v1.11

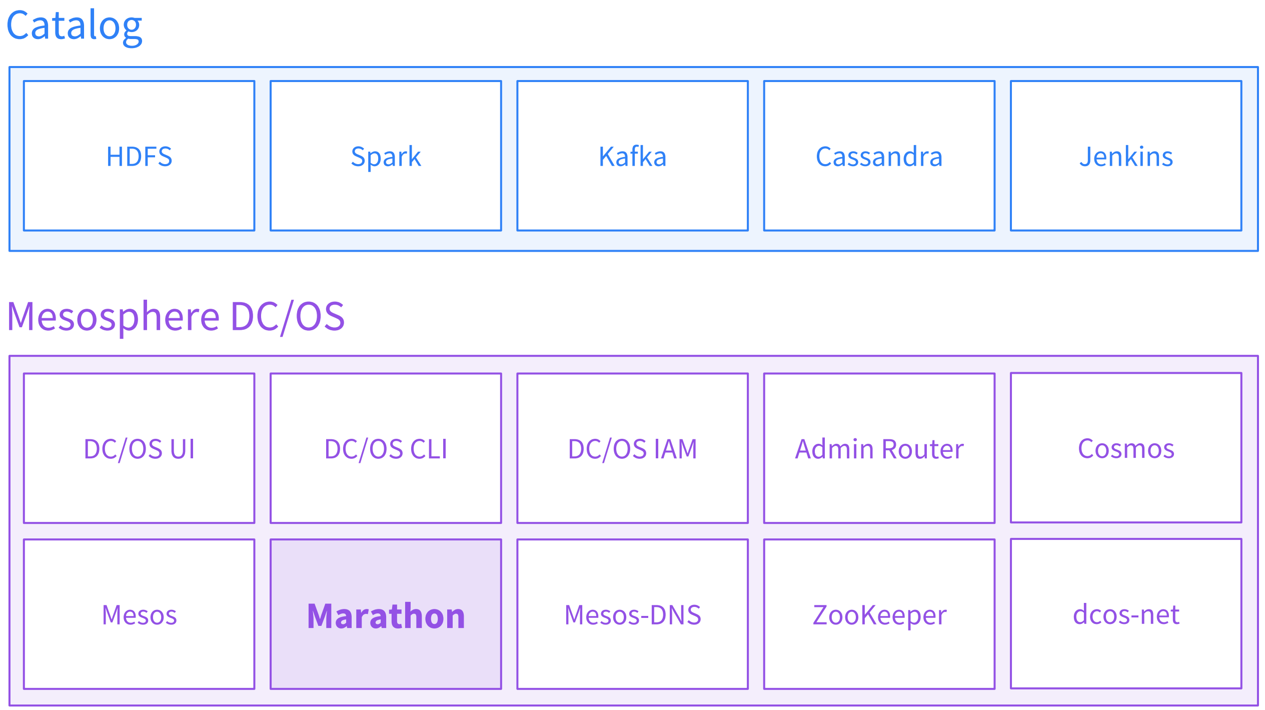
MARATHON

MESOSPHERE

AGENDA

1. Overview
2. Application definition
3. Containers
4. Constraints
5. Health checks
6. App groups
7. Pods
8. Deployment policies
9. Stateful Services

DC/OS COMPONENTS

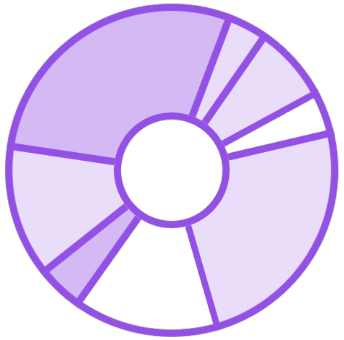


Marathon

OVERVIEW

OVERVIEW

- Marathon is a DC/OS service for long-running services such as:
  - web services
  - application servers
  - databases
  - API servers
- Marathon is not a Platform as a Service (PaaS), it provides a RESTful API which lets you build your own PaaS on top of Marathon
- Also called a ‘meta-framework’



FUNCTIONAL ORCHESTRATION CAPABILITIES

SCHEDULING

- Placement
- Replication/Scaling
- Resurrection
- Rescheduling
- Rolling Deployment
- Upgrades
- Downgrades
- Collocation

RESOURCE MANAGEMENT

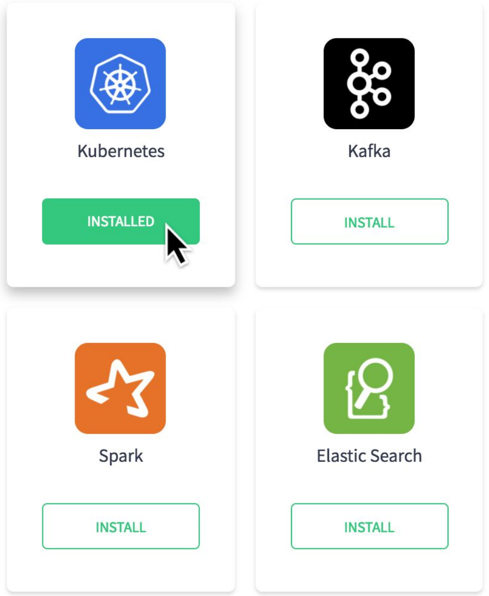
- Memory
- CPU
- GPU
- Volumes
- Ports
- IPs
- Images/Artifacts

SERVICE MANAGEMENT

- Labels
- Groups/Namespaces
- Dependencies
- Load Balancing
- Readiness Checking

WHAT IS A SERVICE?

- Frameworks -> DC/OS Services
  - Scheduler managed by Marathon
  - Command or Custom Executor
- Marathon Apps -> Services
- https://mesosphere.com/service-catalog



Marathon

SERVICES (MARATHON APP)

SERVICES

- A unit of deployment in Marathon
- Specified using JSON
- App definition has a few required fields, but most are optional
- App definitions match what the Marathon API `/v2/apps` endpoint returns (with some exceptions)
- REST API
  - Marathon has a powerful API that can be used to build your own PaaS
  - https://mesosphere.github.io/marathon/docs/generated/api.html

BASIC PARAMETERS

Name	Description
id	Unique identifier for the app consisting of a series of names separated by slashes
cpus	The number of CPUs this application needs per instance. This number does not have to be integer, but can be a fraction
mem	The amount of memory in MB that is needed for the application per instance
cmd	The command that is executed. This value is wrapped by Mesos via /bin/sh -c ${app.cmd}. Either cmd or container must be supplied
container	Additional data passed to the containerizer on application launch. These consist of a type, zero or more volumes, and additional type-specific options
instances	The number of instances of this application to start

ADDITIONAL PARAMETERS

Name	Description
uris	(Array) List of URIs that will be fetched, cached, and optionally extracted by Mesos Protocols: `file:///`,`http://`,`https://`,`ftp://`,`ftps://`,`hdfs://`,`s3://`,`s3a://`,`s3n://` Extracts: `tgz`,`.tar.gz`,`.tbz2`,`.tar.bz2`,`.txz`,`.tar.xz`,`.zip`
env	(Object) Set of key-value pairs which are passed to the app as environment variables
constraints	(Array) List of app constraints
acceptedResourceRoles	(Array) List of roles from which offers will be accepted for this app
labels	(Object) Arbitrary set of key-value pairs which can be used to tag an application
healthChecks	(Array) List of health checks to perform on the app

EXAMPLE APPLICATION JSON

```
{
  "id": "/hello-world",
  "cpus": 0.1,
  "mem": 32,
  "cmd": "while [ true ]; do echo ‘Hello Marathon’; sleep 10; done”,
  "instances": 2
}
```

Lab 5a

FIRST APP

LAB 5A - CREATE FIRST MARATHON APP

1. Use the DCOS CLI or Marathon UI to deploy the following application:

```
{
   "id": "/hello-world",
   "cmd": “while [ true ]; do echo 'Hello Marathon'; sleep 5; done",
   "cpus": 0.1,
   "mem": 10.0,
   "instances": 2
 }
```

2. Observe Marathon launching the application and view the STDOUT of the mesos task

Extra Credit

Add this application using the Marathon API

Marathon

CONTAINERS

CONTAINERIZERS

- Marathon supports Mesos containers and Docker containers
- If no format is specified, Mesos containers are used by default
- Docker and Mesos containers both use the same underlying containerization technology: Linux kernel cgroups
- Universal containerizer: Mesos containerizer with support for multiple container formats (such as docker, appc) and no dependencies on external runtimes (i.e. Docker daemon)

EXAMPLE APPLICATION JSON (DOCKER)

```
{
  "id": "inky",
   "container": {
       "docker": {
           "image": "mesosphere/inky",
            “network”: “HOST”,
       },
       "type": "DOCKER"
   },
   "args": ["hello"],
   "cpus": 0.2,
   "mem": 32.0,
   "instances": 1
 }
```

