils the "resource manager" needs and provides a very raw monitoring / debugging GUI. In order to have a full blown solution one has three ways:

Generic OpenSource PaaS: *Using one of the several opensource "Platform as a Service" (PaaS) already available, e.g. Mesosphere's **Marathon** or Twitter's **Aurora**. We use Marathon at large in or build cluster and we started looking into Aurora.*

Write a custom application (Framework): this is what we did for the release validation, in order to integrate with the existing Workqueue code.

- C++ / Python / Java API (requires linking / JNI against libmesos.so and Protobuf)
- HTTP REST API (preferred choice nowadays, no direct dependency)

Ad hoc PaaS: *e.g. Apple J.A.R.V.I.S., or "**mesos-dds**" plug-in for DDS (<https://github.com/alisw/mesos-dds> kudos to Kevin).*

By design you are not forced to select one single solution. Mesos is based on the assumption that different problems require different solutions and makes sure **those solutions can coexist**.

Running builds on top of Mesos

In production since over 1 year, it gave us:

Platform abstraction

We keep doing production builds on slc5, while managing an slc7 / ubuntu infrastructure (thanks to docker support).

High availability

Builders are really "cattle" by now, we do not care if they get lost by CERN Openstack, migrated from one Hypervisor to another in 10 hours, or if Costin updates to the latest Ubuntu. ;-)

Resource sharing

Release validation and builds (and a few other services) happily coexists on the same set of machines.

Installing Mesos

By now I think I can deploy a Mesos cluster in less than a day, starting from 0...

Deploy with different tools:

- **Custom script:** *it is really a bunch of scp and ssh.*
- **Puppet:** *it is really needed for all the parts which need to integrate with CERN/IT services (e.g. SSO, DNS Load balancing, secrets storage).*
- **Ansible:** *It is IMHO the best of the tools. Given the Mesos configuration is actually small, even ansible "client mode" can deploy few hundreds of machines quickly.*
- **docker-compose:** *it is very good for play-testing on your laptop, see <https://github.com/alisw/ali-bot/blob/master/docker-compose.yml>.*

Docker very valuable to deploy on resources we do not control.

Future investigation: deploy via CVMFS? You would still need to customise the configuration on a per-machine basis, but in principle it could be done.

Operating Mesos

Operate as a Service

The idea is really that a Mesos cluster is provided to you as a service, like Condor or SWAN would. You can run your own Mesos setup on your laptop, but it's clearly not the way it's intended to be used.

Pick a pre-existing solution

Mesos by itself only provides primitives to write your own distributed architecture. Unless you have a specific use-cases, best option is to pick one or more of the already available "frameworks" (e.g. Mesosphere Marathon, the Mesos Jenkins plugin) and start building on top of it.

High availability

Everything in Mesos is thought with redundancy in mind and once you get used to it, anything which is not annoys you. A lot.

Rolling upgrades

Most / all changes can be done in a rolling upgrade mode. One can even update and restart the Mesos agent, without having to lose the actual job / service.

Operating Mesos: using Marathon

An off the shelf solution for long running jobs

Marathon is an "init.d for your cluster" by Mesosphere, a commercial company focused on providing a Mesos based solution for the datacenter (a.k.a. DC/OS).

- Especially thought for long running jobs (hence the name).
- Easy to use GUI + complete REST based API + command line client.
- Provides High Availability, support for stateful applications (e.g. MySQL databases), flexible constraints, service discovery, health checks, metrics, (optional) Docker support.
- Used in the Offline Build and QA cluster (250 cores, 20 machines)

Marathon: Top level view

The screenshot shows the Marathon web interface at `alimarathon.cern.ch`. The interface is divided into a left sidebar and a main content area. The sidebar contains filters for STATUS (Running: 28, Deploying, Suspended: 3, Delayed, Waiting), HEALTH (Healthy: 15, Unhealthy, Unknown: 13), LABEL (Select), and RESOURCES (Volumes). The main content area is titled "Applications" and features a search bar and two buttons: "Create Group" and "Create Application". Below this is a table listing various applications with their respective CPU, Memory, Status, Running Instances, and Health indicators.

