Mesosphere

DC/OS

Architecture and Troubleshooting

Arthur Johnson
Solutions Architect, Mesosphere
ajohnson@mesosphere.io

Mesosphere - Field Engineering - Troubleshooting

Presentation Overview

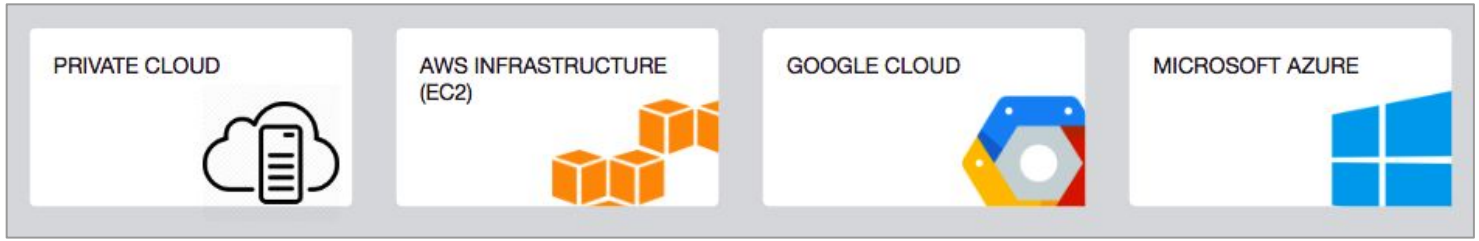
In this session, the following topics will be discussed/presented:

- DC/OS Architecture and Theory of Operations
  - Examine high-level DC/OS architecture and components to understand troubleshooting points
  - Discuss how DC/OS operations to know where to look for issues
- Troubleshooting DC/OS Installations
  - List common causes which create trouble for DC/OS deployments
- Troubleshooting Strategy and Application Deployments
  - Discuss strategy and approach for troubleshooting DC/OS and apps
- Diagnostics and Tools for Debugging
- Troubleshooting Examples (Adventures in Troubleshooting)

Topic One

DC/OS Architecture and Theory of Operations

ANY INFRASTRUCTURE



- AWS, Azure, and Google Cloud are all supported
- On prem datacenters / Private clouds
- Virtualization - VM servers

DC/OS Architecture

TYPES OF NODES

- DC/OS Clusters are made up of two main types of nodes: master nodes and agent nodes
- Master nodes run the cluster and agent nodes run the services and applications
- Agent nodes are often referred to as “slaves” in both the config files and the docs

DC/OS Architecture

CONCEPTS: CLUSTER COMPONENTS

DC/OS cluster

- Bootstrap node
- Master node (quorum)
- Agent node(s)
    - Private
    - Public
- DC/OS services
- System service
- User service