Lab 5b

DOCKER APP

LAB 5B - CREATE DOCKER APP

1. Use the DCOS CLI or Marathon UI to deploy the following application:

```
{
  "id": "python",
  "cmd": "python -m http.server $PORT0",
  "cpus": 0.1,
  "mem": 32,
  "instances": 1,
  "container": {
    "type": "DOCKER",
    "volumes": [],
    "docker": {
      "image": "python:3"
    }
  }
}
```

2. Observe Marathon launching the docker container and view the STDOUT of the mesos task

PRIVATE DOCKER REPOSITORY

- Private Docker repository:
  - To run an image from a private repository, include the URI pointing to a `dockercfg.tar.gz` that contains the login information
  - The `.dockercfg` file will be pulled into the sandbox as specified by the `$HOME` environment variable so the Docker CLI will automatically pick up the config

COMMANDINFO

- An entrypoint and/or a default command are currently supported for Docker Images
- To run a Docker image with a default command (i.e., `docker run image`), the `CommandInfo`'s value must not be set. If set, it will override the default command
- To run a Docker image with an entrypoint defined, the `CommandInfo`'s shell option must be set to `false`. If set to `true`, the Docker containerizer will run the user's command wrapped with `/bin/sh -c`, which will also become the parameters to the image content

Marathon

CONSTRAINTS

CONSTRAINTS

Constraints allow the specification of app placement policies to optimize for fault tolerance or locality

`[fieldname:OPERATOR:parameter]`

Use cases:

- Pinning apps to hosts
- Distributing app instances across racks or DCs
- Ensuring multiple app instances don't run on the same hosts
- Ensuring apps only run on machines with specific features (such as SSDs)

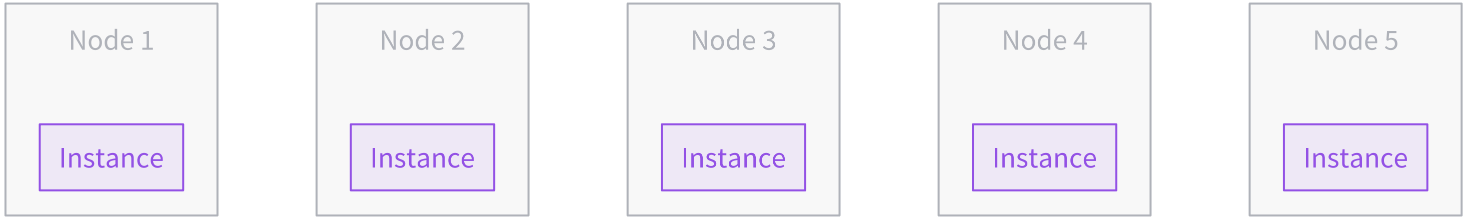
Constraints are distinct from *affinity*, a feature not yet implemented in Marathon

CONSTRAINTS

Operator	Definition
UNIQUE	All instances of the app must have a unique value for this attribute. [example: Run one instance per host] `“constraints”: [[“hostname”, “UNIQUE”]]`
GROUP_BY	All app instances will be ‘grouped’ evenly around this attribute, according to the number of groupings specified. This can be thought of as similar to the SQL ‘GROUP BY’ clause, with a limit. [example: Spread evenly by rack] `“constraints”: [[“rack”, “GROUP_BY”, 3]]`
CLUSTER	All app instances will be run on agents that share this attribute. [example: Only run on x86] `“constraints”: [[“cpu_arch”, “CLUSTER”, “x86”]]`
LIKE	A regular expression constraint that is applied to the attribute’s value. App instances will only run on agents that match the pattern. [example: Only run on 2x or 4x L] `“constraints”: [[“size”, “LIKE”, “m4.[24]xlarge”]]`
UNLIKE	Opposite of LIKE operator, avoids running workloads on certain agent nodes. [example: Don’t run in prod env] `“constraints”: [[“prod”, “UNLIKE”, “true”]]`

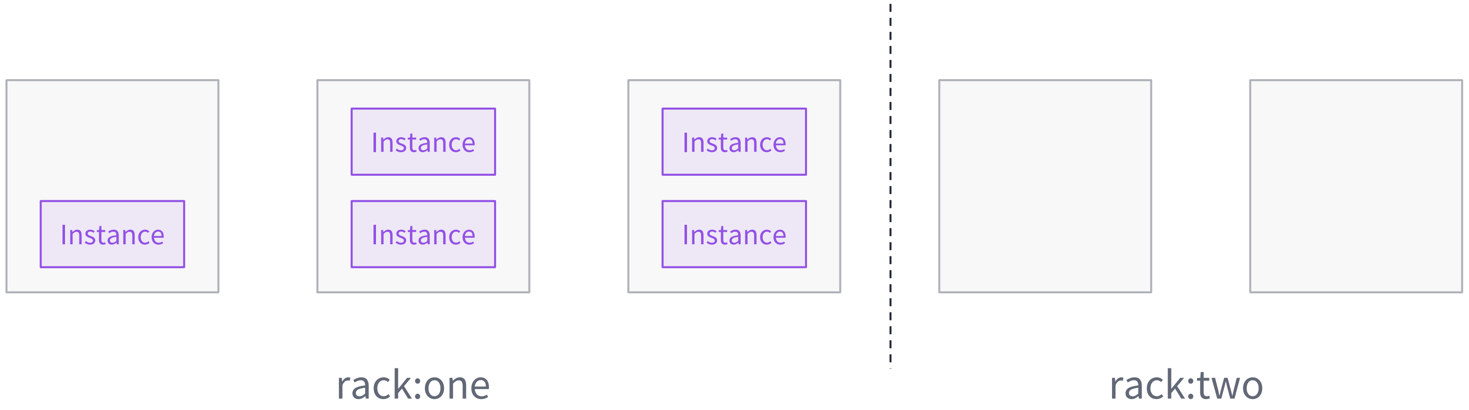
CONSTRAINT: UNIQUE

```
“constraints”: [[“hostname”, “UNIQUE”]]
“instances”: 5
```



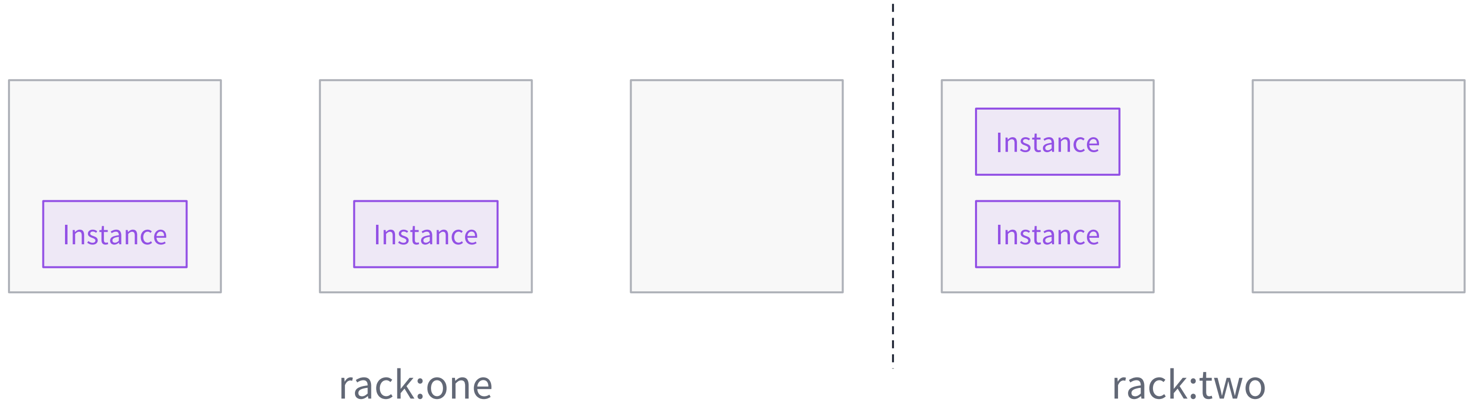
CONSTRAINT: CLUSTER

```
“constraints”: [[“rack”, “CLUSTER”, “one”]]
“instances”: 5
```