Name	CPU	Memory	Status	Running Instances	Health
airflow	0.0	0 B	Suspended	0 of 0	...
alice-github <small>traefik.enable:true</small>	0.1	1 GiB	Running	1 of 1	...
chronos	1.0	1 GiB	Running	1 of 1	...
coverage-reports <small>traefik.enable:true</small>	0.1	1 GiB	Running	1 of 1	...
elasticsearch-client	2.0	2 GiB	Running	2 of 2	...
elasticsearch-data	10.0	20 GiB	Running	5 of 5	...
gh-mysql	0.1	2 GiB	Running	1 of 1	...
hubot	0.0	0 B	Suspended	0 of 0	...
hubot-dev	0.1	512 MiB	Running	1 of 1	...
hubot-prod	0.1	512 MiB	Running	1 of 1	...
igprof-io <small>traefik.enable:true</small>	0.1	1 GiB	Running	1 of 1	...

Marathon: fine grained resources

The screenshot displays the Marathon web interface with a table of applications. A blue arrow points to the resource specifications of the 'coverage-reports' application, which are circled in red. The table columns are Name, CPU, Memory, Status, Running Instances, and Health. The 'coverage-reports' application is running with 0.1 CPU and 1 GiB memory.

Name	CPU	Memory	Status	Running Instances	Health
airflow	0.0	0 B	Suspended	0 of 0	
alice-github	0.1	1 GiB	Running	1 of 1	
chronos	1.0	1 GiB	Running	1 of 1	
coverage-reports	0.1	1 GiB	Running	1 of 1	
elasticsearch-client		2 GiB	Running	2 of 2	
elasticsearch-data		20 GiB	Running	5 of 5	
gh-mysql	0.1	2 GiB	Running	1 of 1	
hubot	0.0	0 B	Suspended	0 of 0	
hubot-dev	0.1	512 MiB	Running	1 of 1	
hubot-prod	0.1	512 MiB	Running	1 of 1	
igprof-io	0.1	1 GiB	Running	1 of 1	

Marathon: health checks

The screenshot displays the Marathon web interface. On the left, there are filters for STATUS (Running: 28, Deploying, Suspended: 3, Delayed, Waiting), HEALTH (Healthy: 15, Unhealthy, Unknown: 13), LABEL (Select), and RESOURCES (Volumes). The main area shows a table of applications with columns for Name, CPU, Memory, Status, Running Instances, and Health. A blue callout box points to the Health column, stating: "A given endpoint of the application can be used to retrieve health conditions and act accordingly". The 'alice-github' application's health bar is circled in red.

Name	CPU	Memory	Status	Running Instances	Health
airflow				0 of 0	
alice-github				1 of 1	Healthy
chronos				1 of 1	Healthy
coverage-reports			Running	1 of 1	Healthy
elasticsearch-client	2.0	2 GiB	Running	2 of 2	Healthy
elasticsearch-data	10.0	20 GiB	Running	5 of 5	Healthy
gh-mysql	0.1	2 GiB	Running	1 of 1	Healthy
hubot	0.0	0 B	Suspended	0 of 0	Unhealthy
hubot-dev	0.1	512 MiB	Running	1 of 1	Healthy
hubot-prod	0.1	512 MiB	Running	1 of 1	Healthy
igprof-io	0.1	1 GiB	Running	1 of 1	Healthy

Marathon: configuring application through the GUI

The screenshot shows the Marathon GUI interface. A modal window titled "Edit Application" is open, displaying configuration options for an application named "elasticsearch-data". The modal has a blue header with the title and a "JSON Mode" toggle. The left sidebar of the modal lists various configuration sections: General, Docker Container, Ports, Environment Variables, Labels, Health Checks, Volumes, and Optional. The "General" section is currently selected. The configuration fields are as follows:

- ID:** /elasticsearch-data
- CPUs:** 2
- Memory (MiB):** 4000
- Disk Space (MiB):** 0
- Instances:** 5
- Command:**

```
bash /docker-entrypoint.sh -Des.node.name=$HOSTNAME -Des.node.master=false -  
Des.node.data=true -Des.http.enabled=false -  
Des.discovery.zen.ping.unicast.hosts=alimesos01.cern.ch:9300,alimesos02.cern.ch:93  
00,alimesos03.cern.ch:9300
```

Below the command field, there is a note: "May be left blank if a container image is supplied".

In the background, the main interface shows the application is "Running" with 5 instances. There are buttons for "Scale Application" and "Refresh".