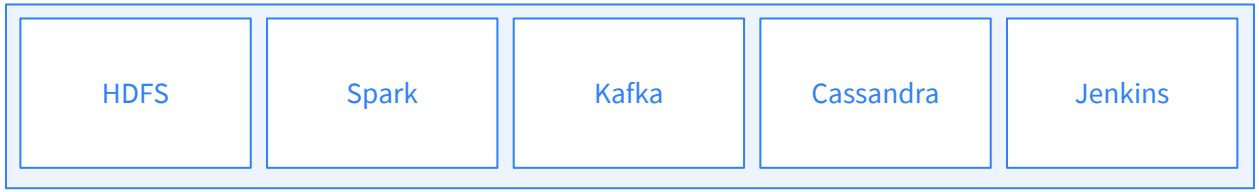
DC/OS

- Job
- Scheduler
- Component
- Package
- Package registry

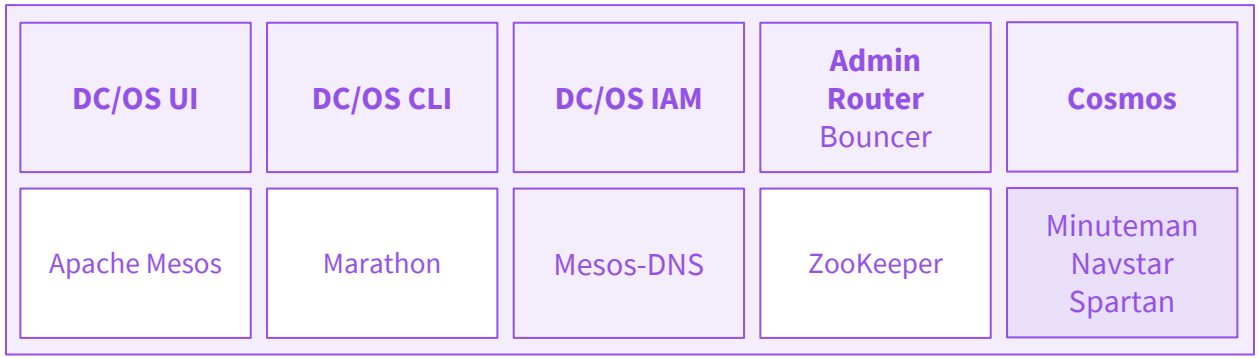
DC/OS Architecture

DC/OS COMPONENTS

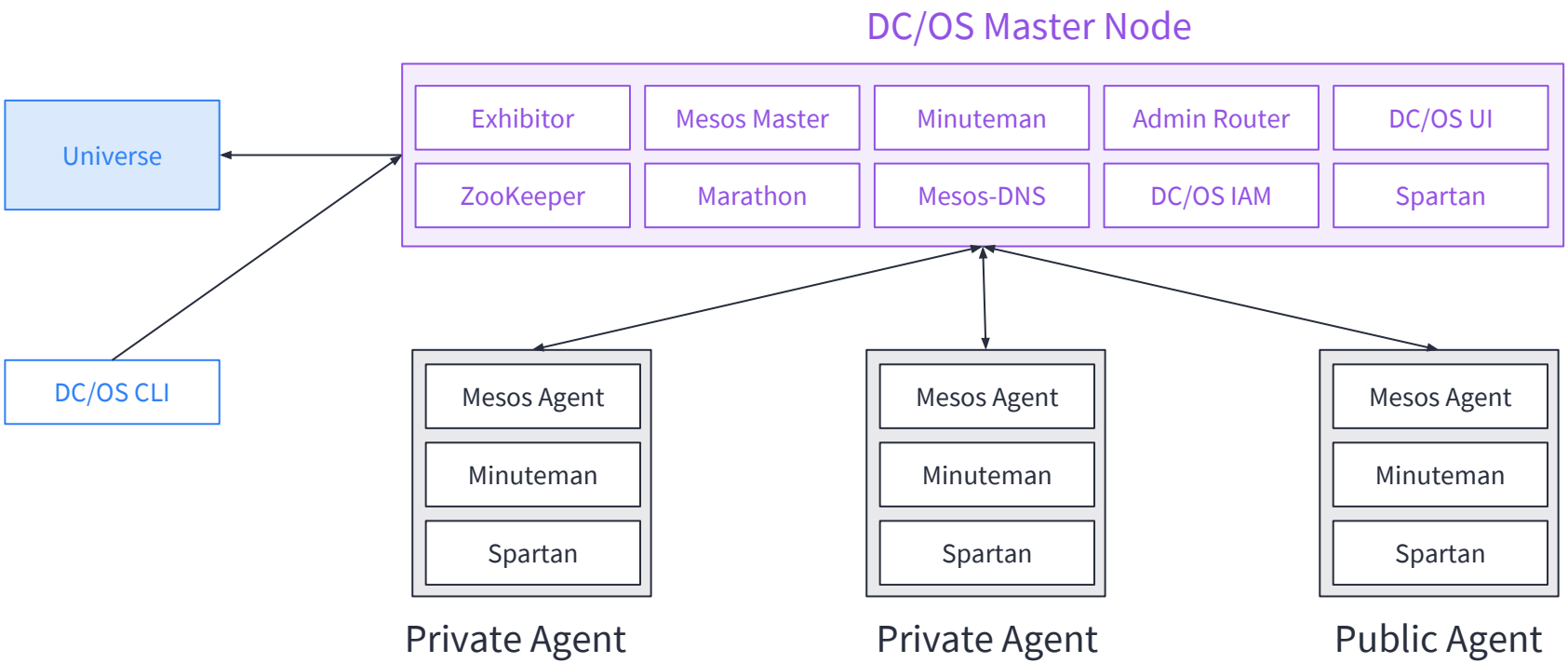
Frameworks



Mesosphere DC/OS