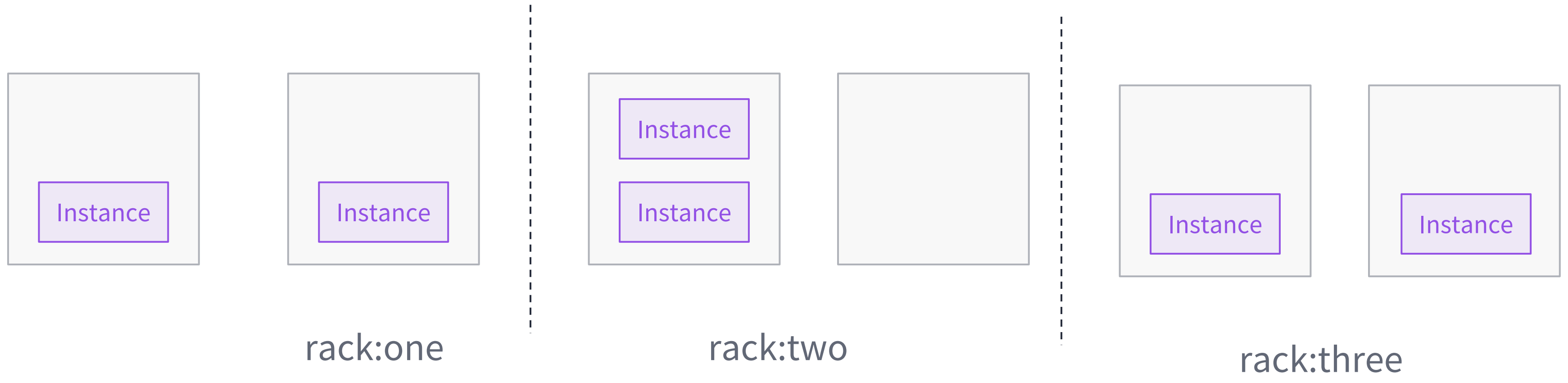
CONSTRAINT: GROUP BY

```
“constraints”: [["rack", "GROUP_BY"]]
“instances”: 4
```



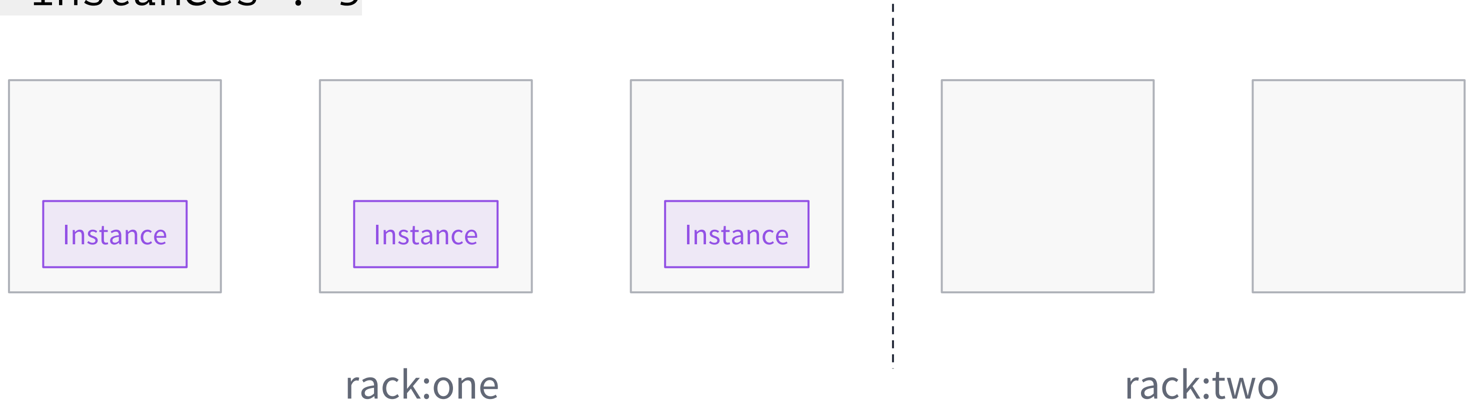
CONSTRAINT: GROUP BY

```
“constraints”: [["rack", "GROUP_BY", "3"]]
“instances”: 6
```



COMBINING CONSTRAINTS

```
“constraints”: [[“hostname”, “UNIQUE”], [“rack”, “CLUSTER”,
“one”]]
 “instances”: 3
```



Lab 5c

CONSTRAINTS

LAB 5C - CONSTRAINTS

1. Use the DCOS CLI or Marathon UI to deploy the following application:

```
#see 1-nginx.json
   …

 "constraints": [["hostname", "UNIQUE"]],
 "acceptedResourceRoles":["slave_public"]
```

2. Obtain marathon generated port for the nginx task
3. In a browser go to `http://<public-agent-ip>:<marathon port>`

Extra Credit

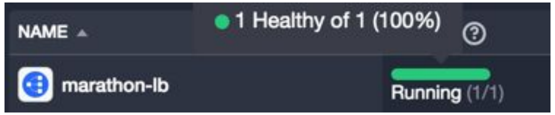
Can you scale the application out? If not, fix the issue.

Marathon

HEALTH CHECKS

HEALTH CHECKS

- Health check types
  - HTTP / HTTPS
  - TCP
  - CMD (executed by mesos agent)
- Interpretation of health checks is up to the framework—in the case of Marathon, an app instance which fails health checks will be killed and restarted
- Data from health checks may also be consumed by external services in order to collect availability metrics



HEALTH CHECK PARAMETERS

Parameter	Description
protocol	(Optional. Default: "HTTP"): Protocol of the requests to be performed. One of "HTTP", "HTTPS", "TCP", or "Command"
path	(Optional. Default: "/"): Path to endpoint exposed by the task that will provide health status.
portIndex	(Optional. Default: 0): Index in this app's ports array to be used for health requests
timeoutSeconds	(Optional. Default: 20): Number of seconds after which a health check is considered a failure regardless of the response
gracePeriodSeconds	(Optional. Default: 15): Health check failures are ignored within this number of seconds of the task being started or until the task becomes healthy for the first time
intervalSeconds	(Optional. Default: 10): Number of seconds to wait between health checks
maxConsecutiveFailures	(Optional. Default: 3) : Number of consecutive health check failures after which the unhealthy task should be killed

Lab 5d

HEALTH CHECKS

LAB 5D - HEALTHCHECKS

1. Use the DCOS CLI or Marathon UI to deploy the following application:

```
 #see 3-toggle-1.json
    …

"healthChecks": [
    {
      "protocol": "HTTP",
      "path": "/",
      "portIndex": 0,
      "timeoutSeconds": 10,
      "gracePeriodSeconds": 10,
      "intervalSeconds": 2,
      "maxConsecutiveFailures": 5
    }
```

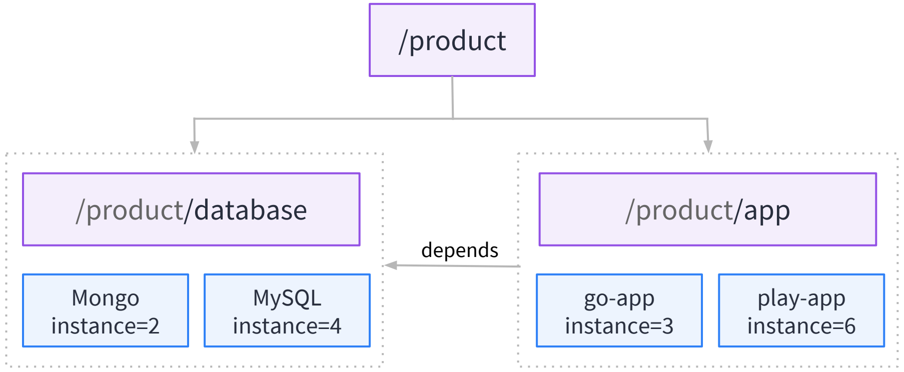
2. Observe Marathon launching the docker container and view the STDOUT of the mesos task

Marathon

GROUPS

GROUPS

- Service groups provide an abstraction for groups of applications
- Multiple apps can be partitioned into sets of apps for ease of management
- Some applications have dependencies and need to deployed with ordering
- Interdependent apps can often be modeled as DAGs



GROUP JSON EXAMPLE

```
{
  "id": "/product",
  "groups": [
    {
      "id": "/product/database",
      "apps": [
         { "id": "/product/mongo", ... },
         { "id": "/product/mysql", ... }
       ]
    },{
      "id": "/product/app",
      "dependencies": ["/product/database"],
      "apps": [
         { "id": "/product/go-app", ... },
         { "id": "/product/play-app", ... }
      ]
    }
  ]
}
```

GROUP USE CASES

- Allow for multiple layers of groups and services
- Organize groups of services (i.e. for tenancy)
- Provide for intelligent scaling of all applications within a group
- Scaling factor is a multiplier of current deployed instances
  - i.e., to double instances within a group
    - dcos marathon group scale <group-name> 2
  - To reduce the current instances within a group by half
    - dcos marathon group scale <group-name> .5

Lab 5e

GROUPS

LAB 5E - DEPLOY GROUP

1. Use the DCOS CLI or Marathon UI to deploy the following application:

```
#see 4-group.json
```

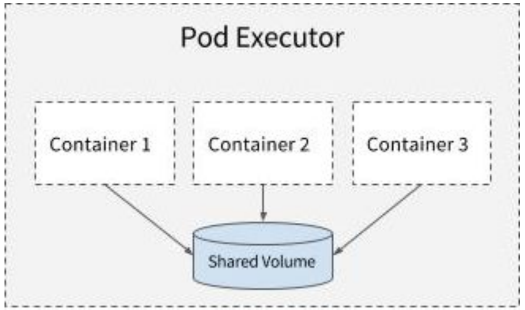
2. Observe Marathon launching the application group
3. Scale out by doing `dcos marathon group scale 2`
4. Scale down by doing `dcos marathon group scale .5`

