v1.11

DC/OS TECHNICAL

MESOSPHERE

AGENDA

1. Architecture
2. Installation
3. Linked clusters
4. Workload bursting
5. Multi-datacenter

DC/OS abstracts your datacenter into a single computer, pooling distributed workloads, simplifying rollout and operations.

DC/OS COMPONENTS

Catalog

HDFS	Spark	Kafka	Cassandra	Kubernetes

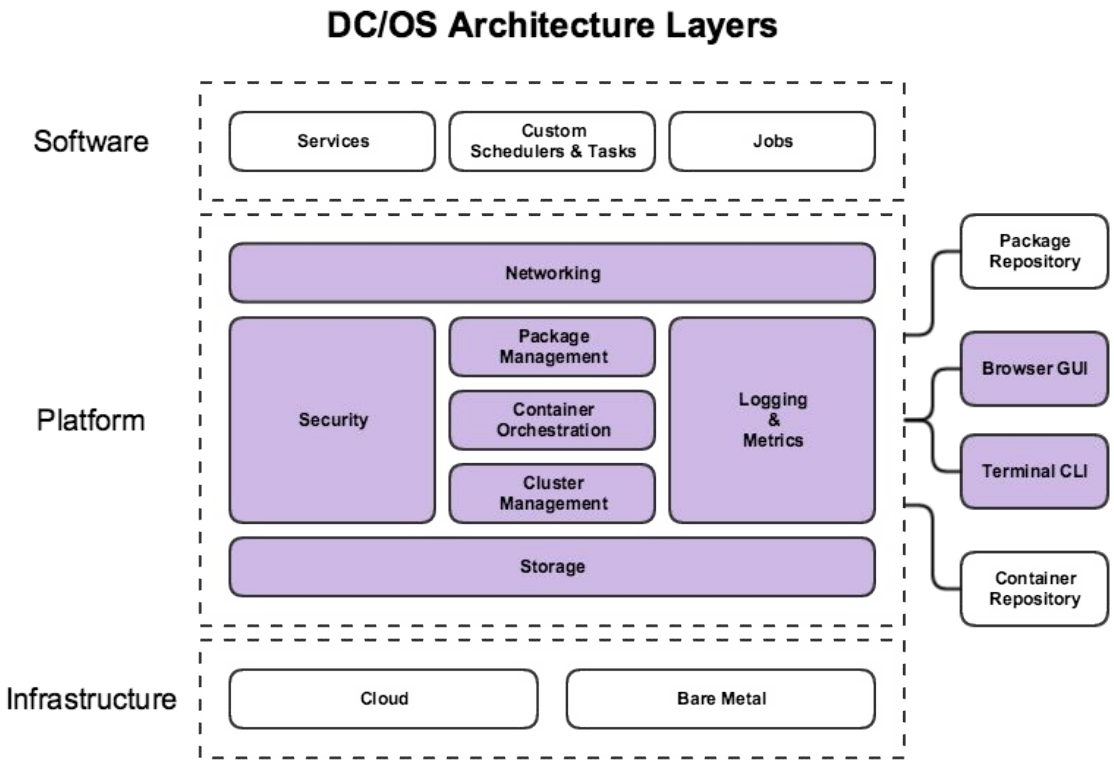
Mesosphere DC/OS

DC/OS UI	**DC/OS CLI**	**DC/OS IAM**	**Admin Router** Bouncer	**Cosmos**
Apache Mesos	Marathon	Mesos-DNS	ZooKeeper	**dcos-net**

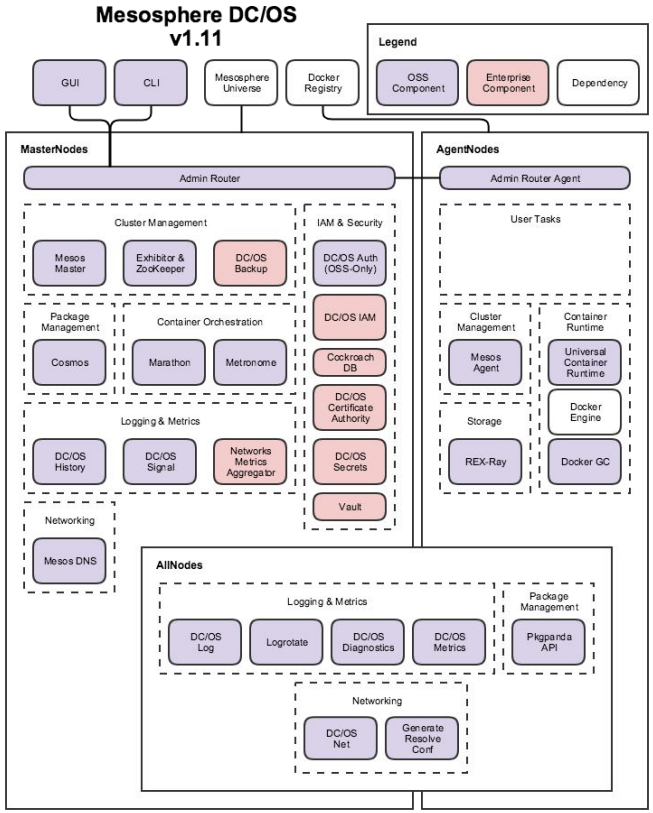
DC/OS Technical

ARCHITECTURE

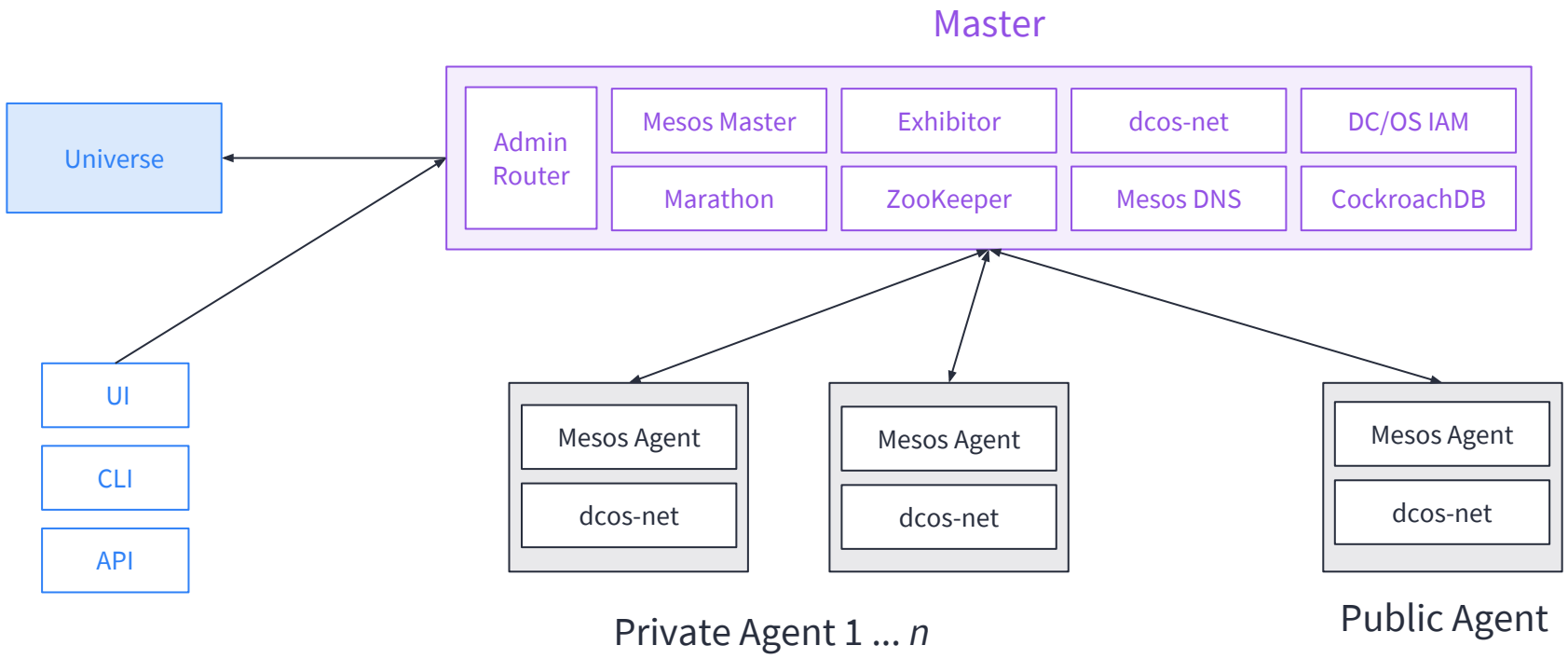
ARCHITECTURE



COMPONENTS



DC/OS ARCHITECTURE



CONCEPTS

DC/OS cluster

- Master node (quorum)
- Agent node(s)
  - Private
  - Public
- Bootstrap node

TAXONOMY

- DC/OS service (framework)
- User service (Marathon app)
- System service
- Job (scheduled task)
- Package
- Package registry

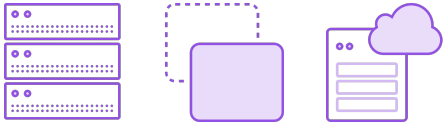
DC/OS Technical

INSTALLATION

ENVIRONMENTS

Public Cloud

- AWS
- Azure
- other



On-prem

- Bare-metal
- Virtual
- Private Cloud

Multiple

DEPLOYMENT TOOLS

Vagrant (local)

Docker (local)

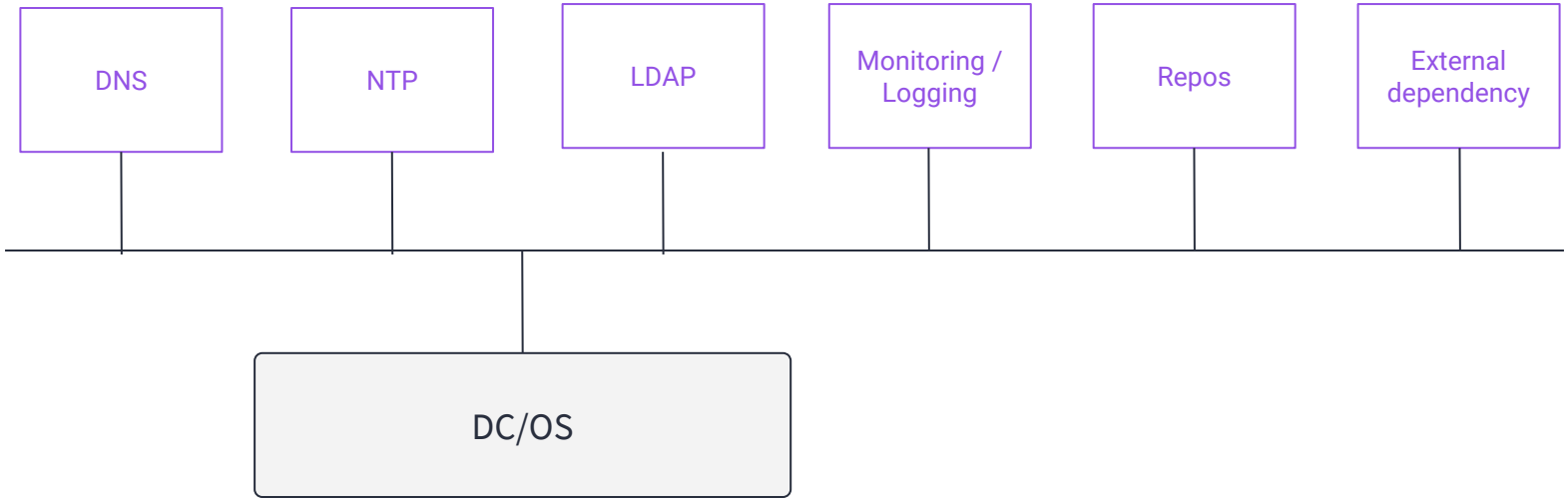
DC/OS Installer

- CLI
- Advanced

Cloud Templates

- Cloud Formations
  - Basic
  - Advanced
- Azure Resource Manager (ARM)