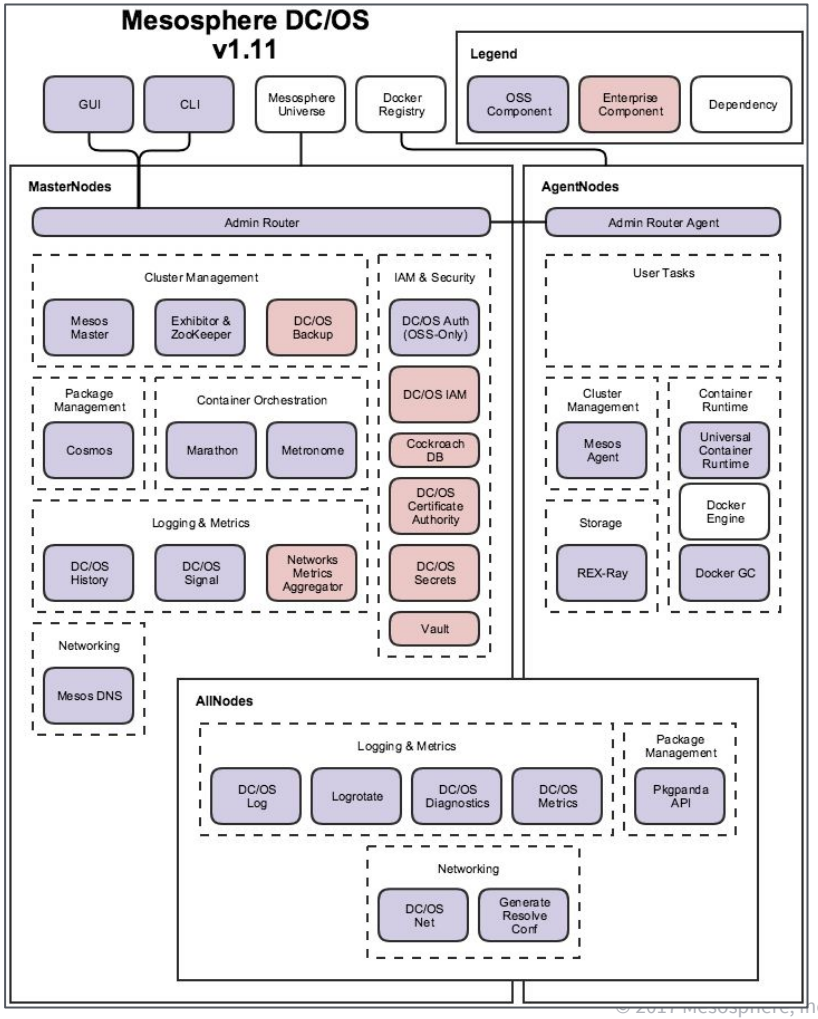


DC/OS ARCHITECTURE



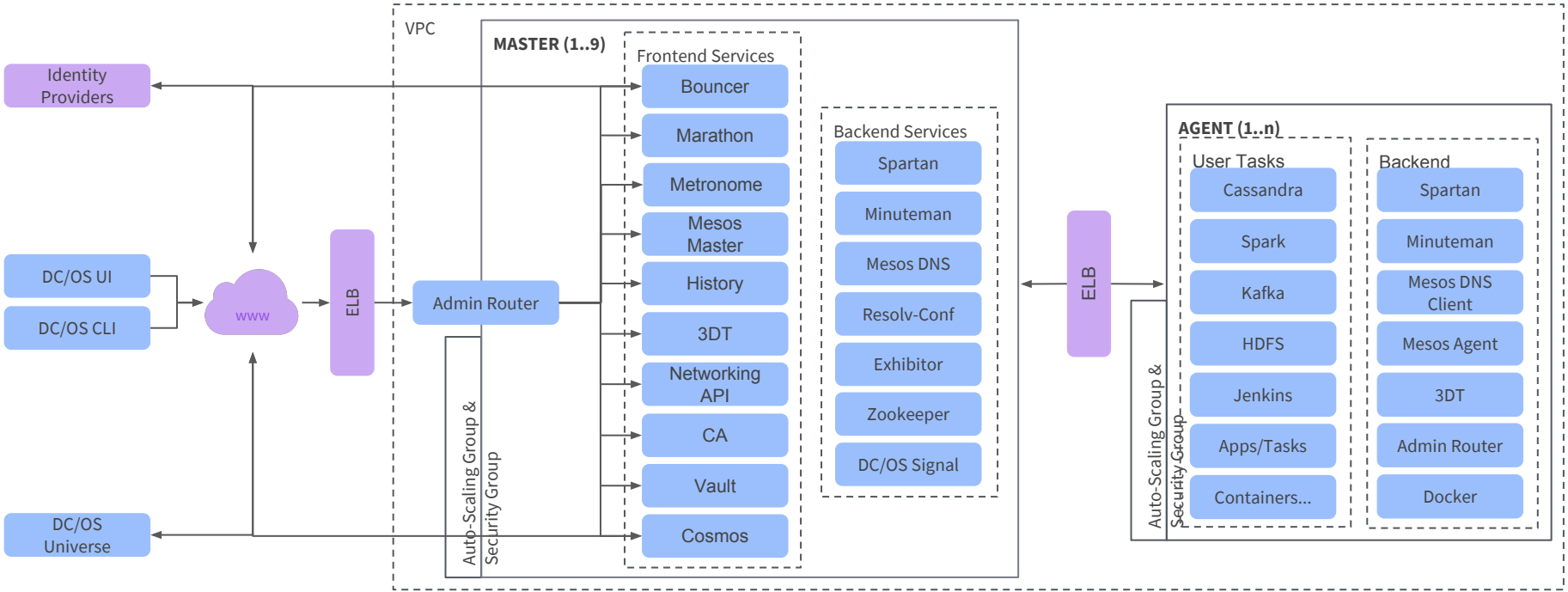
DC/OS Architecture

DC/OS ARCHITECTURE - V1.11 STANDARD

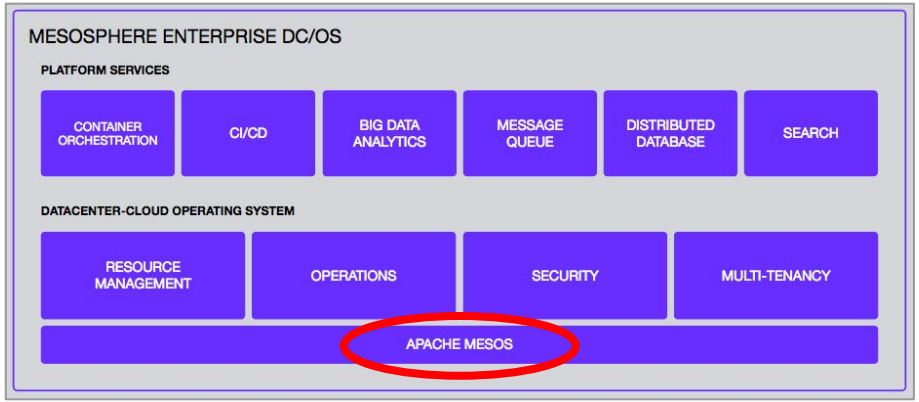


DC/OS Architecture

LOGICAL ARCHITECTURE