Marathon: fine grained task details

The screenshot shows the Marathon web interface for the application 'elasticsearch-data'. The application is in a 'Running' state with 5 instances, all of which are 'Healthy' (100%). A large blue arrow points to the 'Instances' tab, which displays a table of task details. The table includes columns for ID, Health, Status, Error Log, Output Log, Version, and Updated. The 'Health' column shows 'Healthy' for all instances, and the 'Status' column shows 'Started' for all instances. The 'Updated' column shows the last update time for each instance, ranging from 4 August 2016 at 12:14:29 GMT+2 to 13:53:59 GMT+2.

elasticsearch-data
Running (5 of 5 instances)
5 Healthy (100%)

Instances Configuration Debug

Refresh

ID	Health	Status	Error Log	Output Log	Version	Updated
elasticsearch-data.387ac73f-5a2c-11e6-b2aa-56847afe9799 alibuild12.cern.ch:31763	Healthy	Started	stderr	stdout	4 months ago	4 August 2016 at 12:14:29 GMT+2
elasticsearch-data.752fbe69-5a2d-11e6-b2aa-56847afe9799 alibuild13.cern.ch:31287	Healthy	Started	stderr	stdout	4 months ago	4 August 2016 at 12:23:22 GMT+2
elasticsearch-data.a7fbd8f2-5a37-11e6-b2aa-56847afe9799 alibuild14.cern.ch:31963	Healthy	Started	stderr	stdout	4 months ago	4 August 2016 at 13:36:24 GMT+2
elasticsearch-data.1d3dc8a2-5a3a-11e6-b2aa-56847afe9799 alibuild10.cern.ch:31622	Healthy	Started	stderr	stdout	4 months ago	4 August 2016 at 13:53:59 GMT+2

Marathon: scaling applications with a click

The screenshot shows the Marathon web interface for the application 'elasticsearch-data'. The application is in a 'Running' state with 5 instances, all of which are 'Healthy'. A 'Scale Application' dialog is open, asking 'How many instances would you like to scale to?' with a spinner control set to 5. The background shows a table of instances with columns for ID, Health, State, Logs, and Update Time.

Scale Application

How many instances would you like to scale to?

5

Scale Application Cancel

ID	Health	State	Logs	Updated
elasticsearch-data.387ac73f-5a2c-11e6-b2aa-56847afe9799 alibuild12.cern.ch:31763	Healthy	Started	stderr stdout	4 August 2016 at 12:14:29 GMT+2
elasticsearch-data.752fbe69-5a2d-11e6-b2aa-56847afe9799 alibuild13.cern.ch:31287	Healthy	Started	stderr stdout	4 months ago 4 August 2016 at 12:23:22 GMT+2
elasticsearch-data.a7fbd8f2-5a37-11e6-b2aa-56847afe9799 alibuild14.cern.ch:31963	Healthy	Started	stderr stdout	4 months ago 4 August 2016 at 13:36:24 GMT+2
elasticsearch-data.1d3dc8a2-5a3a-11e6-b2aa-56847afe9799 alibuild10.cern.ch:31622	Healthy	Started	stderr stdout	4 months ago 4 August 2016 at 13:53:59 GMT+2

Apache Aurora

What Twitter runs

Aurora is the PaaS developed at Twitter (now Open Source) to run all of their workloads. Oldest Mesos based project to my knowledge. They disclosed it scales for their 250,000 containers running on 30,000 nodes.*

- *Both for **long-running** and **cron jobs***
- *Tasks described in a Python based DSL. Command-line based interaction. Monitoring GUI.*
- ***Multi role / environment support**, with associated quotas and preemption. E.g. online / offline role, development / production environment.*
- *Supports multiple deployment updates strategies (e.g. canary)*

*: <https://youtu.be/FU7wrqsRj3o?t=21m11s>

Apache Aurora: toplevel view

The screenshot shows the Apache Aurora web interface. At the top, there is a browser window with the URL `aidrefpc001:8081/scheduler/root`. Below the browser, the Apache Aurora logo and the text "updates" are visible. The main content area shows the breadcrumb "o2 / root" and a section titled "Resource consumption".