INFRASTRUCTURE SERVICES



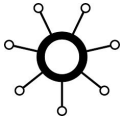
IMAGING

Base image

- OS version (EL7, CoreOS)
- Kernel version
- Container runtime version

Additional services

- Monitoring agents
- SDN agents



INSTALL PREREQUISITES

	Count	CPU	Mem (GB)	Disk (GB)
Bootstrap	1	2	16	60
Master	1,3,5	4	32	120
Agent	6+	2	16	60

INSTALL PREREQUISITES

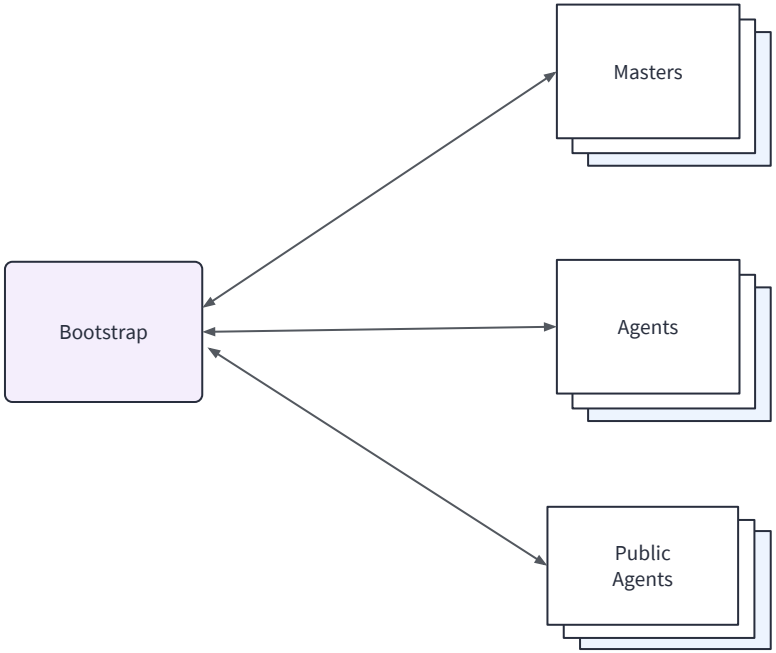
- Supported OS version (Centos/RHEL/CoreOS)
- Supported docker version
- Network access to Docker repository
- DNS resolution
- NTP configured
- IP connectivity to all nodes

https://docs.mesosphere.com/1.11/installing/ent/custom/system-requirements/

DC/OS INSTALLER

DC/OS Installer

- CLI
- Advanced



BOOTSTRAP NODE

The bootstrap node is used to generate the DC/OS installation bundle.

The following is required:

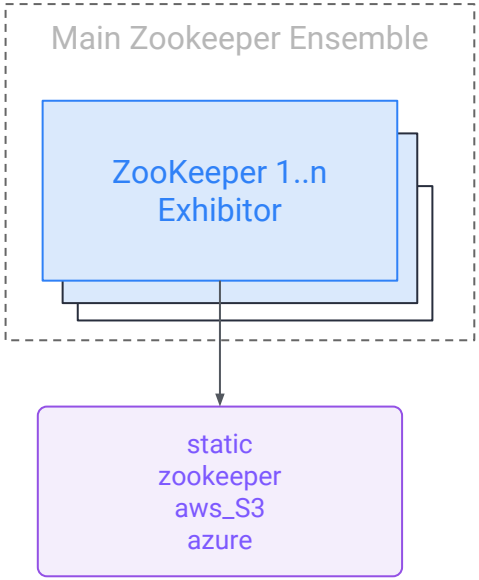
- Exhibitor storage (zooKeeper, S3, NFS, static)
- config.yaml
- ip-detect script
- license file
- nginx docker image to serve files

EXHIBITOR

Exhibitor is a supervisor for ZooKeeper and is the first DC/OS service installed. Exhibitor automatically configures your ZooKeeper installation on the master nodes during DC/OS installation.

There are two types of Exhibitor backends:

- Internal DC/OS storage (static)
- External (zookeeper, aws_s3, and azure)



INSTALL CONFIGURATION

```
/home/centos
├── dcos-genconf.<HASH>.tar
├── dcos_generate_config.sh
└── genconf
    ├── config.yaml
    ├── ip-detect
    ├── license.txt
    ├── serve
    ├── ssh_key
    └── state
```

```
$ cat /home/centos/genconf/config.yaml
---
agent_list:
- <agent-private-ip-1>
- <agent-private-ip-2>
exhibitor_storage_backend: static
public_agent_list:
- <agent-public-ip-1>
bootstrap_url: file:///opt/dcos_install_tmp
cluster_name: <cluster-name>
master_discovery: static
master_list:
- <master-private-ip-1>
- <master-private-ip-2>
- <master-private-ip-3>
resolvers:
- <internal-dns-1>
- <internal-dns-2>
superuser_password_hash: <hashed-password>
superuser_username: <username>
```

CLI INSTALL

Preflight Cluster
`--preflight`

Deploy DC/OS
`--deploy`

Postflight Cluster
`--postflight`

Uninstall DC/OS
`--uninstall`

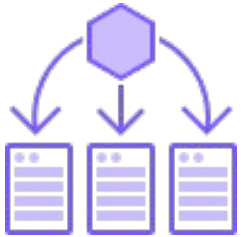
```
|Running mesosphere/dcos-genconf docker with BUILD_DIR set to /home/centos/genconf
|23:50:54:: ====> EXECUTING PREFLIGHT
|23:50:54:: ====> START run_preflight
|23:50:55:: ====> STAGE preflight
|23:50:55:: ====> STAGE preflight
|23:50:55:: ====> STAGE preflight
|23:50:55:: ====> STAGE preflight
|23:50:55:: ====> STAGE preflight
|23:50:55:: ====> STAGE preflight
|23:50:55:: ====> STAGE preflight
|23:50:55:: ====> STAGE preflight
|23:50:55:: ====> STAGE preflight_cleanup
|23:50:55:: ====> STAGE preflight
|23:50:55:: ====> STAGE preflight_cleanup
|23:50:55:: ====> STAGE preflight_cleanup
|23:50:55:: ====> STAGE preflight_cleanup
|23:50:55:: ====> STAGE preflight_cleanup
|23:50:55:: ====> STAGE preflight_cleanup
|23:50:55:: ====> STAGE preflight_cleanup
|23:50:55:: ====> STAGE preflight_cleanup
|23:50:55:: ====> STAGE preflight_cleanup
|23:50:55:: ====> OUTPUT FOR run_preflight
|23:50:55:: ====> 10.10.0.173:22 FAILED
|23:50:55::      TASK:
|/usr/bin/ssh -oConnectTimeout=10 -oStrictHostKeyChecking=no -oUserKnownHostsFile=/dev/null -oBatch
|.10.0.173 sudo bash /opt/dcos_install_tmp/dcos_install.sh --preflight-only master
|23:50:55::      STDERR:
|23:50:55::           Connection to 10.10.0.173 closed.
|23:50:55::      STDOUT:
|23:50:55::           Running preflight checks
|23:50:55::           Checking docker version requirement (>= 1.6): FAIL (0)
|23:50:55::           Checking Docker is configured with a production storage driver: Cannot connec
|23:50:55::
|23:50:55:: ====> END run_preflight with returncode: 1
|23:50:55:: ====> SUMMARY FOR run_preflight
|23:50:55:: 8 out of 9 hosts successfully completed run_preflight stage.
|23:50:55:: The following hosts had failures detected during run_preflight stage:
|23:50:55::      10.10.0.173:22 failures detected.
|23:50:55:: ====> END OF SUMMARY FOR run_preflight
|[root@ip-10-10-0-168 centos]#
```

ADVANCED INSTALLER

For advanced, automated pipelines

- Create DC/OS distribution package
- Connect to each node to manually run install
- Integrate with existing config systems

```
curl -O http://<bootstrap-ip>:<your_port>/dcos_install.sh
sudo bash dcos_install.sh <master|slave|slave_public>
```



INSTALLER BEST PRACTICES

Test dependencies

- external DNS resolvers
- ip-detect script

If genconf script errors out

- examine debug output

If dcos install script errors out

- examine journalctl log
- check bootstrap_url is valid

PROXY CONFIGURATION

If nodes are behind a proxy, configure the following:

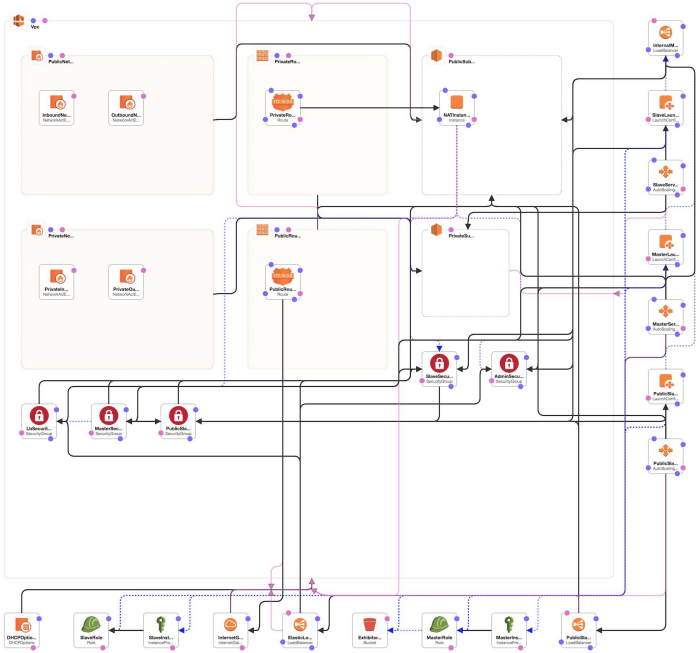
- Masters (`/var/lib/dcos/environment.proxy`)
- Agents (`/var/lib/dcos/mesos-slave-common`)
- Docker (`/etc/systemd/system/docker.service.d/http-proxy.conf`)

BASIC CF TEMPLATES

For demos and PoCs

Inputs:

- S3 template URL
- Stack name
- Key Pair
- OAuth Enabled
- PublicSlaveInstanceCount
- PrivateSlaveInstanceCount