Marathon

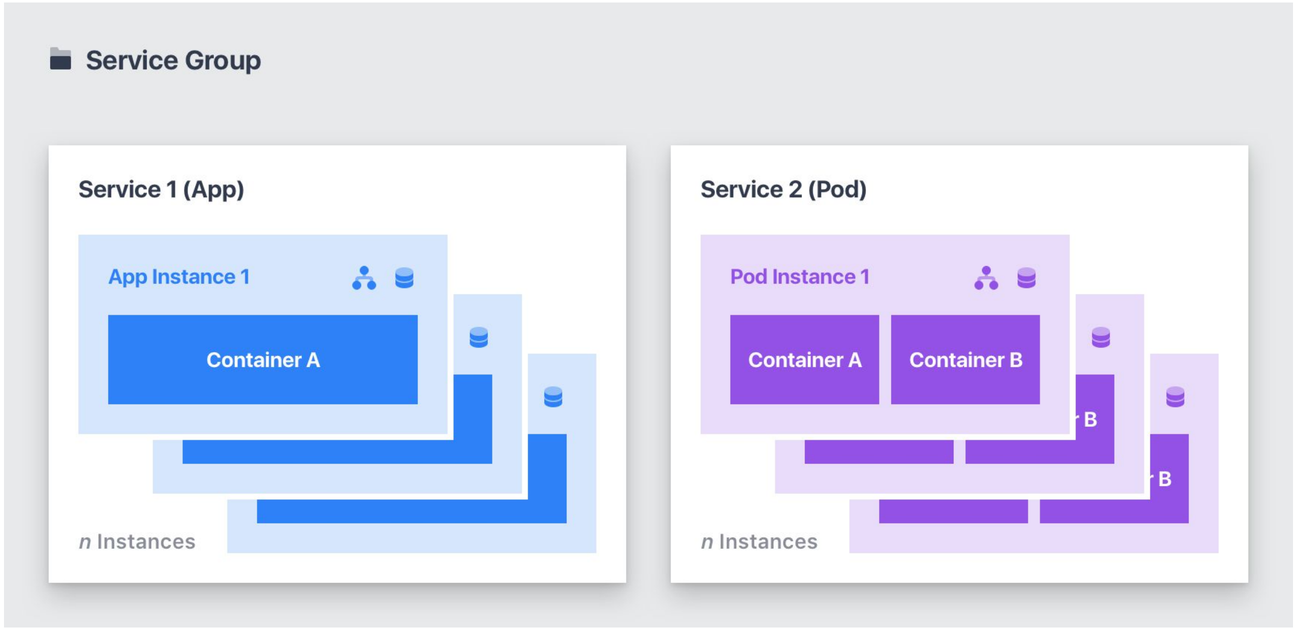
PODS

PODS

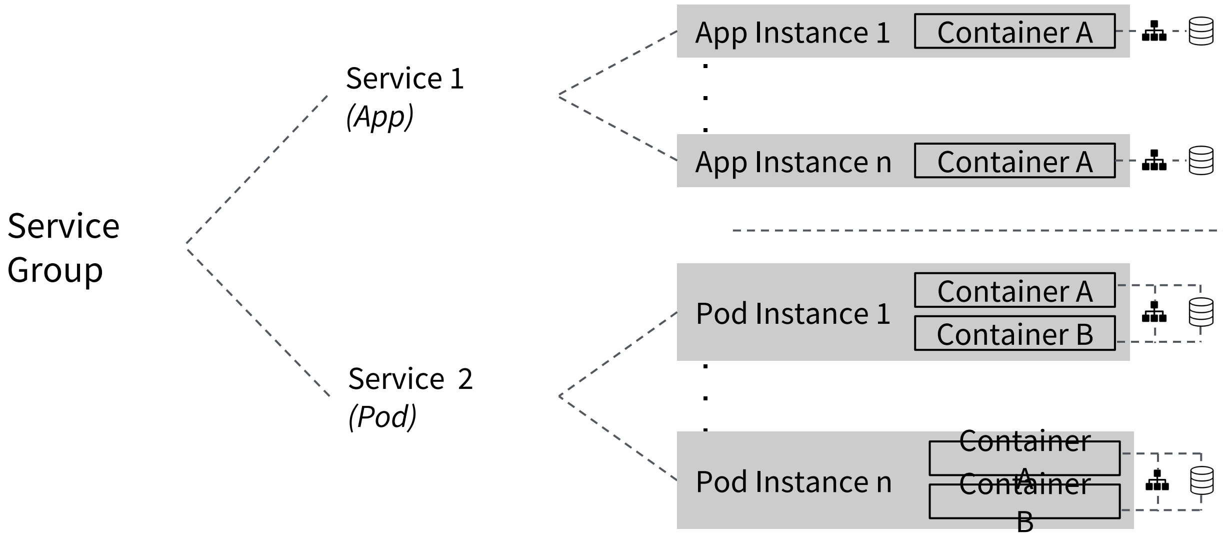
- A pod is a grouping of containers (task group)
- A pod instance's containers are launched together atomically on the same agent node and share networking namespace and ephemeral volumes
- Represented as a single service in Marathon
- Not supported in strict security mode
- Beta support for persistent volumes



APPS, GROUPS, PODS



APPS, GROUPS, PODS



Each application instance is only one container

Each Pod instance

- Contains one or more containers
- Co-locate together on the same host
- Deploys, scales and terminates as one unit
- Share network namespace and the same IP.
- Share storage space (sandbox and ephemeral local volumes)
- Conduct Pod-level health checks
- Restart entire pod when container crash or a health checks fail

MIGRATING LEGACY WORKLOADS

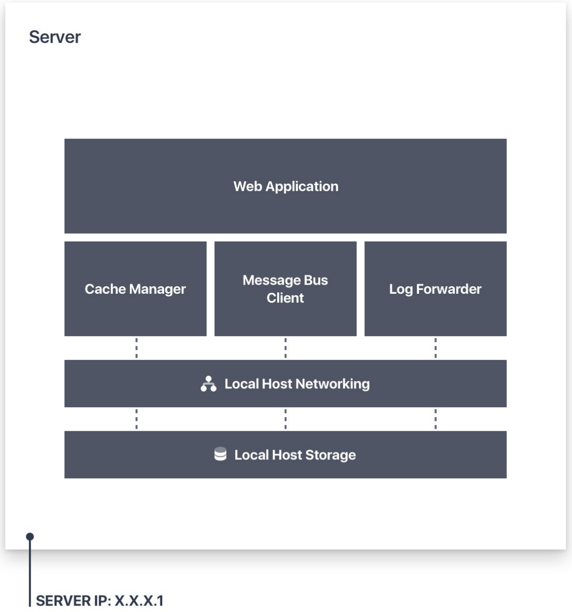
Overview

- Containers and microservices run only one process at a time, and one application has to be split into at least two containers.

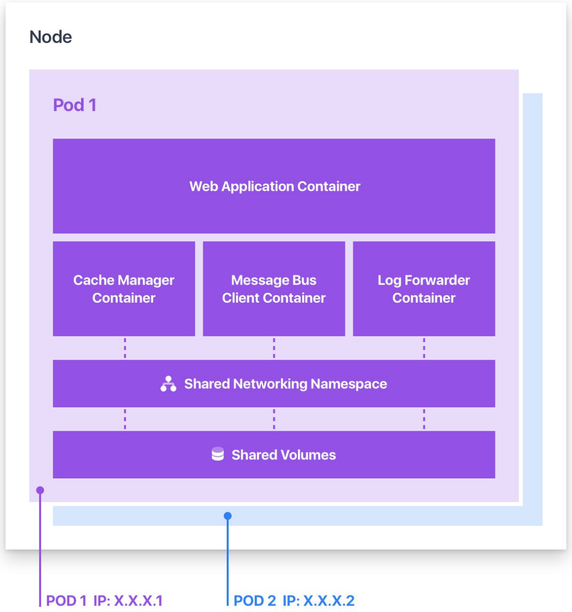
Legacy Workload Migration Challenges

- Pairing two or more containers together and co-locating them within the same host makes up for the differences between virtualized apps and containerized apps.