	CPU	RAM	Disk
Quota	0 core(s)	0.00 MiB	0.00 MiB
Quota Consumption	0 core(s)	0.00 MiB	0.00 MiB
Production Dedicated Consumption	0 core(s)	0.00 MiB	0.00 MiB
Non-Production Consumption	2 core(s)	2.00 MiB	16.00 MiB
Non-Production Dedicated Consumption	0 core(s)	0.00 MiB	0.00 MiB

Below the resource consumption table, there is a search bar and a table of jobs. The search bar contains the text "Search :". The job table has the following columns: Job Type, Environment, Job, production, Pending Tasks, Active Tasks, Finished Tasks, and Failed Tasks.

Job Type	Environment	Job	production	Pending Tasks	Active Tasks	Finished Tasks	Failed Tasks
service	prod	hello_world		0	1	0	0
service	devel	hello_world		0	1	2	4

At the bottom of the job list, there is a pagination control with buttons for "Previous", "1", and "Next". The "1" button is highlighted, indicating the current page.

Apache Aurora: task updates view

The screenshot shows the Apache Aurora web interface. At the top, there's a browser window with the URL `aidrefpc001:8081/scheduler/root/devel/hello_world/update/559a7e83-8e5d-4162-8a3e-ad18bc62bae0`. The page header includes the Apache Aurora logo and the word "updates". Below the header is a breadcrumb trail: `o2 / root / devel / hello_world / 559a7e83-8e5d-4162-8a3e-ad18bc62bae0`.

The main content area is divided into several sections:

- Update Summary:** Shows the update was started by `UNSECURE` and has a final status of `SUCCESS`. It displays a timeline from `THU, AUG 25TH 17:08` to `THU, AUG 25TH 17:09`, both times 12 minutes ago. A wavy line indicates the duration, which is noted as "DURATION: A MINUTE".
- Update Settings:** A table listing various settings and their values.
- INSTANCE STATUS:** Shows `0` instances in a green box, with a total of `1 / 1 (100%)`.
- Update Events:** A table with columns for event, time, user, and message.
- Instance Events:** A table with columns for instance, event, and time.

At the bottom, there's a pagination control showing "21" and buttons for "Previous", "1" (selected), and "Next".

Update Summary

STARTED BY `UNSECURE`
FINAL STATUS `SUCCESS`

THU, AUG 25TH

17:08

12 MINUTES AGO



THU, AUG 25TH

17:09

12 MINUTES AGO

DURATION: A MINUTE

Update Settings

setting	value
UPDATE GROUP SIZE	1
MAX FAILURES PER INSTANCE	0
MAX FAILED INSTANCES	0
MIN WAITING TIME IN RUNNING	45000
ROLLBACK ON FAILURE	true

INSTANCE STATUS

0 1 / 1 (100%)