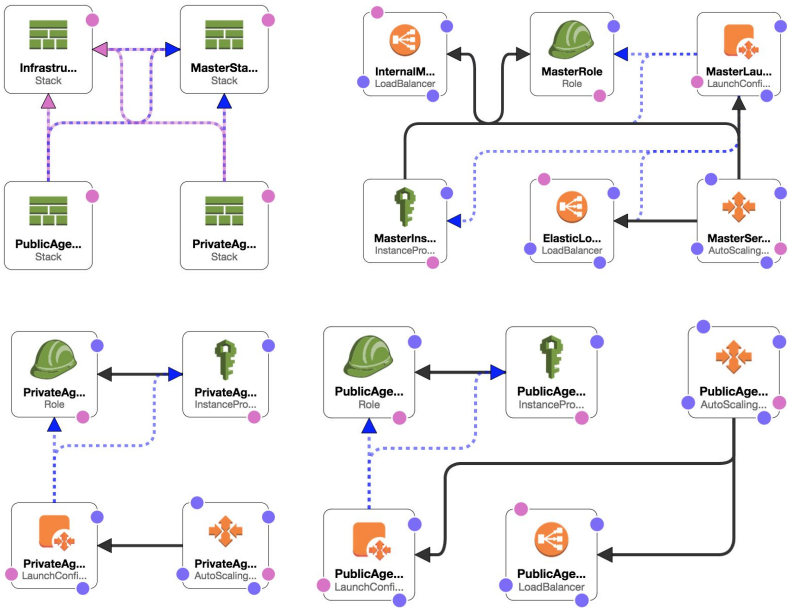
ADVANCED CF TEMPLATES

For advanced or production deployments

- Zen template
- Public agent stack
- Private agent stack
- Master stack
- Infrastructure stack

Can use existing VPC/Subnet

Add additional agent nodes to cluster



DC/OS Technical

LINKED CLUSTERS

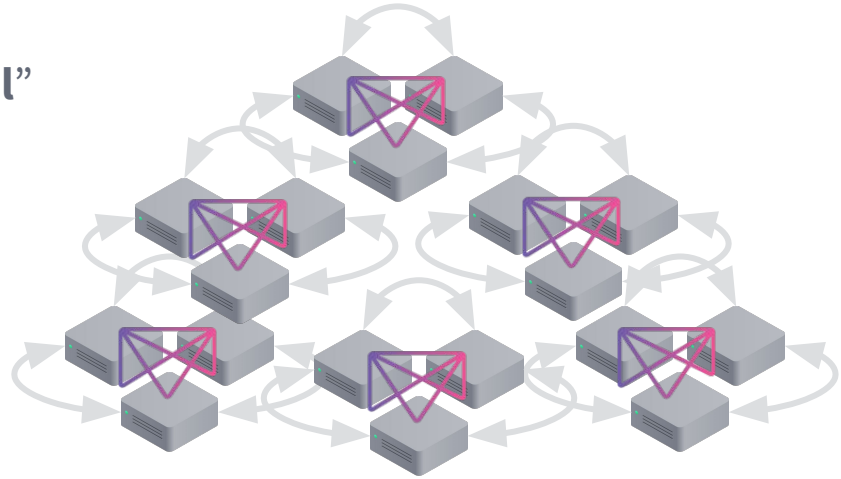
THE PROBLEM

Companies typically operate more than one DC/OS cluster for:

- Separation of environments (test / staging / prod)
- Accommodating geographical distribution

This can lead to a new problem "**cluster-sprawl**"

- Access management difficulty
- Increased complexity
- Potential compliance issues

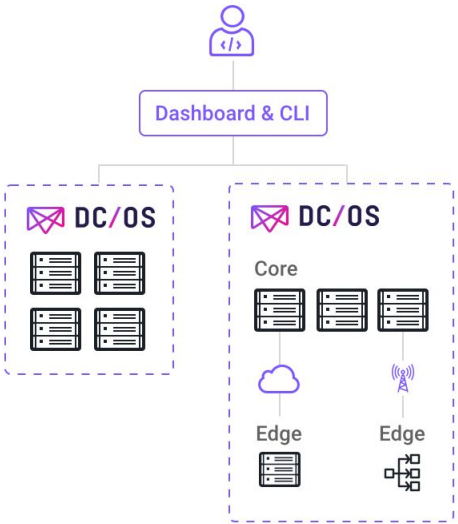


LINKED CLUSTERS

Single pane of glass (UI / CLI) across all DC/OS environments

DC/OS Cluster Linker New component written in “golang” and runs on all the master nodes (Active - Active)

- Provides HTTP API to link/unlink/list clusters
- CLI support Link, Unlink, List and Switch (SSO)
- UI support List and Switch (SSO)



DCOS LINKER

DC/OS Cluster Linker

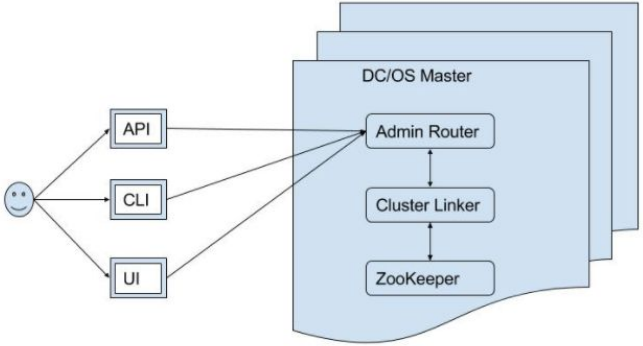
- New component written in “go”
- Runs on all the master nodes (Active - Active)
- Provides HTTP API to link/unlink/list clusters

CLI support

- Link, Unlink, List and Switch (SSO)

UI support

- List and Switch (SSO)



CLUSTER CONNECTIONS

Connect to multiple clusters with DC/OS CLI `dcos cluster` commands

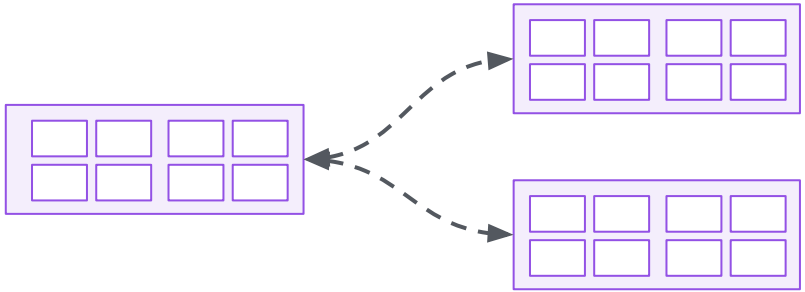
Allows you to setup, attach, rename, and remove clusters

```
dcos cluster setup <dcos_url>
dcos cluster attach <connected-cluster-name>
dcos cluster list

dcos cluster rename <name> <new-name>
dcos cluster remove <name>
```

LINKED CLUSTERS

A cluster link is a **unidirectional** relationship between a cluster and another cluster. A configured link allows easy switching between clusters using the CLI or UI.

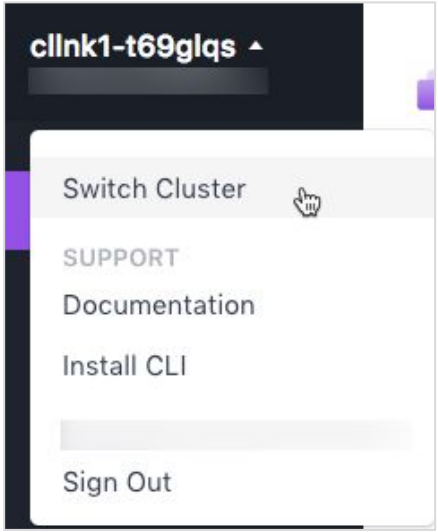


LINKED CLUSTERS

Add and remove links from one cluster to another cluster using DC/OS CLI or the cluster link API.

If the links have been set up using an SSO provider, you will not need to provide credentials to switch clusters.

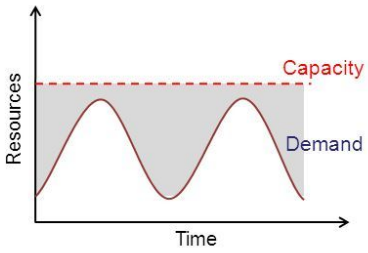
```
dcos cluster link <dcos-url>
Choose the login method and provider to enable
switching to this linked cluster::
1) Provider 1
2) Provider 2
(1-2):
```

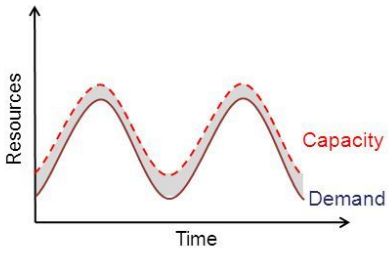


DC/OS Technical

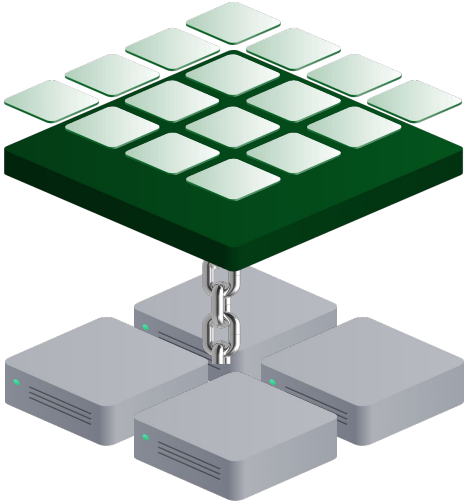
WORKLOAD BURSTING

WORKLOAD BURSTING





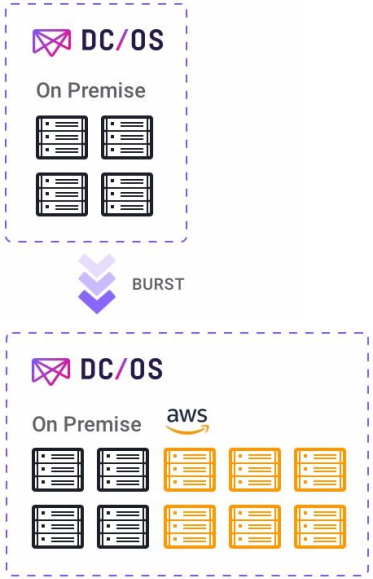
Static Provisioning is as big a problem as Static Partitioning



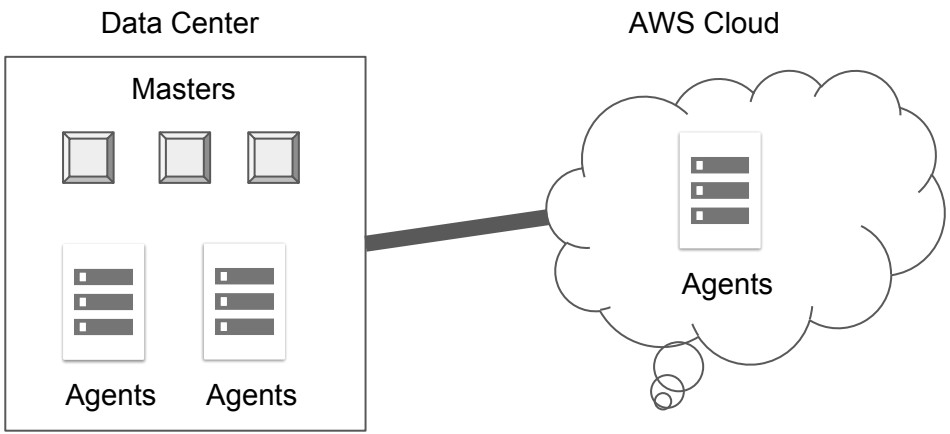
Without dynamic provisioning, container infrastructure is inflexible and susceptible to workload bursts

WORKLOAD BURSTING

- Add DC/OS Agent Nodes in Cloud for peak capacity bursting.
- Schedule stateless services on burst capacity
- Ability to burst for data & stateful services
- Provision data centers for steady state usage ar burst to cloud for peak capacity
- Easily add cloud capacity to service seasonal demand
- Run seasonal batch & stateless services on burst capacity



ARCHITECTURE



WORKFLOW - INSTALL

Create a fault domain detect script and add it to `genconf` folder

- Example script that works for public clouds is provided here

Go through the standard installation procedure!