Legacy application on a traditional server



Legacy application migrated to Pods on DC/OS



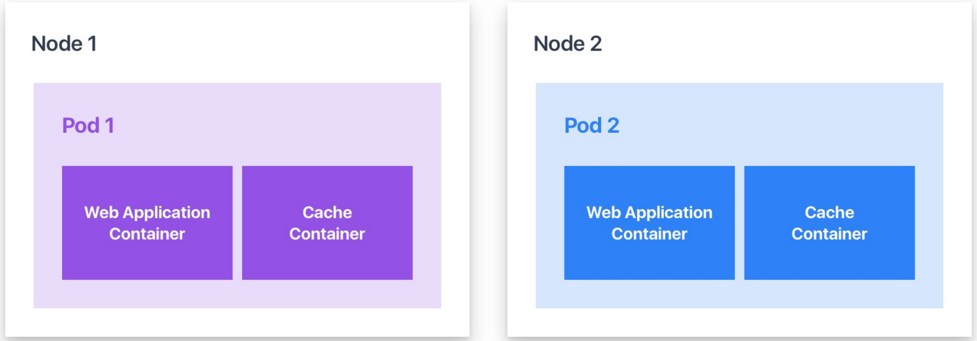
SIDE CAR CONTAINERS

Overview

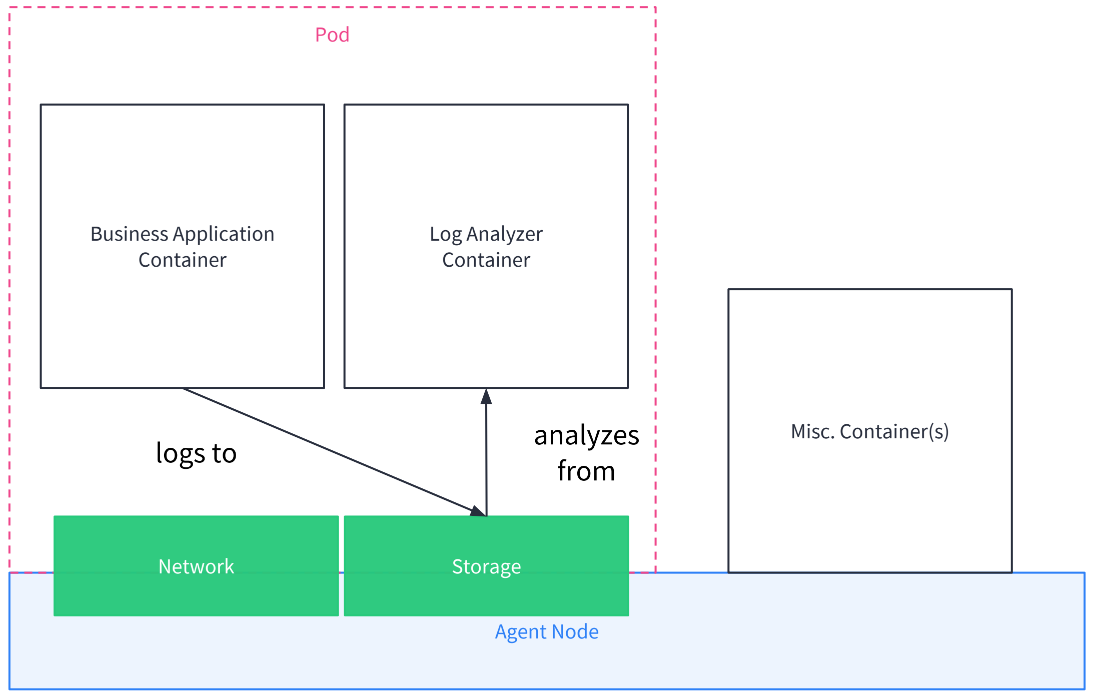
- Containers and microservices run only one process at a time, and one application has to be split into at least two containers.

Side-Car Container Challenges

- This pattern makes deployment challenging. For example, the cache container should be kept close to the web container.
- Having co-located, co-managed helper containers next to the application container simplifies things like:
    - Logging or monitoring agents
    - Backup tooling & Proxies
    - Data change watchers & Event publishers

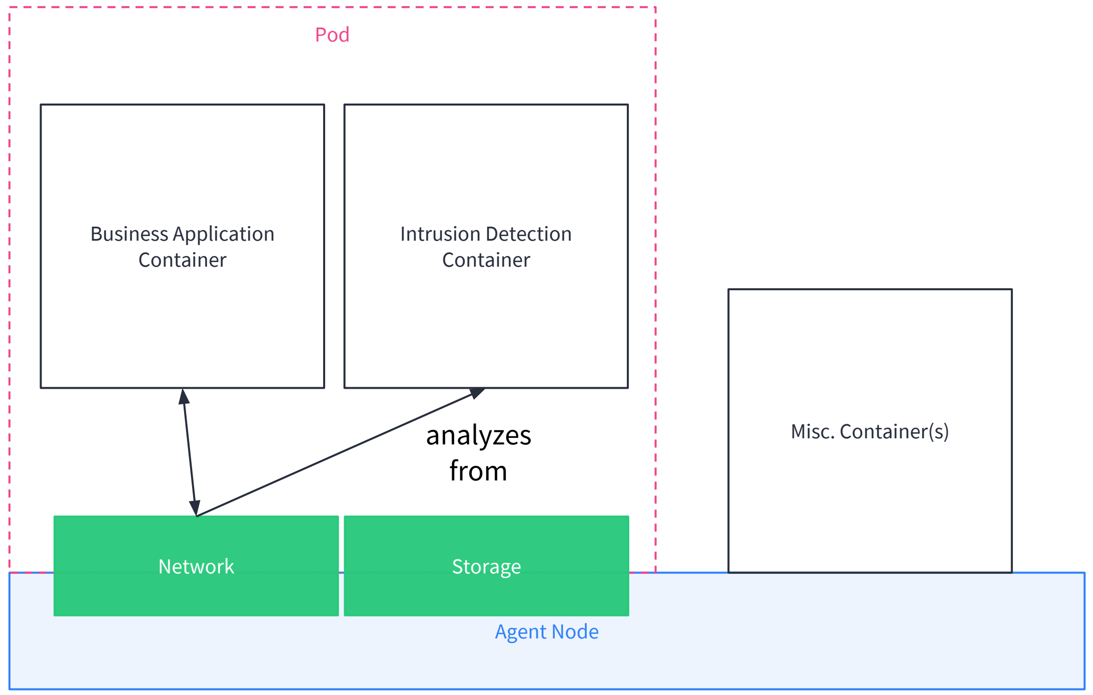


EXAMPLE OF A POD



Pods

EXAMPLE OF A POD



Lab 5f

PODS

LAB 5F - DEPLOY POD

1. Use the DCOS CLI or Marathon UI to deploy the following application:

```
#see 11-pods.json
```

2. Observe Marathon launching the pod
3. Inspect the pod application instance sandboxes
4. Scale out by changing the number of instances to 2

Marathon

ROLLING UPGRADES

DEPLOYMENT POLICIES

- Marathon begins a deployment if the application definition changes, i.e.
  - starting/stopping apps
  - updating app definition
  - scaling an app
- If the app definition is updated (which may be used to perform upgrades), Marathon can execute a rolling upgrade in conjunction with health checks
- Apps belonging to a group with dependencies will be deployed in the correct order, as specified by their dependencies

DEPLOYMENT POLICY CONFIGURATION

minimumHealthCapacity: A real value between 0.0 and 1.0 which specifies the percentage of app instances to maintain as healthy while performing a deployment.

maximumOverCapacity: A real value between 0.0 and 1.0 which specifies the maximum percentage of instances which can be over capacity during deployment.

Lab 5g

ROLLING UPGRADE

LAB 5F - ROLLING UPGRADE

1. Use the DCOS CLI or Marathon UI to deploy the following application:

```
#see 5-python-rolling-upgrade.json
  …
    "upgradeStrategy": {
    "minimumHealthCapacity": 0.85,
    "maximumOverCapacity": 0.15
 }
```

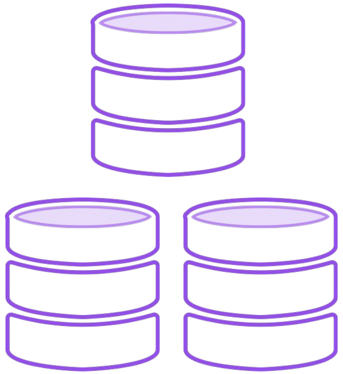
2. Wait until all application tasks are running
3. Update the application by change the docker container
4. From the marathon UI, view the rolling upgrade of the live application

Marathon

STATEFUL SERVICES

PERSISTING STATE

Containerized Services are dynamic and easily move around the cluster!

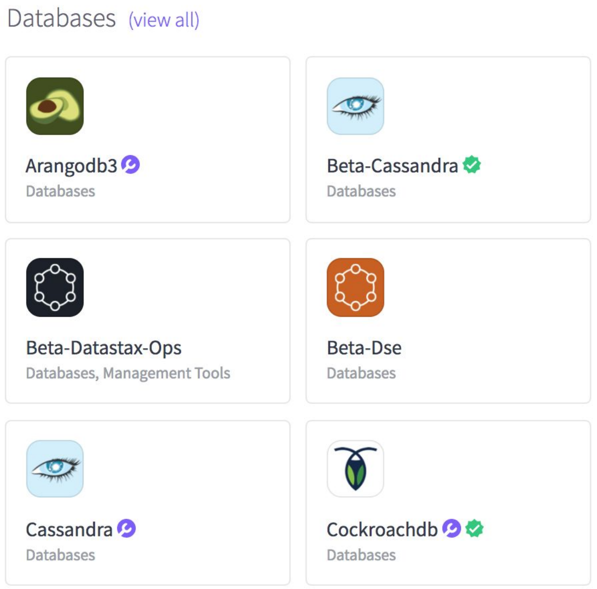


But, what about its state?