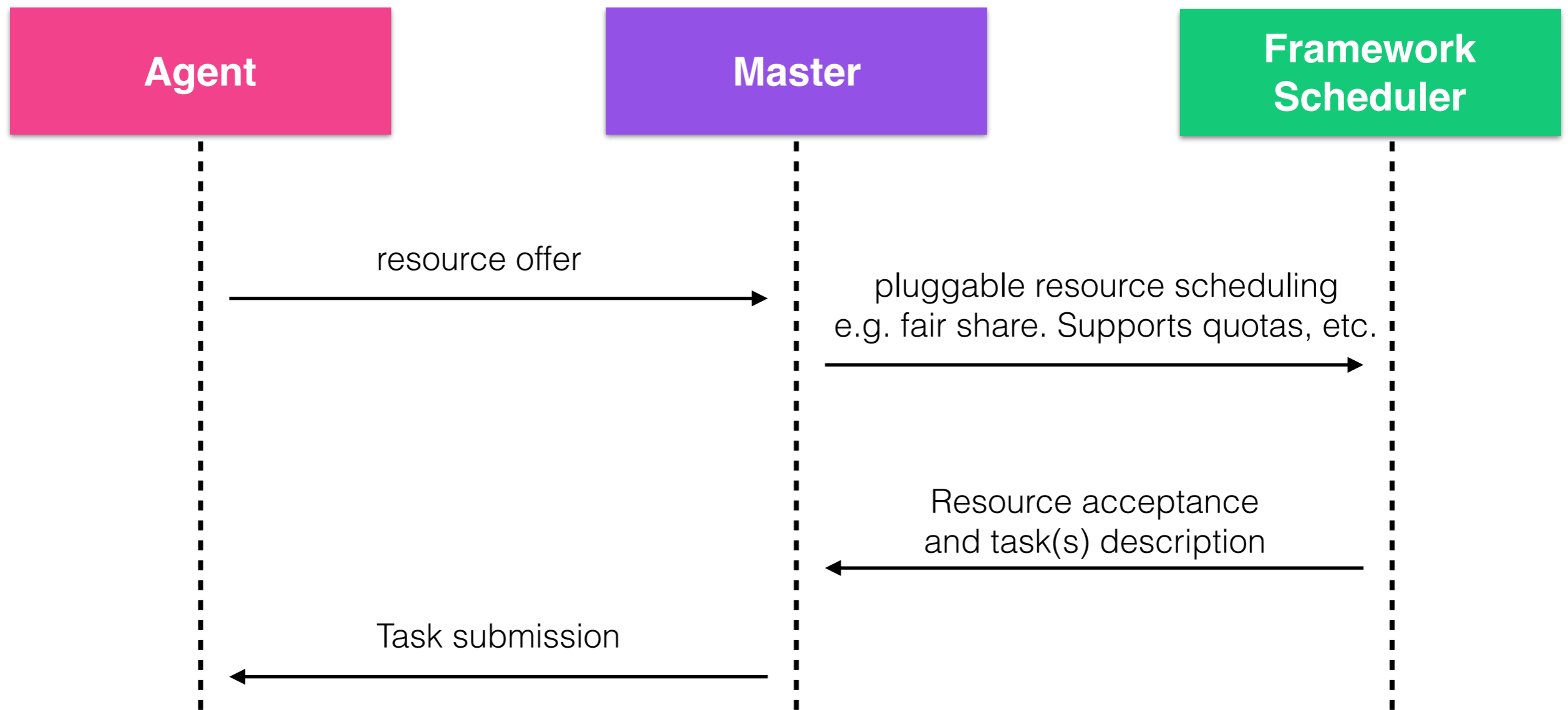
Update Events

event	time	user	message
IN PROGRESS	12 minutes ago	UNSECURE	
SUCCESS	12 minutes ago		

Instance Events

instance	event	time
<code>0</code>	UPDATING	12 minutes ago
<code>0</code>	UPDATED	12 minutes ago

Mesos difference: application aware



Mesos is a **two level scheduler**. The first level being some generic resource scheduling done by the master, the second level being the application (Framework) itself taking finer grained, **application-aware**, decisions.

Adapting Mesos to our specific needs

Why writing your own framework?

Don't: Think twice before writing your own framework, *writing a Mesos framework is easy. Writing a good Mesos framework is hard and one should have a pretty good reason for doing so.*

Automatic scaling: *your applications needs to scale up and down quickly as response to its own internal state.*

Dynamic resource management: *your applications has the need to allocate / deallocate resources (e.g. storage) as part of its life-cycle.*

Complex task placement: *e.g. your application needs to run only once on every node of the cluster every day, but only when the machine is idle.*

Interface with your own scheduler: *you already have a working scheduler and you want to share resources with others, but you do not want to give up ownership of the scheduling to a third party.*

Mesos Workqueue

Integrate legacy system into new one

Old release validation used workqueue, so we decided to keep things unchanged and plug workqueue on top of Mesos.

<https://github.com/alisw/mesos-workqueue>

Total work to do it, roughly 1 week, 362 C++ SLOC. Nothing particularly smart, but it works and runs in production.

Scales dynamically

Depending on the needs of the release validation it can dynamically scale workers.

Missing features

Task reconciliation (if the master dies, all the workers do), high availability.

DDS Mesos

DDS plugin to use Mesos as a RMS

By design DDS does not do any resource management, but it delegates it to external Resource Management Systems. Mesos is a perfect fit there.

<https://github.com/alisw/mesos-dds>

Work done by Kevin Napoli (University of Malta) for his master thesis.

Client - Server architecture

By separating the part which talks to mesos to a separate server-side component, allows multiple DDS to share the same Mesos cluster.

Future developments

Resource constrains, e.g. run task on a node which has a GPU (was discussed in the last DDS workshop). Integrate with Aurora to get all the nice multi-user / multi-tier features?