- Masters and agents will be configured with regions and zones
- NOTE: Make sure to archive the installer (`dcos-install.tar`) for later use

WORKFLOW - ADD CAPACITY

- Provision nodes for installing DC/OS agents in a remote site (e.g., public cloud)

- Copy the archived installation folder from the original installation onto each node
  - Install DC/OS as public/private agent on each node
  - Installer will configure the region and zone based on the fault domain script

- Remote agent nodes will automatically register with the master

APPLICATION BEHAVIOR

- Running apps, pods and service stay in local region
  - Will not be rescheduled to remote region at any cost

- New apps can be launched in remote region by explicit opt-in by users
  - Via new placement constraints

remote-app.json

```
1  {
2    "id": "/sleeper-remote",
3    "instances": 3,
4    "portDefinitions": [],
5    "container": {
6      "type": "MESOS",
7      "volumes": []
8    },
9    "cpus": 0.1,
10   "mem": 128,
11   "requirePorts": false,
12   "networks": [],
13   "healthChecks": [],
14   "fetch": [],
15   "constraints": [
16     [
17       "@region",
18       "IS",
19       "us-west-2"
20     ]
21   ],
22   "cmd": "sleep 1000"
23 }
```

REMOVING REMOTE CAPACITY

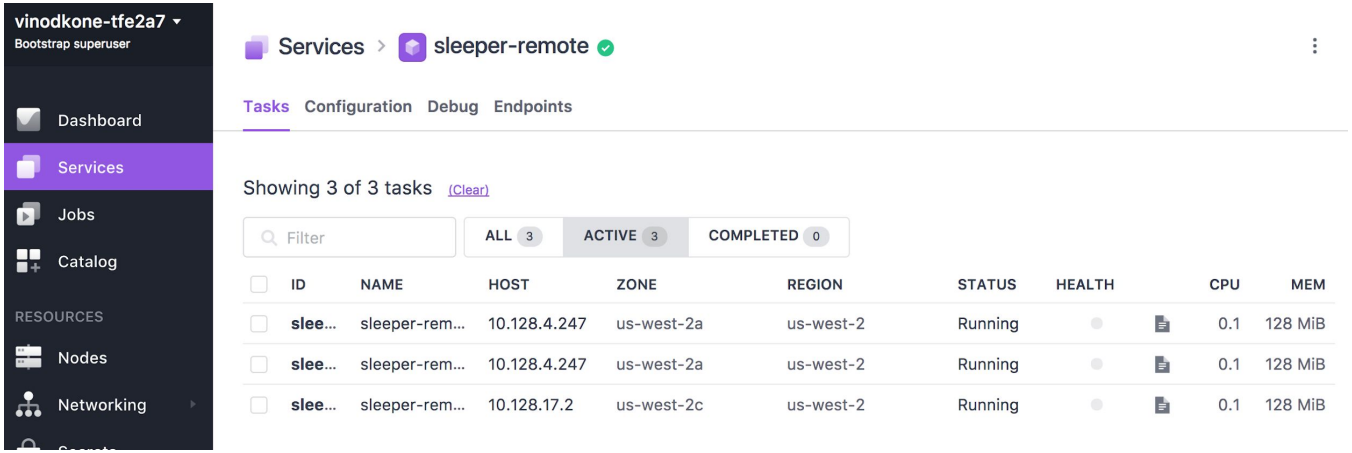
- Shutdown the remote agent nodes as usual
- Inform DC/OS that the remote agents are decommissioned via CLI / API
    - $ dcos node decommission <node-id>
- Frameworks get notified about capacity removal immediately without rate limits!

LAUNCH APP IN REMOTE REGION

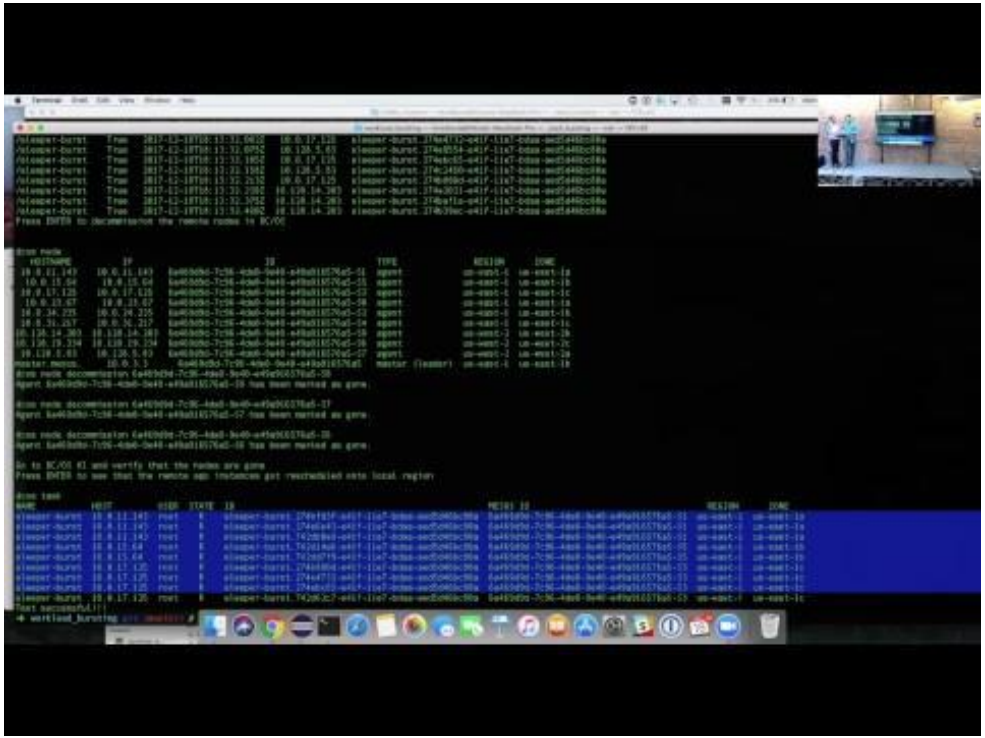
remote-app.json

```
1  {
2    "id": "/sleeper-remote",
3    "instances": 3,
4    "portDefinitions": [],
5    "container": {
6      "type": "MESOS",
7      "volumes": []
8    },
9    "cpus": 0.1,
10   "mem": 128,
11   "requirePorts": false,
12   "networks": [],
13   "healthChecks": [],
14   "fetch": [],
15   "constraints": [
16     [
17       "@region",
18       "IS",
19       "us-west-2"
20     ]
21   ],
22   "cmd": "sleep 1000"
23 }
```

LAUNCH APP IN REMOTE REGION



WORKLOAD BURSTING DEMO



DC/OS Technical

MULTI-DATA CENTER

FAULT DOMAINS

Native Support for Region and Availability Zone(AZ) Semantics in DC/OS.

Loose binding to map Region:AZ to an Agent node located on-prem in a rack or in a cloud.

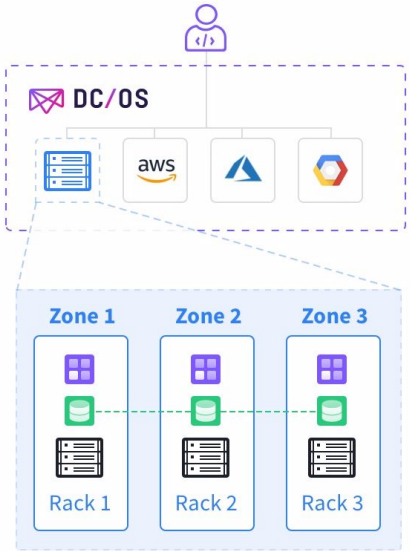
Offer Filtering to enable legacy remote region unaware frameworks.

Auto provision services to withstand fault domain unavailability.

Ability manage service and infrastructure upgrades in a fault domain aware manner

Manage multiple PoPs as Remote Regions.

On-Demand Cloud Bursting into a Remote Region of Compute Nodes.

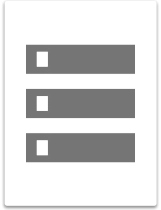


FAULT DOMAINS

Fault Domain: A set of nodes that share similar failure (and latency) characteristics.

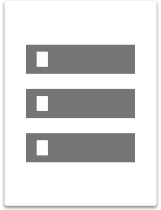
Two kinds of fault domains:

1. **Regions**
   - Highest degree of fault isolation
   - High network latency to a different region
   - Consists of one or more zones
   - Maps to “region” in public clouds and “data center” in on-prem



Rack 1

Fault Domain



Rack 2

Fault Domain

2. **Zones**
   - Moderate degree of fault isolation
   - Low network latency to different zones within a region
   - Maps to “availability zone” in public clouds and “rack” in on-prem

FAULT DOMAIN SOLUTION

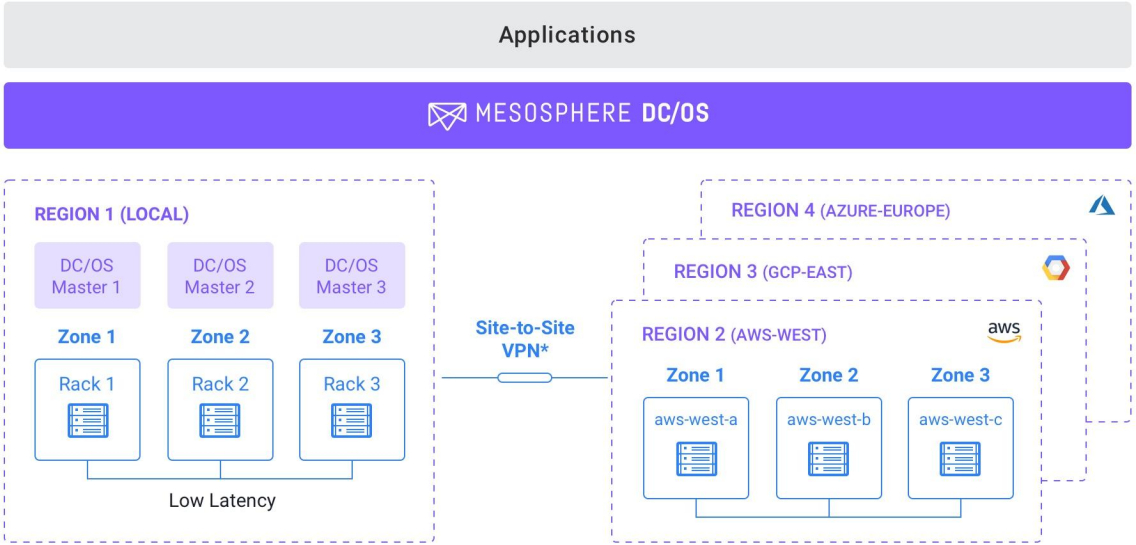
- Masters and agents are configured with a Fault Domain
  - Installer configures fault domain via fault domain detect script
  - Auto detection for public clouds (GCP, Azure and AWS)