STATEFUL SERVICES

Stateful services of all kinds need persistence

- Traditional databases:
  - MySQL, MariaDB, Postgres
- Distributed databases:
  - Cassandra, Cockroach, Mongo
- Stateful message brokers:
  - Kafka, RabbitMQ
- Business applications with storage requirements: file processing, caching



Databases (view all)

Arangodb3 Databases	Beta-Cassandra Databases
Beta-Datastax-Ops Databases, Management Tools	Beta-Dse Databases
Cassandra Databases	Cockroachdb Databases

MESOS STORAGE OPTIONS

Default sandbox

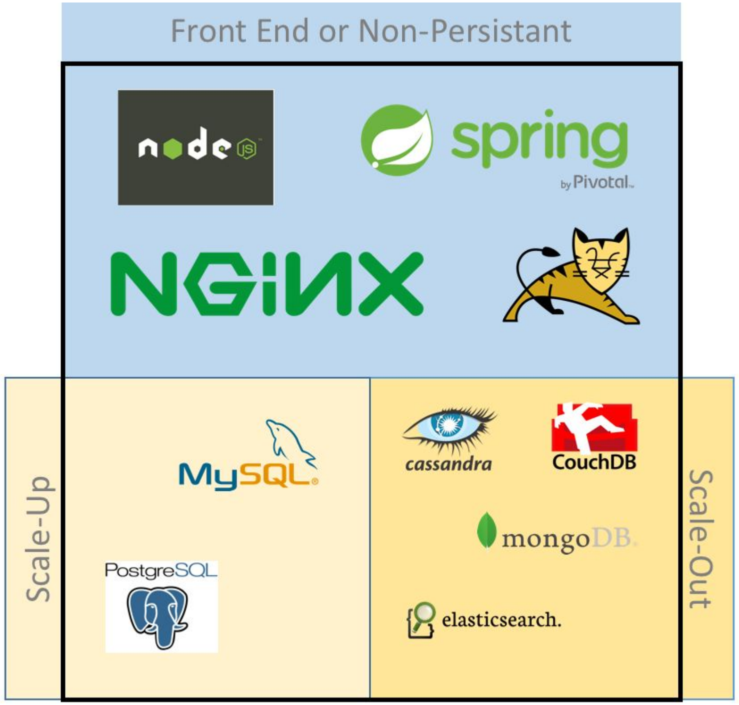
- Simple to use, task failures

Persistent volumes

- Simple to use, (permanent) node failures

DFS / External volumes

- Node failures



HOW DID WE DO THIS BEFORE?

Solution: Pin instances to certain nodes

Downsides:

- Manual process
- Prevents optimal resource utilization
- Cumbersome to scale

PERSISTENT VOLUMES

Features

Simple solution for local storage.

No manual configuration needed.

Benefits

Easily run databases on DC/OS

State survives application and machine restart.

Patterns

Apps with a persistent volume become **resident**; they are pinned to a machine.

For critical data, you must run 2+ instances and replicate the data.

PERSISTENT VOLUMES

Marathon will

- reserve required resources
- create persistent volumes
- launch based on offers
- “pin” tasks to agents

```
{
    "id": "/stateful",
    "container": {
        "type": "Mesos",
        "volumes": [{
            "containerPath": "mydata",
            "mode": "RW",
            "persistent": {"size": 100}
        }]
    },
    "residency": {
        "taskLostBehavior": "WAIT_FOREVER"
    }
}
```

MULTIPLE DISKS

Mesos can expose multiple types of disk resources

- root disk (default)
- path disk (auxiliary disk resource which can be subdivided into smaller chunks for persistent volumes)
- mount disk (auxiliary disk resource which cannot be subdivided)

```
{
      "resources" : [
        {
          "name" : "disk",
          "type" : "SCALAR",
          "scalar" : { "value" : 2048 },
          "disk" : {
            "source" : {
              "type" : "PATH",
              "path" : { "root" :
"/mnt/data" }
            }}]}
```

Lab 5h

PERSISTENT VOLUMES

LAB 5G - PERSISTENT VOLUMES

1. Use the DCOS CLI or Marathon UI to deploy the following application:

```
#see 7-mysql-pv.json
```

2. Wait until all application tasks are running
3. SSH to the agent hosting the application and login to the database

```
 sudo docker exec -it <container id> bash
mysql -u<username> -p -h 3.3.0.6 -P 3306
```

4. Create some tables and rows in the database

```
 use my_database
create table dcos (project VARCHAR(20), owner VARCHAR(20));
INSERT INTO dcos VALUES ('any','time');
show tables;
select * FROM dcos;
```

5. Kill the application task (Marathon will restart the task)
6. Re-login to the database and validate that the previous written data persists

Marathon

EXTERNAL VOLUMES

EXTERNAL VOLUMES

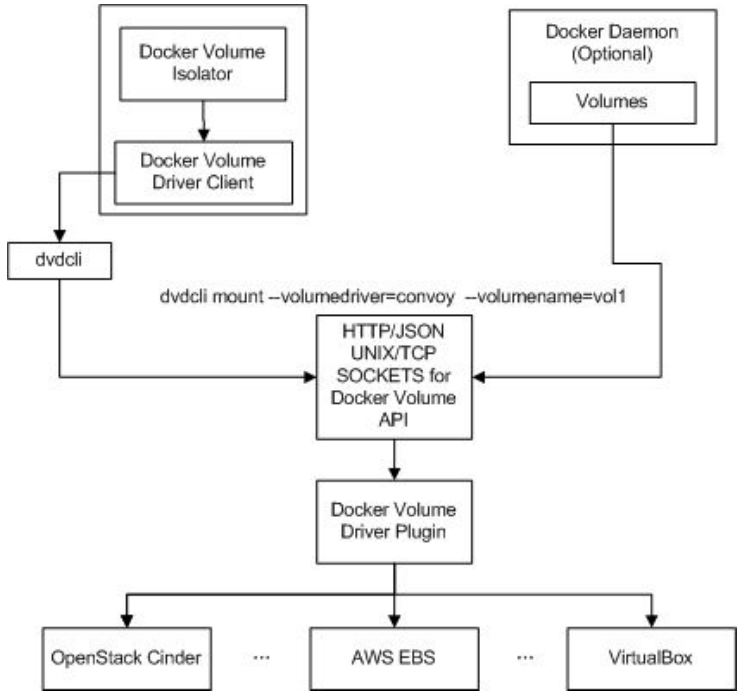
Leverages the `docker/volume` isolator

Uses the joint Mesosphere/EMC project called rexray as the Docker Volume plugin: https://github.com/emccode/rexray

- Vendor agnostic storage orchestration engine
- Abstracts control plane for multiple backends (server)
- Abstracts local host processes (discovery, attach, format, mount)
- Centrally configure shared storage system binding & credentials

Supports the following: AWS EBS, GCE, OpenStack Cinder, EMC XtremeIO, ScaleIO, VMAX, Isilon

DESIGN


Docker Volume Isolator
Docker Volume Driver Client
Docker Daemon (Optional)
Volumes
dvdcli
dvdcli mount --volumedriver=convoy --volumename=vol1
HTTP/JSON UNIX/TCP SOCKETS for Docker Volume API
Docker Volume Driver Plugin
OpenStack Cinder
...
AWS EBS
...
VirtualBox

EXTERNAL VOLUMES

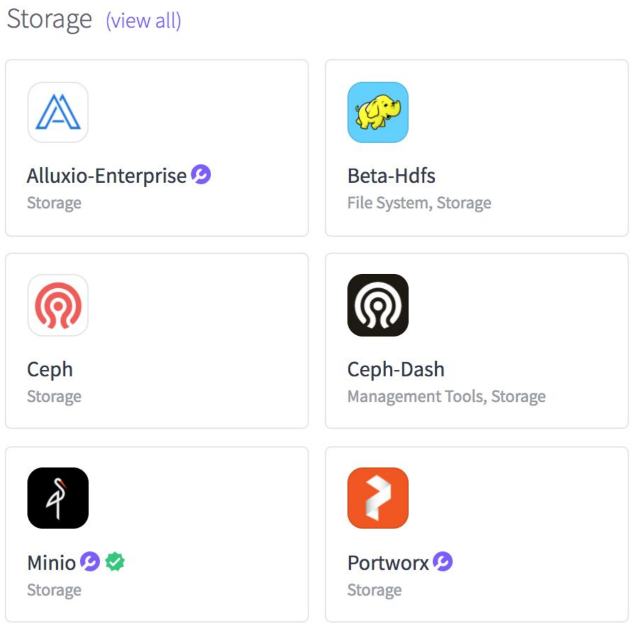
- Uses storage provider in a cluster
- Provides network volumes on any node
- More flexible scheduling with less node constraints

```
{   "id": "hello",
    "cmd": "/usr/bin/tail -f /dev/null",
    "container": {
        "type": "MESOS",
        "volumes": [{
            "containerPath": "test-rexray-volume",
            "external": {
                "size": 100,
                "name": "my-test-vol",
                "provider": "dvdi",
                "options": {"dvdi/driver": "rexray"}
            },
            "mode": "RW"
        }]
}}
```

DISTRIBUTED FILE SYSTEMS

Storage solutions designed for containerized applications

- Portworx
- Ceph
- Alluxio
- Bookkeeper
- Quobyte
- Minio
- Hedvig



Storage (view all)

Alluxio-Enterprise
Storage

Beta-Hdfs
File System, Storage

Ceph
Storage

Ceph-Dash
Management Tools, Storage

Minio
Storage

Portworx
Storage

CSI - THE FUTURE

Defines an industry standard **Container Storage Interface** (CSI) that will enable storage vendors to develop a plugin once and have it work across a number of container orchestration (CO) systems.