End-to-End Mesos solutions

Mesosphere DC/OS

A Mesos Distribution

End to end solution for Mesos. If Mesos is the kernel and a Framework is an application, DC/OS is a Linux distribution. Developed by Mesosphere, recently OpenSourced (Apache Licensed).

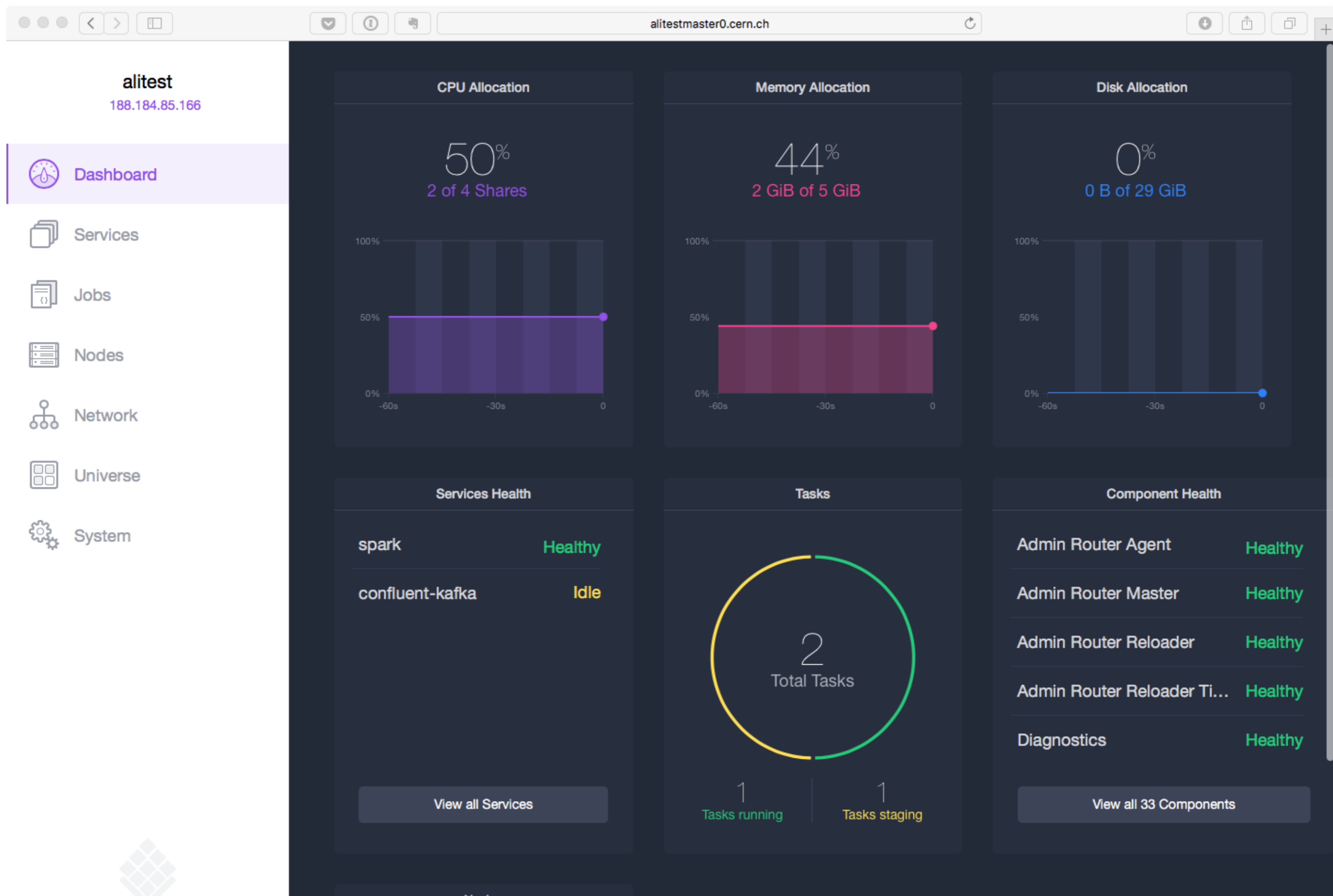
Brainless Mesos Setup

Provides easy installation, user management, prepackaged applications, nicer monitoring GUI, CLI and fully integrated GUI. Built around Marathon.

Too brainless?