- Marathon updated to schedule based on fault domains
  - Region and Zone constraints

TERMINOLOGY

- Default fault domain
  - Fault domain is not configured
- Local Region
  - The region containing masters and local agents
- Remote Region
  - Regions other than local region containing remote agents

BC/DR EXAMPLE



*Max latency between regions is <100ms

- Easily deploy workloads to multiple regions (e.g., to AWS, and also on Azure), to facilitate multi-cloud high availability
- Intelligently define fault domains to recover against this hierarchy to maximize service survivability
- Example:
  Within a region, stateless services recover automatically from failures at the node, cluster, rack, or even site level

LAUNCH HA APP

ha-app.json

```
{
"id": "/sleeper-remote-ha",
"instances": 3,
"portDefinitions": [],
"container": {
 "type": "MESOS",
 "volumes": []
},
"cpus": 0.1,
"mem": 128,
"requirePorts": false,
"networks": [],
"healthChecks": [],
"fetch": [],
"constraints": [
 [
   "@region",
   "IS",
   "us-west-2"
 ],
 [
   "@zone",
   "GROUP_BY",
   "3"
 ]
],
"cmd": "sleep 1000"
}
```

LAUNCH HA APP

vinodkone-tfe2a7
Bootstrap superuser

- Dashboard
- Services
- Jobs
- Catalog

RESOURCES

- Nodes
- Networking

Services > sleeper-remote-ha

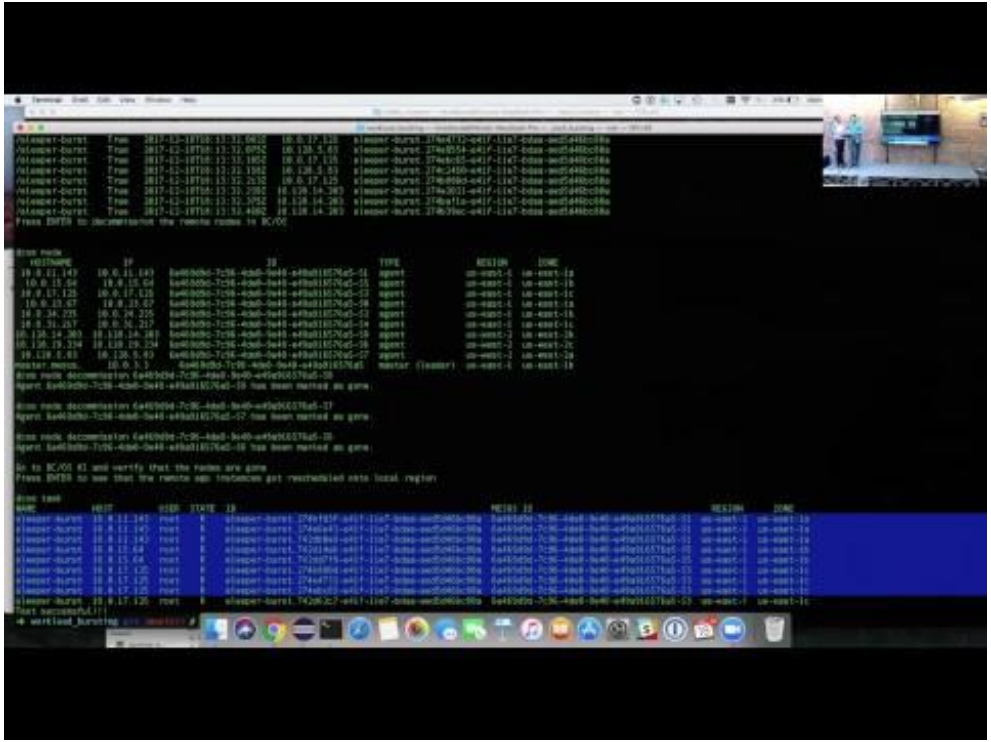
Tasks Configuration Debug Endpoints

Showing 3 of 3 tasks (Clear)

Filter | ALL 3 | ACTIVE 3 | COMPLETED 0

ID	NAME	HOST	ZONE	REGION	STATUS	HEALTH		CPU	MEM
slee...	sleeper-rem...	10.128.13.192	us-west-2b	us-west-2	Running			0.1	128 MiB
slee...	sleeper-rem...	10.128.17.2	us-west-2c	us-west-2	Running			0.1	128 MiB
slee...	sleeper-rem...	10.128.4.247	us-west-2a	us-west-2	Running			0.1	128 MiB

DEMO



Lab 1a

CREATE YOUR CONFIG.YAML

LAB 1A: CREATE YOUR CONFIG.YAML

1. $ `cd /home/centos`
2. $ `mkdir genconf/`
3. $ `vi genconf/config.yaml`
4. Add your cluster configuration
5. Write and quit Vim `:wq`
6. Use `--validate-config` argument to `dcos_generate_config.sh` to validate your configuration file.

Extra Credit

Purposefully break your configuration parameters by giving an invalid IPv4 address, or choosing a out of band port number, then rerun validate.

Lab 1b

CREATE YOUR IP-DETECT SCRIPT

LAB 1B: CREATE YOUR IP-DETECT SCRIPT

1. `$ vi genconf/ip-detect`
2. Write your ip-detect script for AWS:

```
#!/bin/sh
# Example ip-detect script using an external authority
# Uses the AWS Metadata Service to get the node's internal
# ipv4 address
curl -fsSL http://169.254.169.254/latest/meta-data/local-ipv4
```

3. Test your script

Extra Credit

Write a script that can work across all operating systems for any inet interface (even randomly generated iface names).

Lab 1b

EXECUTE CONFIGURATION GENERATION

LAB 1C: EXECUTE CONFIGURATION GENERATION

1. Execute `bash dcos_generate_config.sh --genconf`

Extra Credit

Explore the contents of the created artifacts.

SSH CONFIGURATION

```
$ cat genconf/config.yaml
---
ssh_user: centos
ssh_port: 22
parallelism: 50
...
```

```
$ cat genconf/ssh_key
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAhx70Yc7zSzrqz3OyfoB8S22zmZ4+
jpO+Dir70Qx4p8wtfhcsLjoPaNmArww6
WJdbSUJktDsgpzmJVKS8fAoRnt0XtWmVETiywGKOMCld
FYqdwLjdaWJZ4qqGDs5upMVCcV7cnCf7
TBnkBNv3Az5SyzgBk00zUjcaEiwixPt9vyb4KOpI/WdX
nVAI/nULdv/6DAdWLcCtGudyw9HEc889
Ry2PHWF/6U39WWKG10ln5qYNO2Nf1OBsgI1D5kZlDj5A
LnSbJ4dtH7KG1C/Zg1bOumZ4rLDYICpn
...


NOTE: SSH Key CAN NOT BE ENCRYPTED
```

Lab 1c

PREFLIGHT, DEPLOY, POSTFLIGHT

LAB 1C: PREFIGHT, DEPLOY, POSTFLIGHT

1. Execute `bash dcos_generate_config.sh --preflight`
2. Execute `bash dcos_generate_config.sh --deploy`
3. Execute `bash dcos_generate_config.sh --postflight`

Extra Credit

Login to DC/OS.

SUMMARY

In this module we looked at the following:

- DC/OS Architecture
- DC/OS Installation

QUESTIONS

MESOSPHERE

BACKUP SLIDES

DC/OS Technical

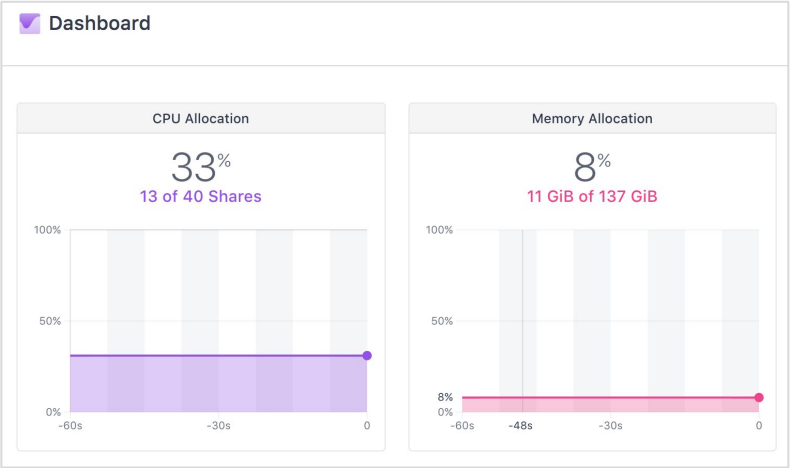
DC/OS UI

DC/OS UI

- Dashboard for DC/OS
- Entrypoint to individual service UIs
- Single endpoint for:
  - Viewing cluster state
  - Managing services
  - View networking stats
  - Configuring security
- Telemetry and component health
- Open source