Actions include:

- Dynamic provisioning and deprovisioning of a volume
- Attaching or detaching a volume from a node
- Mounting/unmounting a volume from a node
- Consumption of both block and mountable volumes
- Local storage providers (e.g., device mapper, lvm).

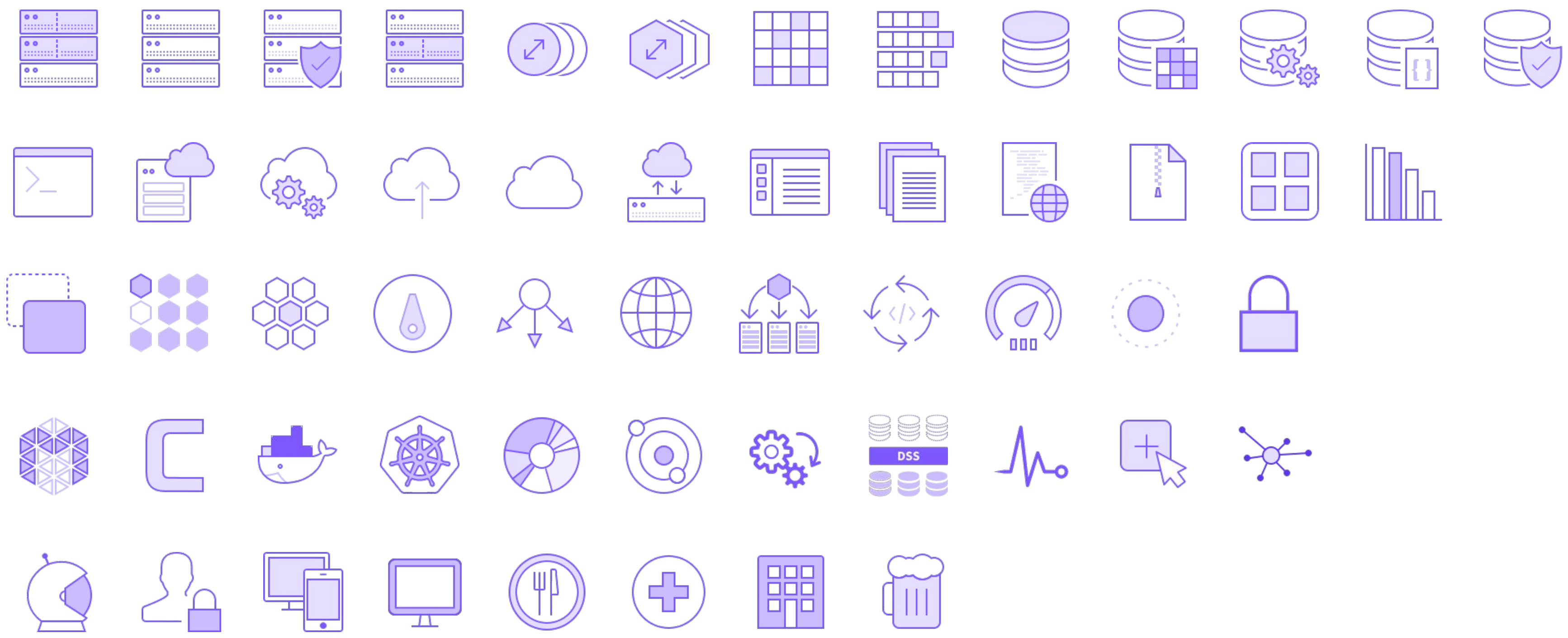
SUMMARY

In this module we looked at the container orchestration capabilities of Marathon

- Launching services
- Specifying constraints
- Configuring healthchecks
- Deploying service groups
- Deploying pods
- Application upgrades
- Data persistence

MESOSPHERE

Marketing Icons



Marathon

ADVANCED

EVENT BUS

- Marathon provides an event bus which contains a firehose of internal events
- Types include:
  - API requests
  - Health check data
  - Task status changes

EVENT BUS EXAMPLE

```
{
  "eventType": “health_status_changed_event”,
  "timestamp": “2014-03-01T23:29:30.158Z”,
  "appId": “/my-app”,
  "taskId": "my-app_0-1396592784349",
  "version": “2014-04-04T06:26:23:051Z”,
  “alive”: true
}
```

ADVANCED FEATURES

- Artifact store
  - Can be used to distribute artifacts for anything running within the cluster
  - May be backed by HDFS
- Task environment variables
  - Marathon will provide several environment variables to tasks, including: ports, app ID & version, container image, Mesos task ID, and the sandbox path
  - Env variables provided differ based on containerizer
- Plugin interface
  - Marathon can be extended via an experimental plugin API
  - https://mesosphere.github.io/marathon/docs/plugin.html

QUESTIONS

COMMAND LINE FLAGS

Flag	Description
--failover_timeout	(Optional. Default: 604800 seconds (1 week)): The failover_timeout for Mesos in seconds. If a new Marathon instance has not re-registered with Mesos this long after a failover, Mesos will shut down all running tasks started by Marathon.
--mesos_role	(Optional. Default: None): Mesos role for this framework. If set, Marathon receives resource offers for the specified role in addition to resources with the role designation '*'.
--reconciliation_initial_delay	(Optional. Default: 15000 (15 seconds)): The delay, in milliseconds, before Marathon begins to periodically perform task reconciliation operations.
--reconciliation_interval	(Optional. Default: 300000 (5 minutes)): The period, in milliseconds, between task reconciliation operations.
--event_subscriber	(Optional. Default: None): Event subscriber module to enable. Currently the only valid value is http_callback.
--[disable_]zk_compression	(Optional. Default: enabled): Enable compression of ZK nodes if the size of the node is bigger than the configured threshold.
--store_cache	(Optional. Default: true): Enable an in-memory cache for the storage layer.

BACKUP SLIDES

MARATHON ON MARATHON (MOM)

Marathon

OPERATIONS

PERSISTENT VOLUMES

- Persistent volumes are not deleted unless you
  - delete the service
  - delete a task adding `?wipe=true`
- MultiDisk not yet supported
- Can't use pre-reserved resources yet
- `hostname UNIQUE` constrained services should be scaled 1 at a time

STATE STORAGE

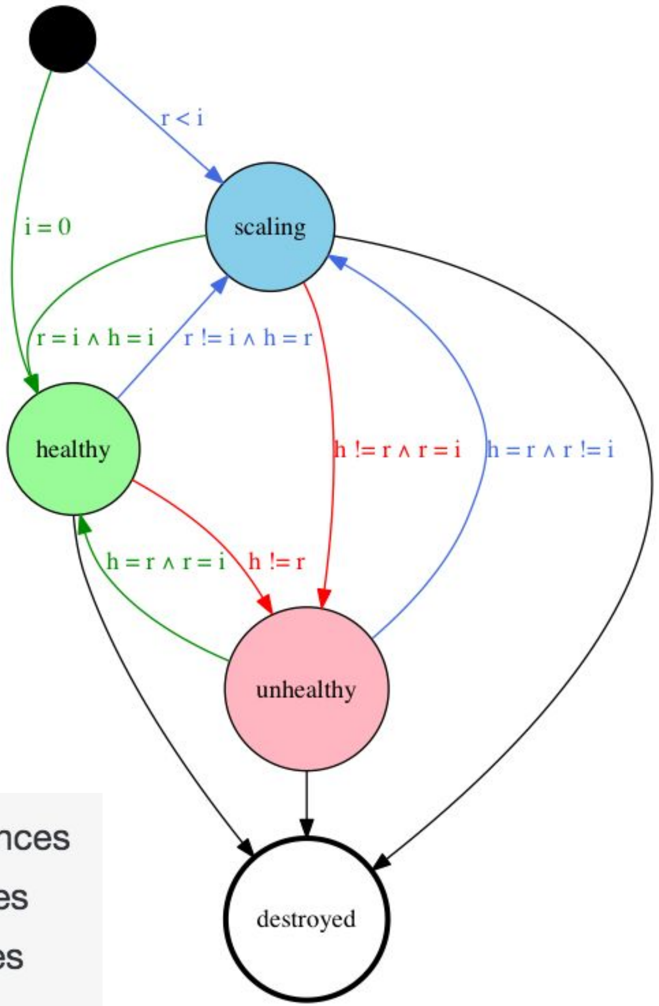
- Marathon stores its state in ZooKeeper
- ZK nodes are created per app, and there’s 1 znode for each:
    - Current app definition
    - Marathon task (app instance)
    - App definition version (up to `` `--zk_max_versions` ``)
- ZK nodes are limited to 1MiB by default: app definitions exceeding 1MiB in size will cause undefined behaviour

METRICS & MONITORING

- Marathon exposes metrics via `/metrics` endpoint
- Data can be collected with a tool such as collectd and forwarded to a time series DB (Graphite, InfluxDB, Elasticsearch, etc)
- Metrics can be forwarded directly to Graphite or Datadog, via CLI flags (`--reporter_graphite` and `--reporter_datadog`).