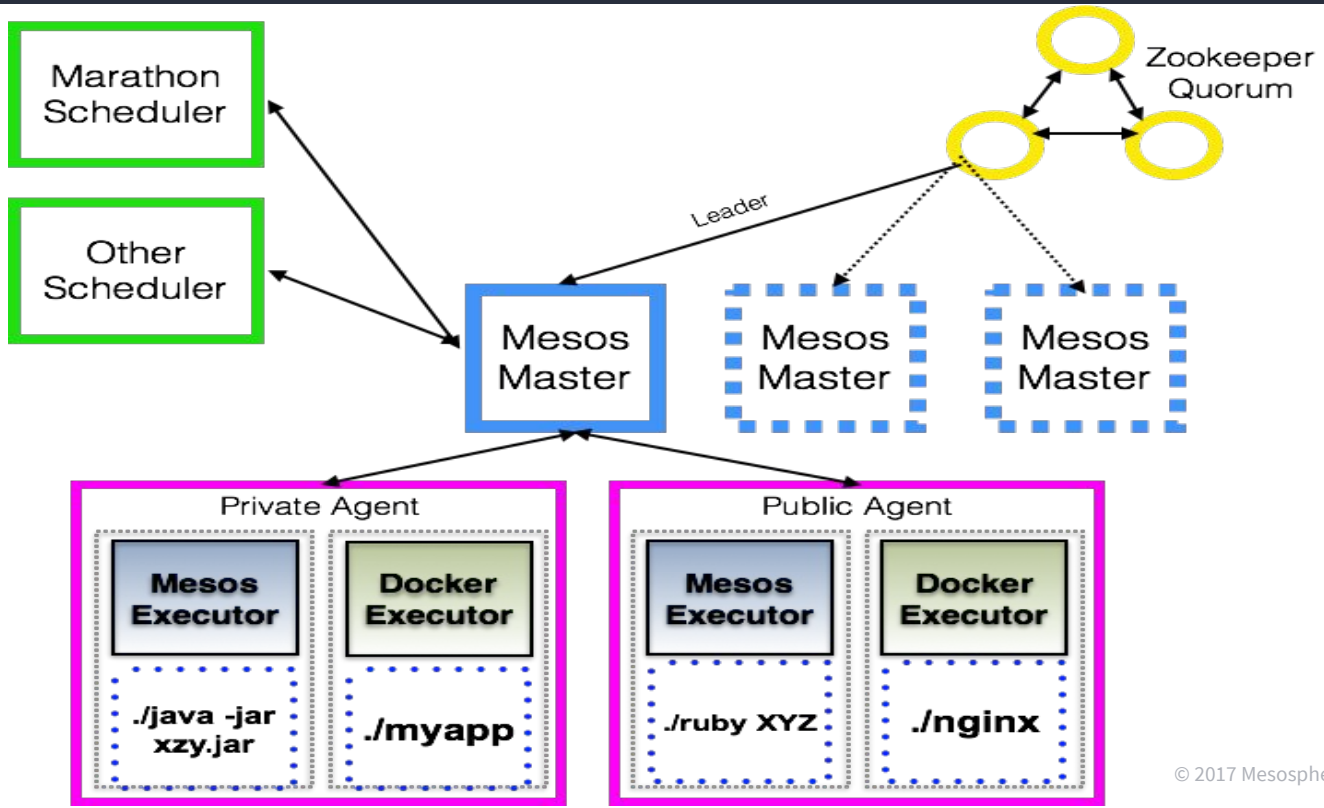


MESOS LAYER



- Apache Mesos is the open source foundation DC/OS is built upon
- Apache project: http://mesos.apache.org/
- DC/OS 1.11 is packaged with Mesos 1.5.0
- Provides “kernel-like” functionality to DC/OS