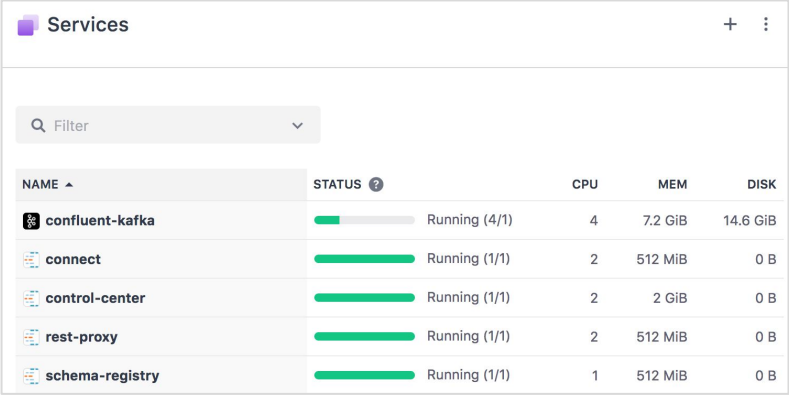
REAL-TIME DASHBOARDS

- CPU
- Memory
- Disk
- Task Failure Rate
- Tasks
- Service Health
- Nodes



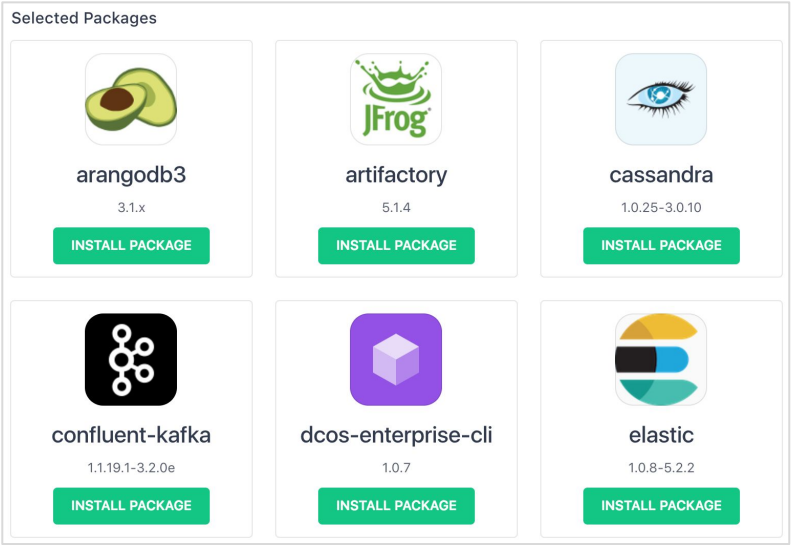
SERVICES

- Manage services through DC/OS UI
- Full marathon functionality
- Jobs with metronome



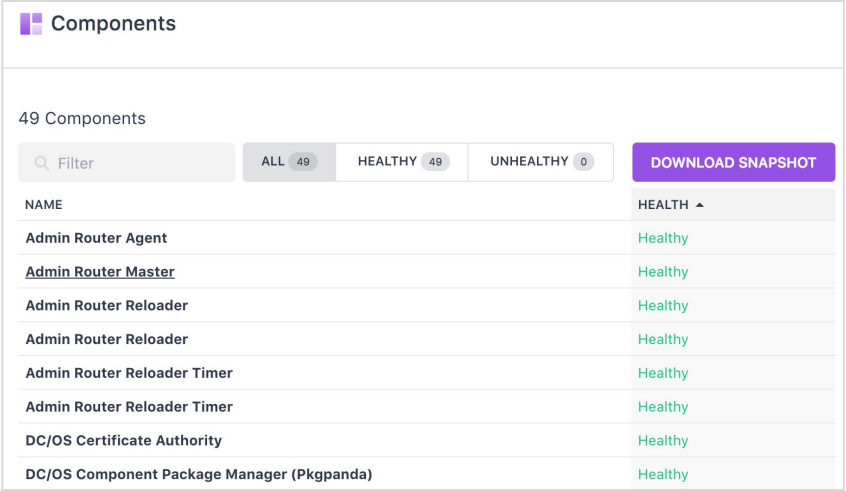
CATALOG

- Install DC/OS services
- Advanced install options
- Uninstall DC/OS services



3DT (DC/OS DISTRIBUTED DIAGNOSTICS TOOL)

- Service on masters which periodically health checks every component via SystemD
- DC/OS-signal
- Data is exposed up through the DC/OS UI



DC/OS Technical

DC/OS CLI

DC/OS CLI

The DC/OS CLI is an API wrapper that allows you to remotely manage multiple clusters, install frameworks, and view cluster state. Examples:

```
dcos package install kafka
dcos node log
dcos kafka broker add 1
```

Each framework can extend with new subcommands (Kafka, Cassandra, etc)

Single binary

DC/OS CLI

```
$ dcos help

Available DC/OS commands:

     cassandra           Communicate with Cassandra-Mesos REST API in DC/OS cluster
     config              Get and set DC/OS CLI configuration properties
     hdfs                Utilities for building and managing your HDFS installation
     help                Display command line usage information
     kafka               Start and manage Kafka brokers
     marathon            Deploy and manage applications on the DC/OS
     node                Manage DC/OS nodes
     package             Install and manage DC/OS packages
     service             Manage DC/OS services
     spark               Run and manage Spark jobs
     task                Manage DC/OS tasks
```

DC/OS SUBCOMMANDS

```
$ dcos spark help

Usage:
    dcos spark --help
    dcos spark --info
    dcos spark --version
    dcos spark --config-schema
    dcos spark run --help
    dcos spark run --submit-args=<spark-args> [--docker-image=<docker-image>
--verbose]
    dcos spark status <submissionId> [--verbose]
    dcos spark log <submissionId> [--follow --lines_count=<lines_count>
--file=<file>]
    dcos spark kill <submissionId> [--verbose]
    dcos spark webui
```

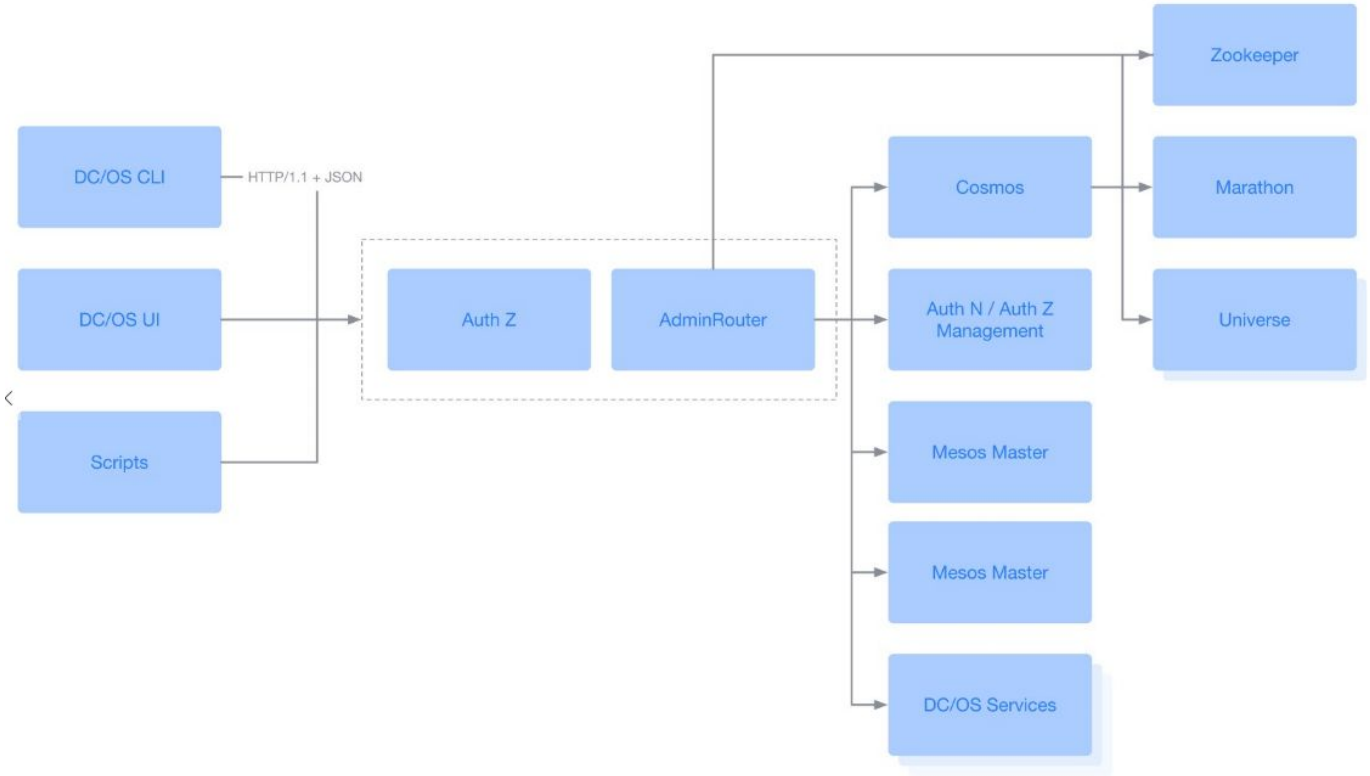
DC/OS Technical

COSMOS

COSMOS

- HTTP server which handles package management
- Responsible for all sub commands of DC/OS package
  - Repo management
  - Package search
  - Package list
  - Package describe
  - Package install / uninstall

COSMOS

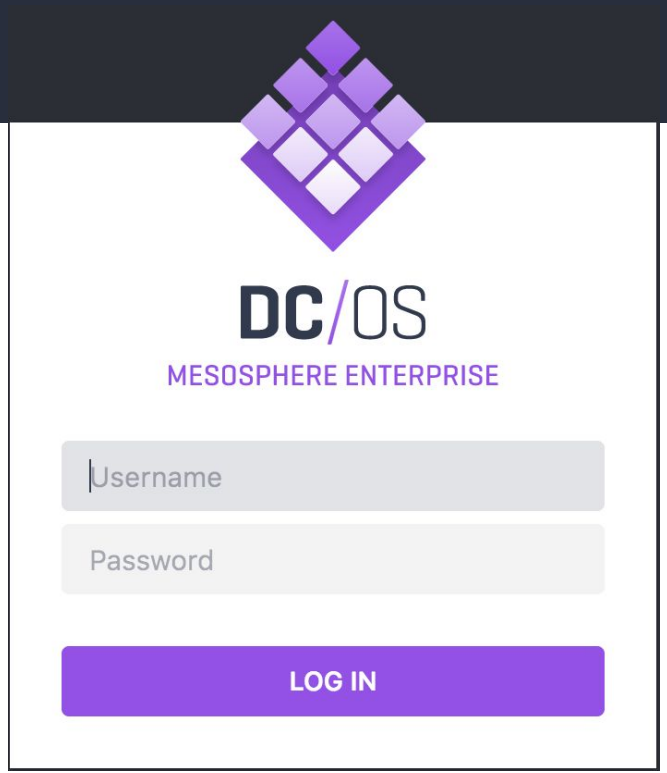


DC/OS Technical

DC/OS IAM

IDENTITY & ACCESS MANAGEMENT

To access DC/OS services (location/service/<servicename>), users must authenticate through the Web UI and be authorized. A service is a resource protected by an ACL. The ACL defines the subject and list of permissions. Implemented via an HTTP server with a RESTful API AdminRouter enforces access control by intercepting and validating auth tokens generated by the DC/OS IAM



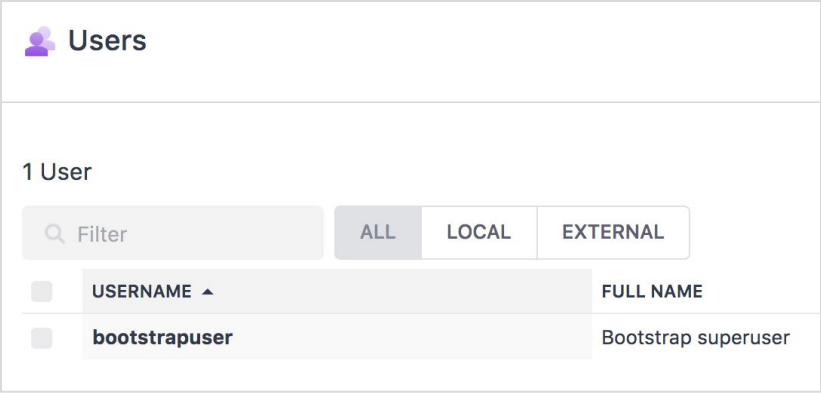
DC/OS IAM

The access control service manages an access control database containing:

- Local users: username, metadata, hashed password
- Groups: groupname, set of local users
- ACLs for service: list of users and groups who have access

Optionally user credentials can be validated against an external LDAP service

A superuser account is specified during install.