DC/OS is extremely nice to quickly deploy and operate well known tools like Hadoop, Spark, Storm. Not clear what is the added value w.r.t. Marathon / Aurora, if you do not need those, and you need to package your own tools.

Mesosphere DC/OS: dashboard



Mesosphere DC/OS: cluster monitoring

The screenshot displays the Mesosphere DC/OS cluster monitoring interface. The browser address bar shows the URL `alitestmaster0.cern.ch/#/nodes/list/`. The left sidebar contains navigation options: Dashboard, Services, Jobs, Nodes (selected), Network, Universe, and System. The main content area features a 'CPU Allocation Rate' chart for 2 total nodes, showing a constant 50% allocation rate over a 60-second period. Below the chart, a summary indicates '2 Nodes' with 2 healthy and 0 unhealthy nodes. A table lists the nodes with their hostnames, health status, task counts, and resource usage (CPU, Memory, Disk).

alitest
188.184.85.166

Dashboard
Services
Jobs
Nodes
Network
Universe
System

CPU Allocation Rate
2 Total Nodes

CPU Memory Disk

100%
80%
60%
40%
20%
0%

-60s -40s -20s 0

2 Nodes

Filter All **2** Healthy **2** Unhealthy **0** Filter by Service List Grid

HOSTNAME	HEALTH ▲	TASKS	CPU	MEM	DISK
188.184.85.189	Healthy	1	50%	40%	0%
188.184.87.12	Healthy	1	50%	47%	0%

Mesosphere DC/OS: GUI based installation

The screenshot shows the Mesosphere DC/OS GUI interface. The browser address bar displays 'alitestmaster0.cern.ch'. On the left sidebar, the 'Universe' menu item is selected. The main content area is titled 'Packages' and shows a list of available software packages. Each package entry includes an icon, the package name, its version, and an 'Install' button. The 'etcd' package is highlighted with a grey background.

Package Name	Version	Action
ecr-login	1.0.0	Install
elasticsearch	1.0.1	Install
etcd	0.0.3	Install
exhibitor	1.0.0	Install
gestalt-framework	0.1.0	Install
hdfs	0.9.0-2.6.0	Install
hue	0.0.1	Install
kafka	1.1.9-0.10.0.0	Install
kafka-manager	1.3.0.8	Install
kibana	4.5.3	Install

CISCO mantl.io

- Similar concept as DC/OS: a "curated" Mesos distribution.
- Works with Centos / Ubuntu
- Can provision servers to OpenStack (albeit CERN/IT setup is a cumbersome), using Hashicorp Terraform [4].
- Deploys configuration via RedHat Ansible [5].
- Tested by Kevin Napoli during his Summer Studentship, not particularly impressed, compared to DC/OS.

mantl.io

The screenshot shows a web browser window with the URL `mantl-control-01.cern.ch`. The page header includes the Mantl logo, navigation links for Home and Health, and a status indicator showing a green dot and the text "Passing". A "Reload Services" button is located in the top right corner of the main content area.

The main content area features five service cards, each with a logo, name, "Web UI" button, and "Checks: Passing" status:

- Mesos**: Logo is a blue and dark blue geometric pattern. Status: Checks: Passing.
- Marathon**: Logo is a circular ring with blue, cyan, and green segments. Status: Checks: Passing.
- Kubernetes**: Logo is a blue ship's wheel. Status: Checks: Passing.
- Consul**: Logo is a purple circular shape with dots. Status: Checks: Passing.
- Traefik**: Logo is the text `/'Træfik/` in a stylized font. Status: Checks: Passing.