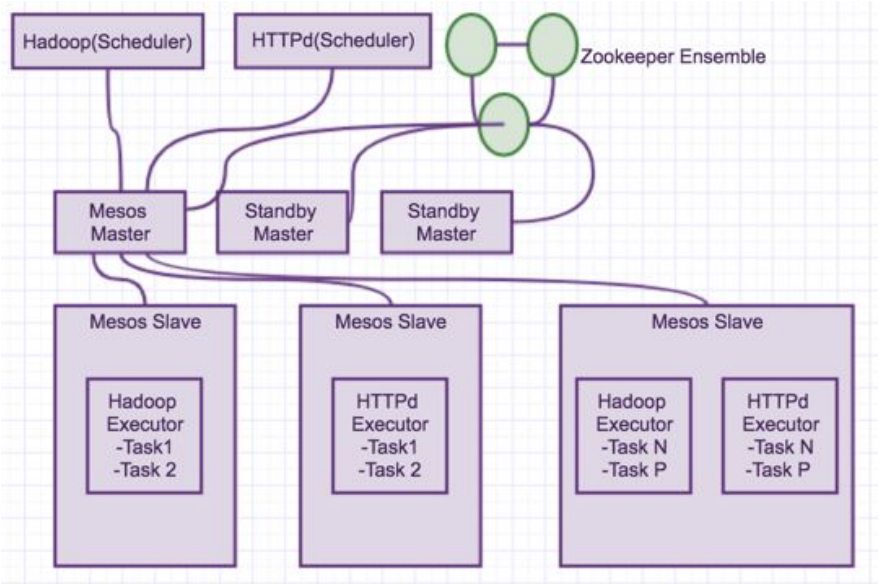
MESOS ARCHITECTURE



DC/OS Theory of Operations

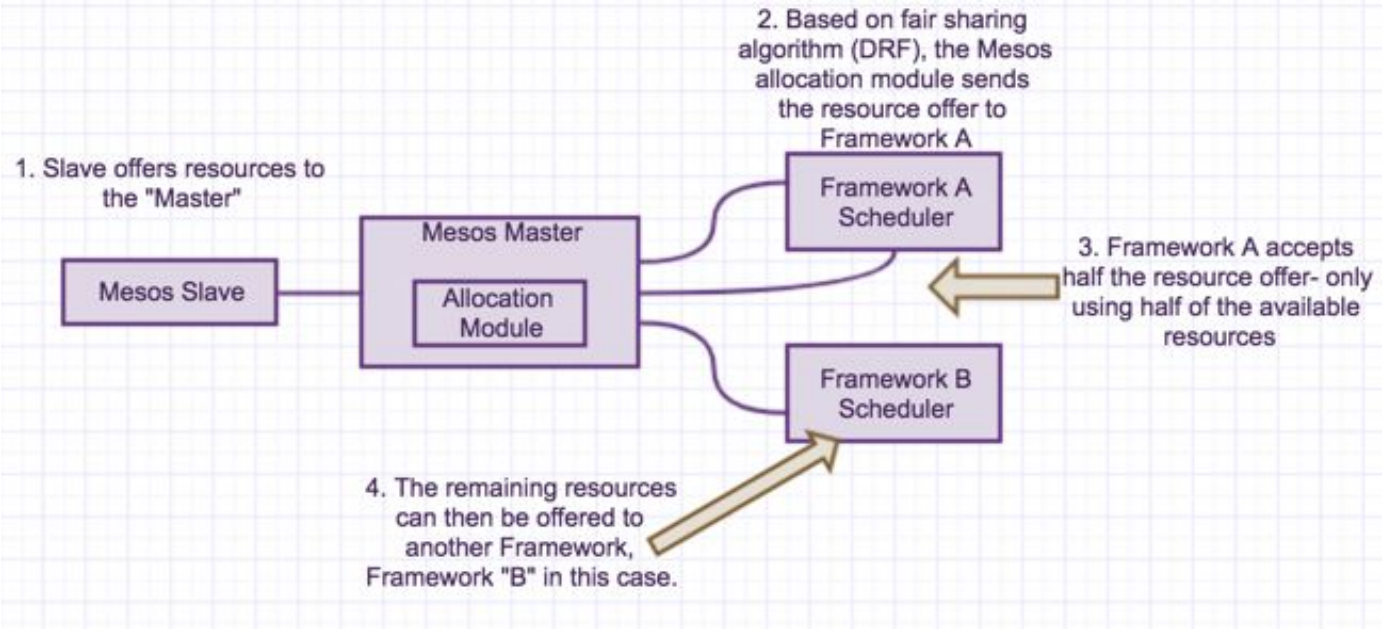
RESOURCE OFFERS & SCHEDULING

- Clusters consist of “Masters” and “Slaves”
  - Slave nodes perform the application processing
- Resource scheduling is the responsibility of the Master node(s)
  - Only one master in control at a time
  - Resource offers from slaves are sent to the Master’s allocation module
  - Allocation module is then responsible to making these offers available to various framework schedulers
- Default resource allocation is done with “Dominant Resource Fairness” (DRF)
  - DRF maximizes the minimum dominant shares across all users