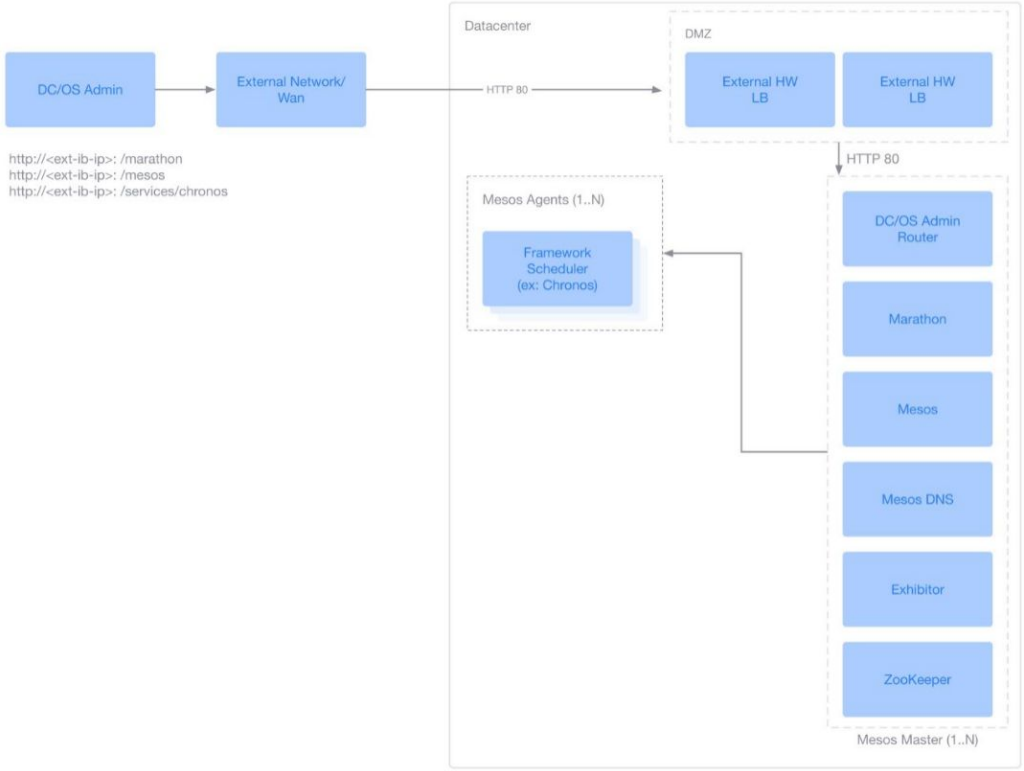
DC/OS Technical

ADMIN ROUTER

DC/OS ADMIN ROUTER

- Entrypoint into the DC/OS cluster
- Nginx reverse proxy
- Routes traffic to DC/OS services and frameworks
- Installed on each DC/OS Master

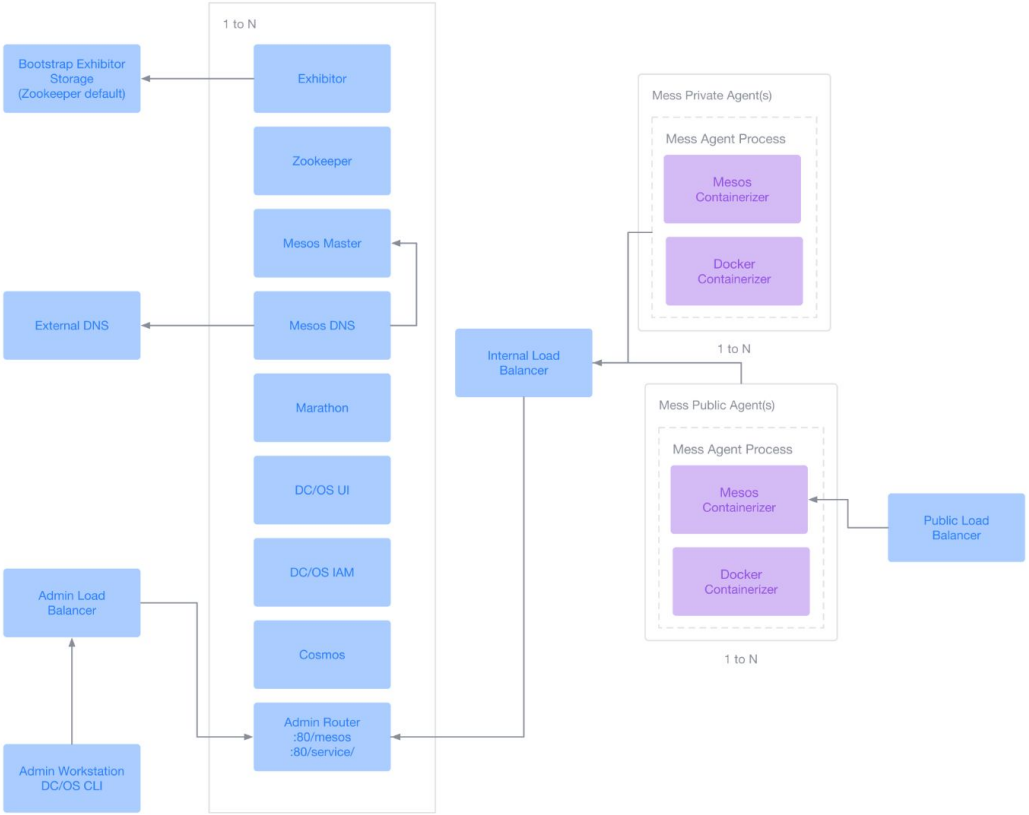
DC/OS ADMIN ROUTER



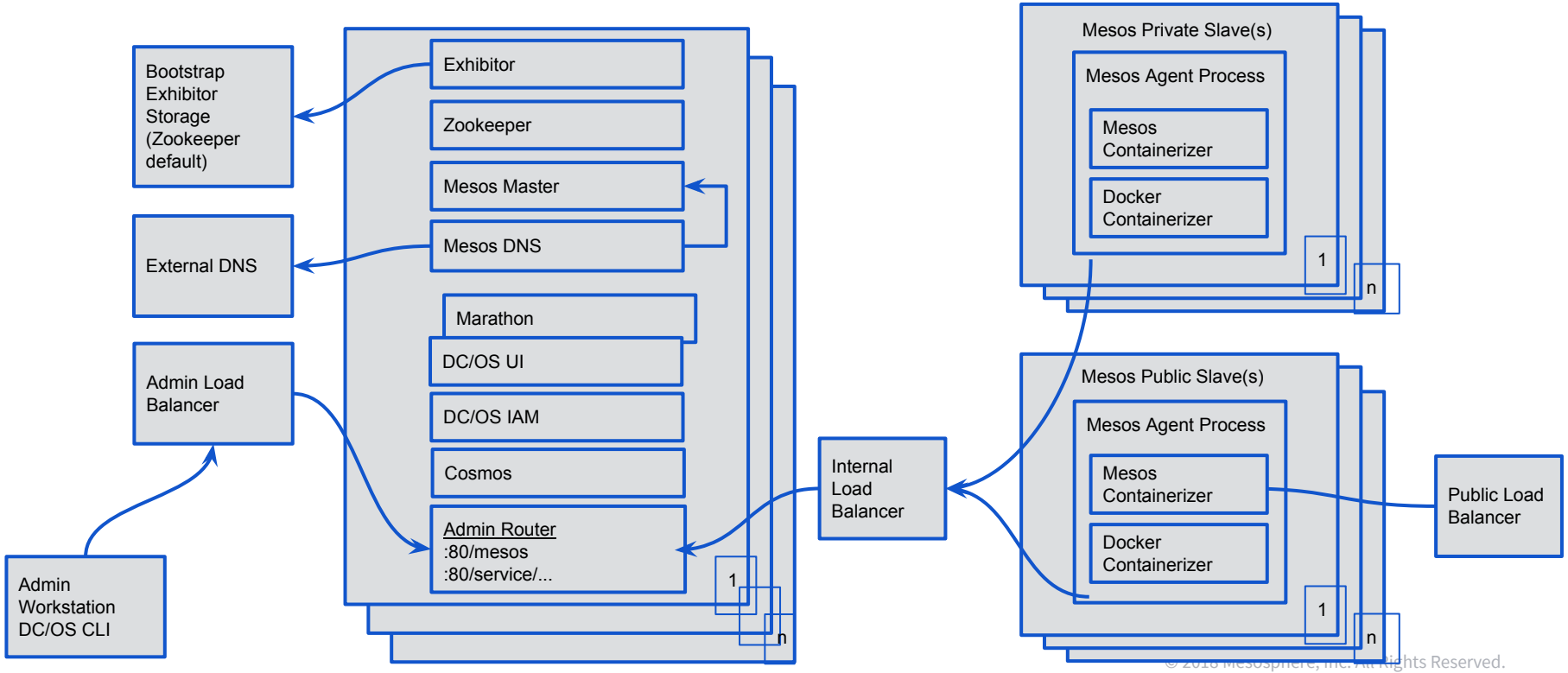
DC/OS ADMIN ROUTER

Component	Open Source Mesos Host : Port	With DC/OS & Admin Router via HTTP
Mesos Master Web UI / API	<mesos-master>:5050	<mesos-master> /mesos/
Mesos Agent UI / API	<mesos-agent>:5051	<mesos-master> /slave/<slave-id>
Exhibitor UI / API	<mesos-master>:8181	<mesos-master> /exhibitor/
Marathon UI / API	<mesos-master>:8080	<mesos-master> /marathon/
Mesos-DNS API	<mesos-master>:8123	<mesos-master> /mesos_dns/v1/...
Frameworks UI	<framework-scheduler> : xyz	<mesos-master> /service/<service-id>
DC/OS History Svc API	N/A	<mesos-master> /DC/OS-history-service/history/...