HEALTH CHECKS

Lifecycle of a health check



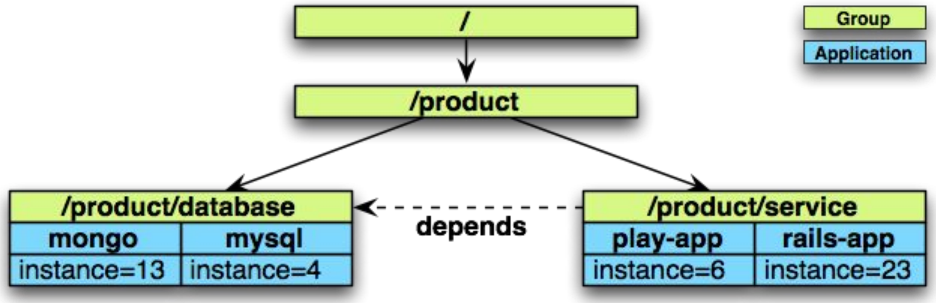
`i` is the number of requested instances

`r` is the number of running instances

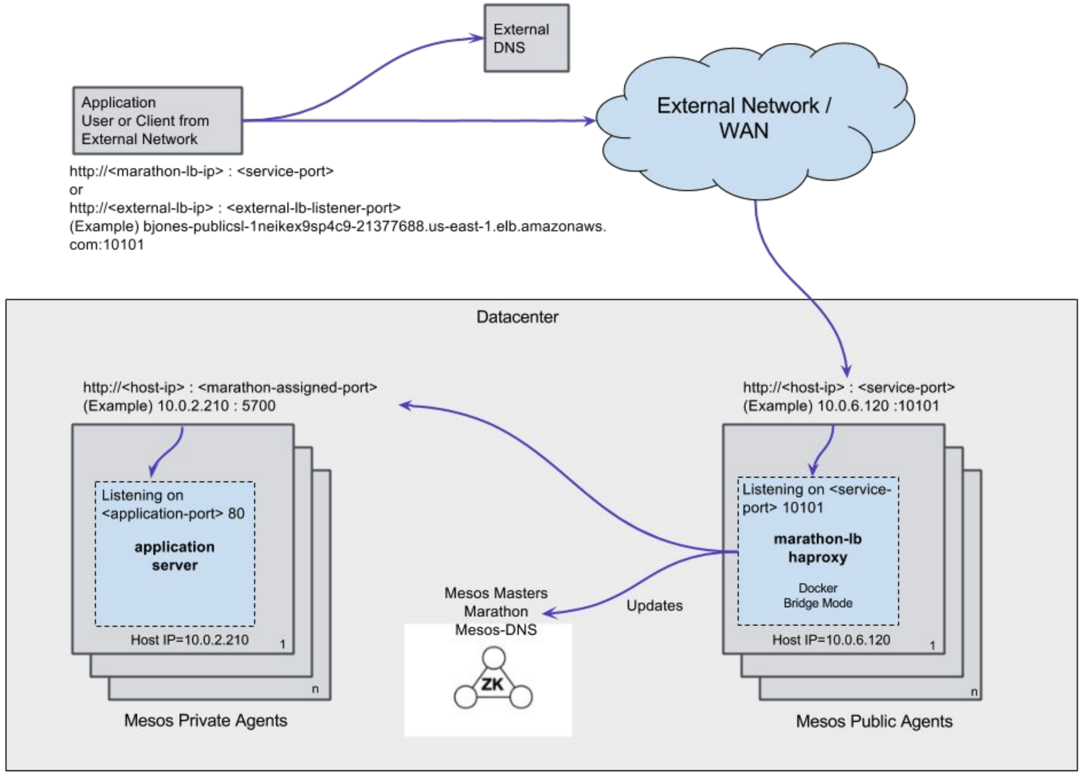
`h` is the number of healthy instances

SERVICE GROUPS

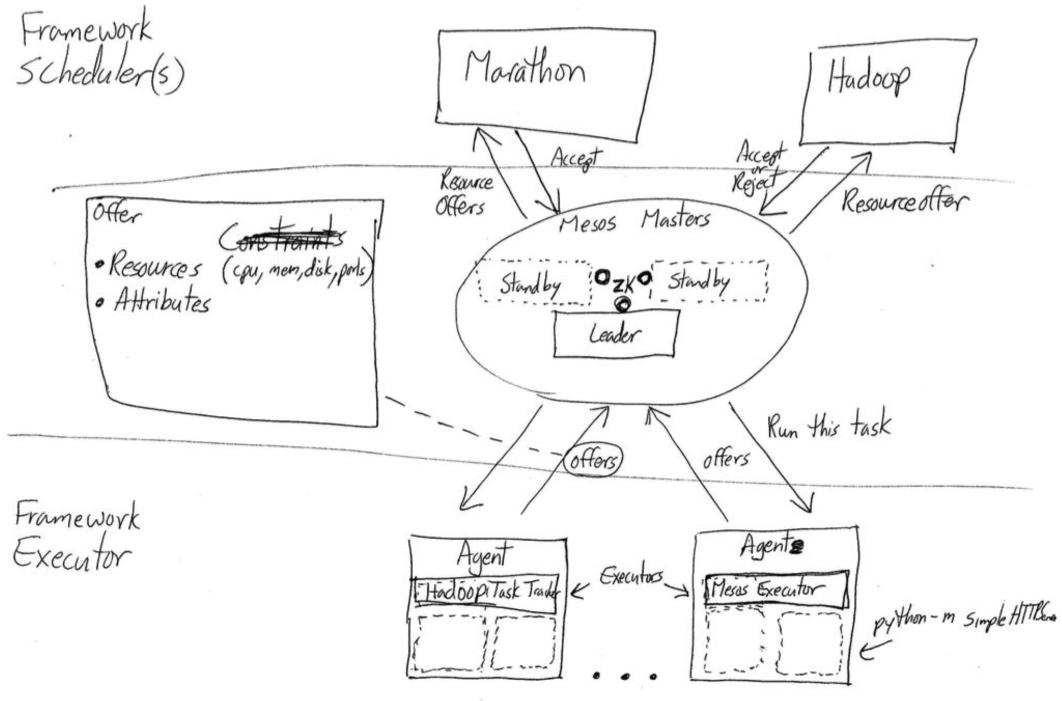
- Service groups provide an abstraction for groups of applications
- Multiple apps can be partitioned into sets of apps for ease of management
- Some applications have dependencies and need to deployed with ordering
- Interdependent apps can often be modeled as DAGs



MARATHON-LB ARCHITECTURE


External DNS
Application User or Client from External Network
External Network / WAN
http://<marathon-lb-ip> : <service-port>
or
http://<external-lb-ip> : <external-lb-listener-port>
(Example) bjones-publicsl-1neikex9sp4c9-21377688.us-east-1.elb.amazonaws.com:10101
Datacenter
http://<host-ip> : <marathon-assigned-port>
(Example) 10.0.2.210 : 5700
http://<host-ip> : <service-port>
(Example) 10.0.6.120 :10101
Listening on <application-port> 80
application server
Host IP=10.0.2.210
1
n
Mesos Private Agents
Listening on <service-port> 10101
marathon-lb haproxy
Docker Bridge Mode
Host IP=10.0.6.120
1
n
Mesos Public Agents
Updates
Mesos Masters
Marathon
Mesos-DNS
ZK

Whiteboard - Marathon & Mesos Resource Offers



DOCKER

Docker-specific features in Marathon and Mesos

- Dynamic port mapping for bridge networking, with service ports (for integration with Marathon-lb)
- Mesos sandbox is mounted inside the container at $MESOS_SANDBOX path