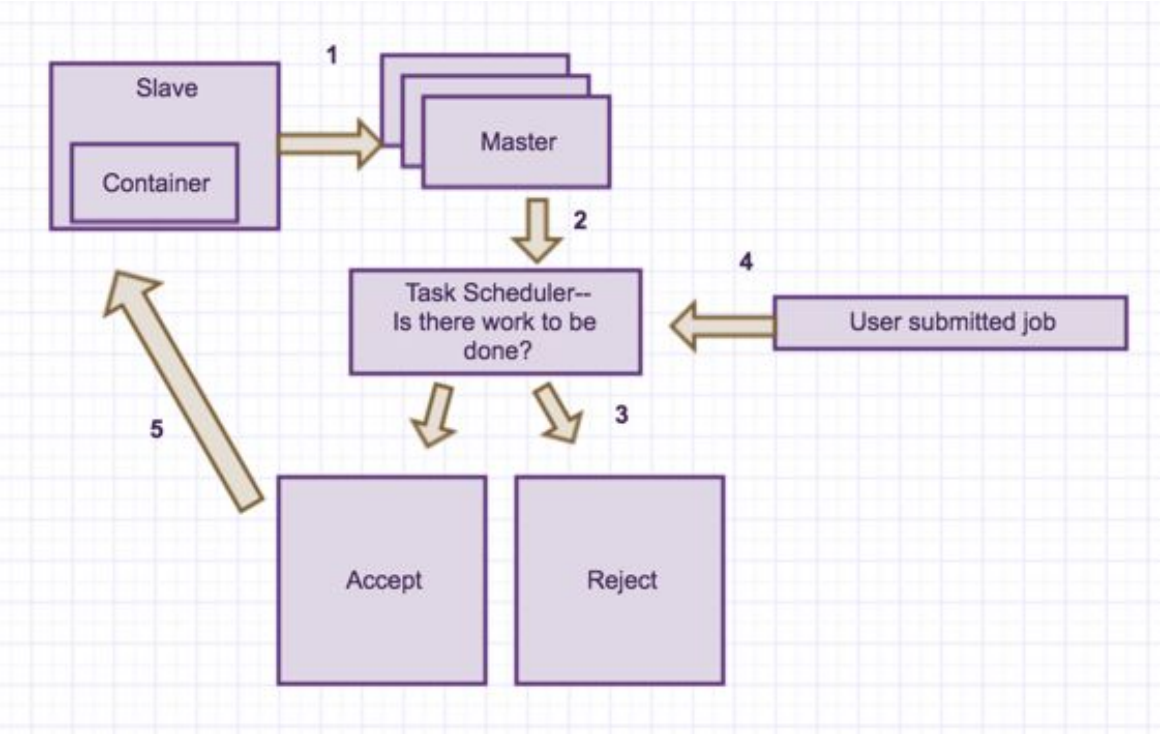


RESOURCE SCHEDULING OVERVIEW

- Other frameworks can use unallocated resource offers



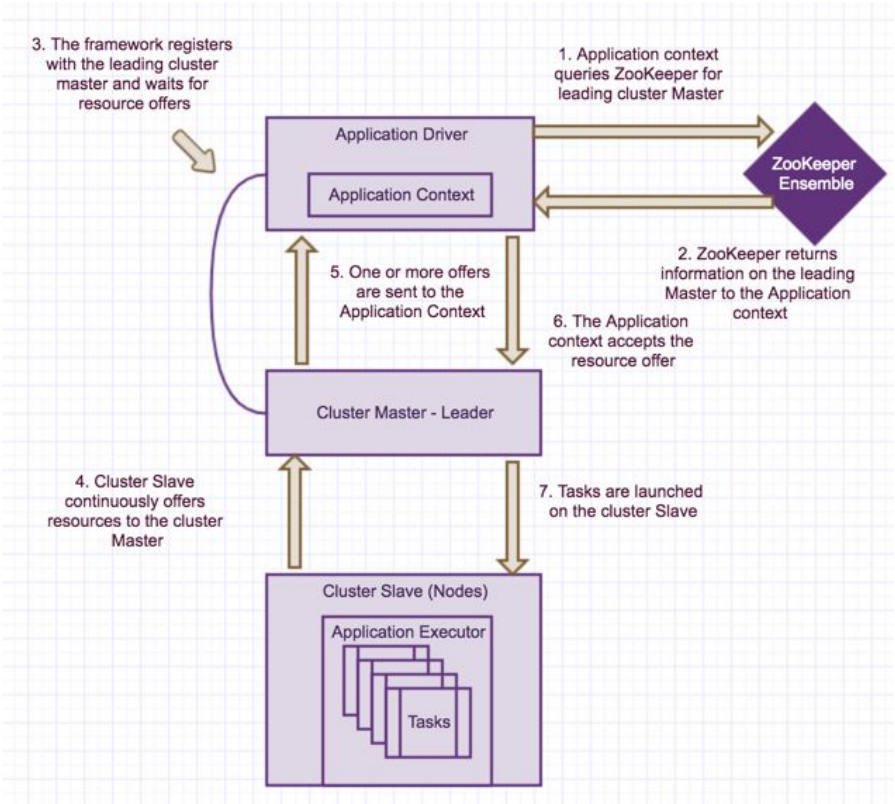
EVENT STREAM FOR ALLOCATED RESOURCE OFFERS