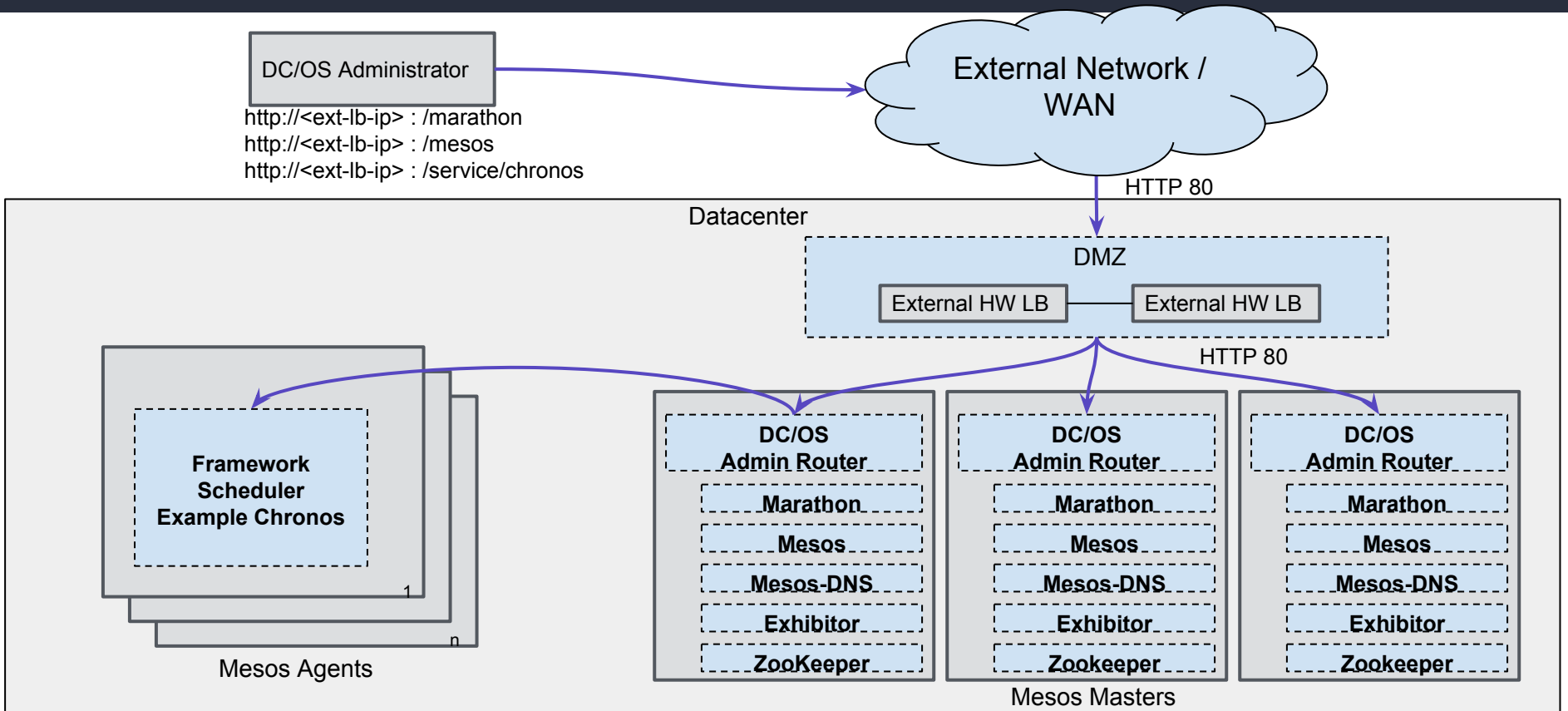
ARCHITECTURE



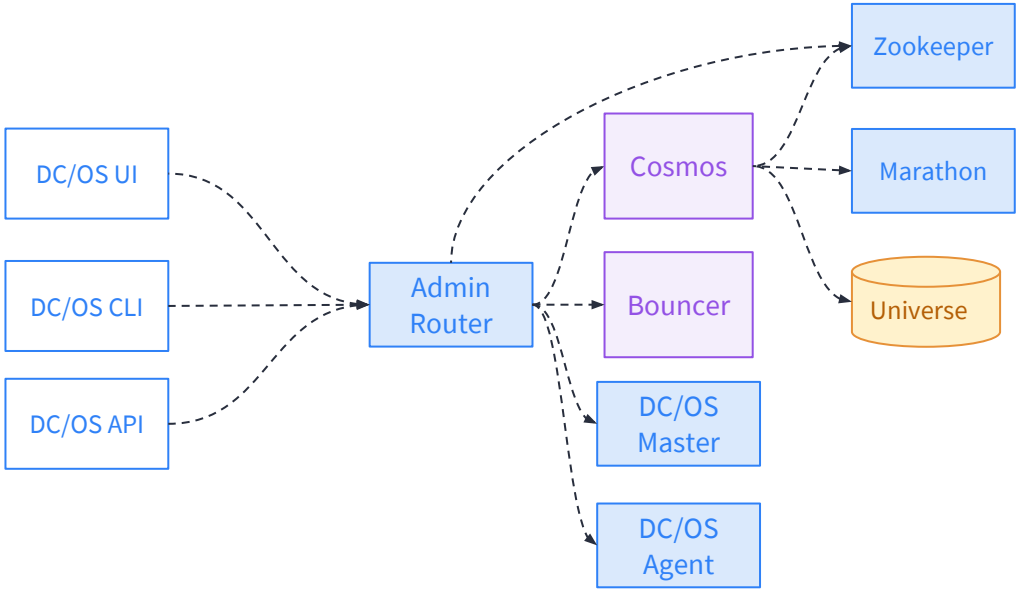
ARCHITECTURE



DC/OS ADMIN ROUTER



COSMOS



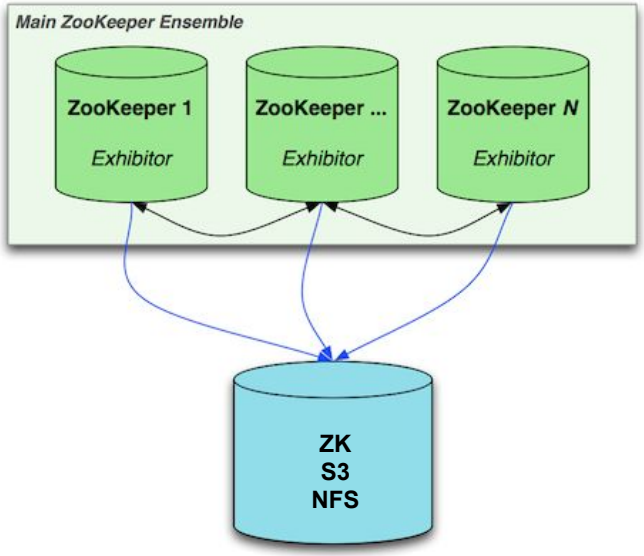
EXHIBITOR

Exhibitor is a supervisor for ZooKeeper

For DC/OS, Exhibitor is installed first to bring up the DC/OS ZK service.

External storage is needed to bootstrap the Exhibitor service (i.e. ZooKeeper, S3, NFS).

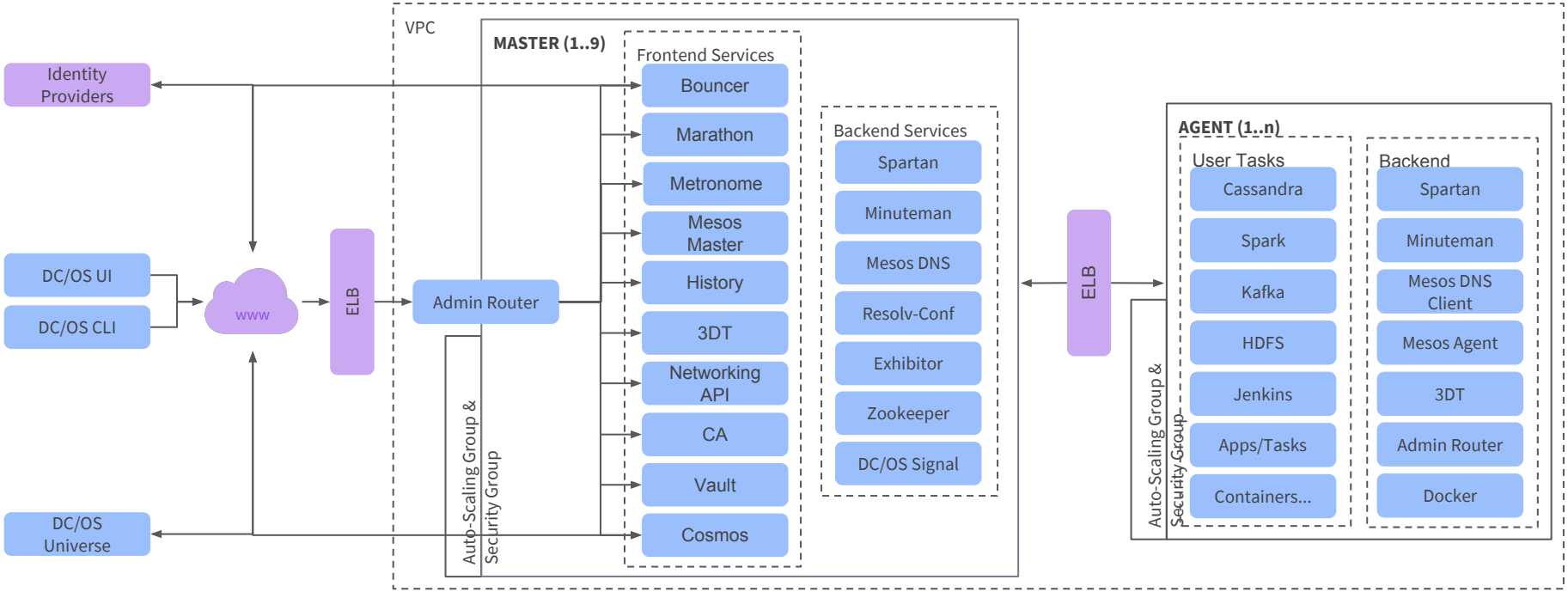
For production, ensure that the external storage backend is highly available.



MESOS-DNS

- In a non-DC/OS environment, you must update **/etc/resolv.conf** on the master and agent nodes to point to mesos-dns as first entries if you have multimaster configuration. Domains other than .mesos will use the resolvers specified in mesos-dns.json.
- In DC/OS Environment, resolv.conf is updated and kept current by the gen_resolvconf.py application
- DNS records will be created with convention
  - `<task-name>.<framework-name>.mesos`
- If you create new Marathon application in DC/OS
  - **# DC/OS marathon app add hello-marathon.json**
  - and your hello-marathon.json specifies an app ID of **hello**
  - your application would have an A record of **hello.marathon.mesos**

ARCHITECTURE



SPECIFIC REQUIREMENTS

Disk Space and Performance Characteristics for /var/lib

The bulk of the disk space will be consumed from `/var/lib` for the Mesos (/var/lib/mesos) and Docker (/var/lib/docker) containers

Ensure enough disk space with the desired performance characteristics (RAID/SSD) allocated to /var/lib

/opt must not be mounted on a separate partition/volume

/opt must be on the root (/) filesystem because of this limitation of systemd: https://lists.freedesktop.org/archives/systemd-devel/2016-May/036646.html

AZURE ARM TEMPLATES

```
$ azure config mode arm
$ azure group create <globally_unique_resource_group_name> <region>
$ azure group deployment create --template-file <ARM_template_file_location> --resource-group
<your_globally_unique_resource_group_name> --name <deployment_name/resource_group_name>
$ azure group delete <your_resource_group>
```

Use Azure CLI version 0.10.0

Ensure region has enough quota