Reload Health Checks

consul

docker

etcd

kubernetes

marathon

mesos

traefik-admin

vault

zookeeper

● consul

Serf Health Status

Status	Check ID	Node	Service ID	Service Name
Passing	serfHealth	mantl-worker-001		

Output:

```
Agent alive and reachable
```

consul Distributive Mantl Health Checks

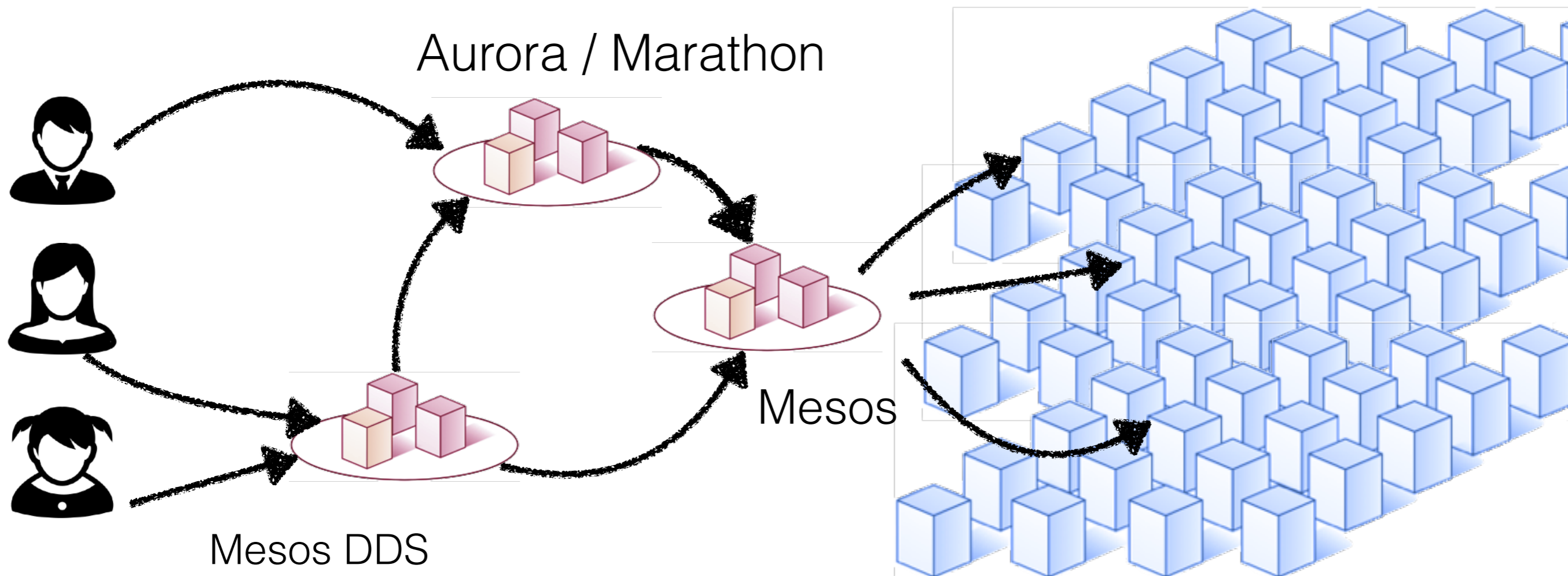
Status	Check ID	Node	Service ID	Service Name
Passing	distributive-consul-checks	mantl-worker-001		

Output:

```
time="2016-08-18T14:38:22Z" level=info msg="Creating checklist(s)..." path="/etc/distributive
time="2016-08-18T14:38:22Z" level=info msg="Running check DockerRunningRegexp"
time="2016-08-18T14:38:22Z" level=info msg="Running check Directory"
time="2016-08-18T14:38:22Z" level=info msg="Running check File"
time="2016-08-18T14:38:22Z" level=info msg="Running check Installed"
time="2016-08-18T14:38:22Z" level=info msg="Running check File"
time="2016-08-18T14:38:22Z" level=info msg="Running check File"
time="2016-08-18T14:38:22Z" level=info msg="Running check File"
time="2016-08-18T14:38:22Z" level=info msg="Running check File"
time="2016-08-18T14:38:22Z" level=info msg="Running check File"
time="2016-08-18T14:38:22Z" level=info msg="Running check File"
time="2016-08-18T14:38:22Z" level=info msg="Report from checklist" check
```

What's next?

Provide a demonstrator for a **multi-user, multi-tier** cluster which can be used to deploy tasks via DDS (e.g. O2 Devices) or Aurora / Marathon (e.g. cronjobs, non O2 services).



References

- [1]: Mesos: <https://people.eecs.berkeley.edu/~alig/papers/mesos.pdf>
- [2]: Omega: flexible, scalable schedulers for large compute clusters <http://research.google.com/pubs/pub41684.html>
- [3] Powered by Mesos: <http://mesos.apache.org/documentation/latest/powered-by-mesos/>
- [4]: Terraform: <https://www.terraform.io>
- [5]: Ansible: <https://www.ansible.com>