SAMPLE EVENT STREAM

This is a sample event stream for an application running on a DC/OS cluster

- These events occur when an app runs on a cluster



DC/OS FRAMEWORKS

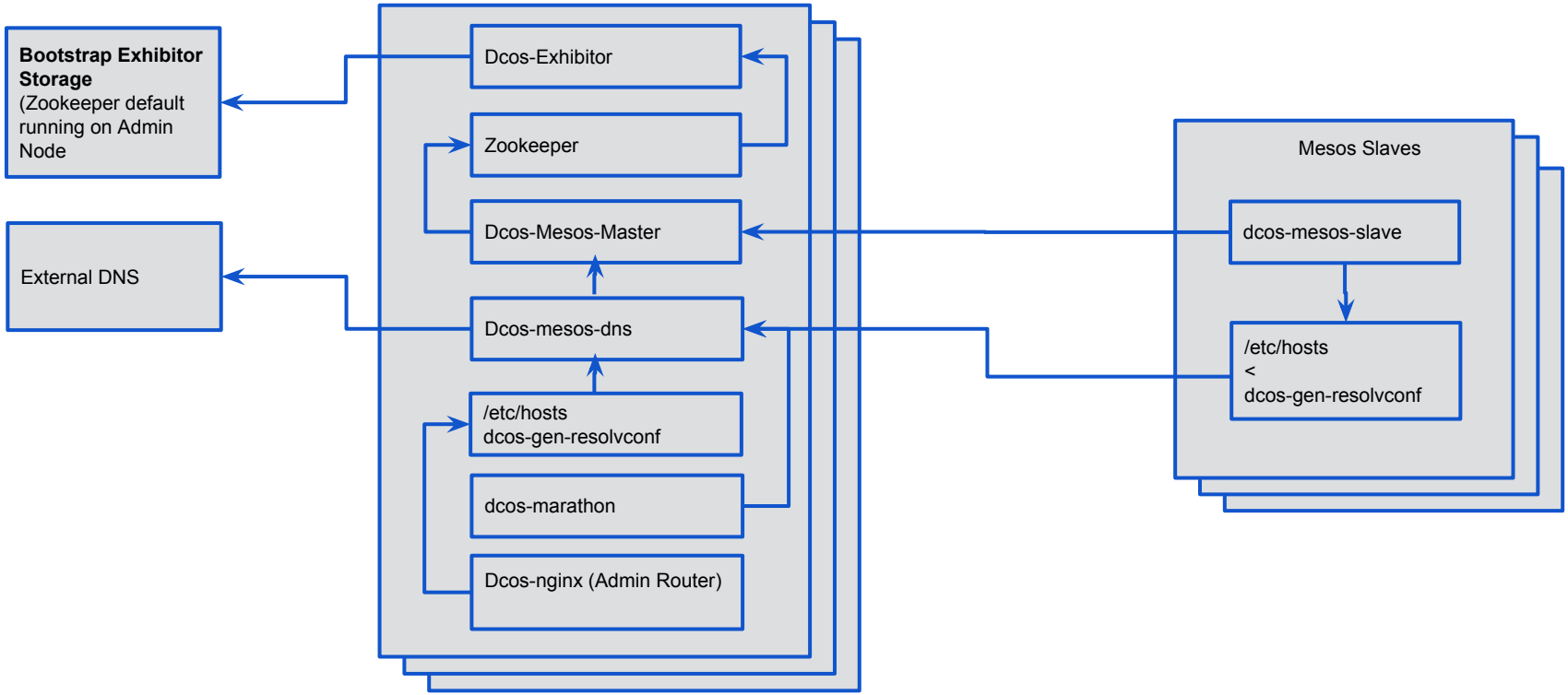
- A framework is any application that is responsible for scheduling and executing tasks on a cluster. A framework has two components:
  - Scheduler
  - Executor
- Scheduler
  - A long running service responsible for connecting to a Master and accepting or rejecting resource offers
- Executor
  - A process launched on a Slave that runs a framework's tasks on the Slave
  - Runs natively as a Mesos Container or Docker (with Docker engine)

CLUSTER DESIGN CONSIDERATIONS

- One or more Master Nodes
  - If the number is greater than 1, it must be an odd number
  - 1 node suffers no failures
  - 3 nodes suffer one failures
  - 5 nodes suffer multiple failures
- One or more Slave Nodes
- Zookeeper ensemble
  - Greater than 1 node, must also be an odd number
  - Co-exists on the Master Nodes
- Docker engine running on the Slave Nodes
- Zookeeper is required for coordination in the cluster and for Master leader election

ZooKeeper cluster size (nodes)	**Quorum Size**	**Number of Machine Failures Tolerated**
1	1	0
3	2	1
5	3	2
Master Cluster Size (nodes)	Quorum Size	Number of Machine Failures Tolerated
2 x N - 1	N	N - 1

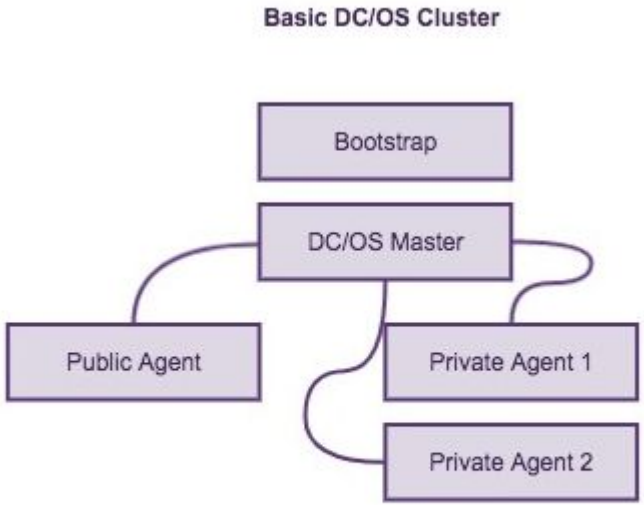
DEPENDENCIES



Topic Two

Troubleshooting DC/OS Installations

SIMPLE CLUSTER INSTALLATION



Details: CentOS 7.3; DC/OS 1.10

CLASSIFICATION OF ISSUES

DC/OS is a complex, distributed platform, comprised of many different individual components and subsystems. As such, troubleshooting such environments can be difficult and time consuming.

- Potential issues:
  - Installation issues resulting from configuration parameters; OS configuration
  - Networking issues
  - Name resolution issues
  - Operating environment issues
  - DC/OS internal issues
  - Incorrect usage of the cluster, assumption on how the DC/OS solution operates

DC/OS INSTALLATION TROUBLESHOOTING ORDER

Hopefully, default installations of DC/OS will work correctly from initial install. However, if they don't the order of investigation of issues should be as follows:

1. Exhibitor
2. Mesos Master
3. Mesos-DNS
4. DNS-Forwarder (Spartan)
5. DC/OS Marathon
6. Jobs
7. Admin Router

CONFIG.YAML: CLUSTER CONFIGURATION STARTING POINT

`security: disabled`
- All Enterprise DC/OS encryption and HTTPS turned off

`security: permissive`
- Cluster encryption turned on, but unencrypted frameworks still allowed

`security: strict`
- Encrypt all the things!

`bouncer_expiration_auth_token_days: '0.5'`
- Auth timeout specified in days including fractions using the Python float syntax in quotes
- Note the default is 5 days

`resolvers:`
- Authoritative upstream DNS resolvers outside the cluster – max 3
- Care should be taken since these can only be changed by a full upgrade of the cluster

`exhibitor_storage_backend: static`
- Use this setting to run Zookeeper on your master nodes and store its data locally on the masters

Examples in the docs: https://docs.mesosphere.com/1.11/installing/ent/custom/configuration/configuration-parameters/

CONFIRMING CORRECT IP

- DC/OS Clusters are configured at installation defining their IP address from the ~/genconf/ip-detect script -- if this script has issue, the DC/OS nodes may not correct find and report their internal IP address
- Contents of ~/genconf/ip-detect (AWS installation)
  - `#!/bin/sh`
  - `curl -fsSL http://169.254.169.254/latest/meta-data/local-ipv4`
- Verify that DC/OS nodes are using the correct internal IP address by running:
  - `/opt/mesosphere/bin/detect_ip`

VERIFYING EXHIBITOR

There are three activities to ascertain the functionality of Exhibitor and whether the cluster is up and available:

1. Verify that Exhibitor is up and running
   a. `Curl http://<masterIPaddr>:8181/exhibitor`
2. If Exhibitor is not running, ssh to the master and
   a. `Sudo journalctl -flu dcos-exhibitor`
3. Check the output of /exhibitor/v1/cluster/status to ensure all masters are joined and one is the ‘leader’
   a. `Curl -fsSL http://<masterIPaddr>:8181/exhibitor/v1/cluster/status |jq`

VERIFY OTHER DC/OS COMPONENTS

- Running these commands will verify the service/service logs and any issues

COMPONENT	Command
Admin Router	journalctl -u dcos-adminrouter -b
Spartan	journalctl -flu dcos-spartan; ping ready.spartan
Mesos-DNS	journalctl -flu dcos-mesos-dns
Mesos-Master (on the masters)	journalctl -flu dcos-mesos-master
Mesos-Slave (on the slaves)	journalctl -u dcos-mesos-slave -b
Marathon	journalctl -u dcos-marathon -b
gen_resolvconf	journalctl -u dcos-gen-resolvconf -b
Exhibitor	journalctl -u dcos-exhibitor -b

Topic Three

Troubleshooting Strategy and Application Deployments

DC/OS Troubleshooting Strategy

DEBUGGING STRATEGY

Debugging a distributed system such as DC/OS can be a daunting and time-consuming endeavor. Knowing where to start is half the battle! Start in the following order:

- Task logs
- Scheduler logs
- Mesos Agent
- Task interactive
- Master

POTENTIAL ISSUES

Potential cluster issues:

- Master node leader election
- Agent nodes not registered (registering)
- Incorrect time on nodes within cluster
- Incorrect reported cluster resources
- High workloads, queuing of tasks, delayed deployments

Potential application issues:

- Apps not deploying
- slow/delayed deployments
- Deployed but not starting and running correctly
- Apps repeatedly restarting
- Apps not responding, or not reachable inside or outside of the cluster

COMPONENTS AND TROUBLESHOOTING BASICS

Component	Service Name	Layer	Dependencies	Subcomponent	Description	Impact to Environment	Where to begin
AdminRouter	`dcos-adminrouter.service`	DC/OS	dcos-master; leader.mesos:5050		Control plane proxy for DC/OS admin and components	Loss of the adminrouter component will prevent control plane access (management) to the cluster. Loss of adminrouter on the agents will prevent management via the UI/CLI to agents and will need to be recitified on the local agent itself. Loss of agent from management UI.	Check status of the service: `systemctl status dcos-adminrouter.service`; Check logs of adminrouter from boot to see if there are issues: `journalctl -u dcos-adminrouter -b` (looking for exit codes); restart admin router: `systemctl restart dcos-adminrouter`; Inspect logs for running adminrouter: `journalctl -flu dcos-adminrouter`
Exhibitor	`dcos-exhibitor.service`	DC/OS	Zookeeper		Zookeeper supervisor	Cluster master node communication and quorum voting is conducted via Zookeeper. Monitoring and watching of Zookeeper is accomplished via Exhibitor. Losing Exhibitor is tantamount to losing quality and reliability of the entire cluster.	Verify that Exhibitor is up and operational at: `http://<master-node-ip-addr>:8181/exhibitor`; on a master node, check the Exhibitor logs: `journalctl -flu dcos-exhibitor`; Check that all masters are in the cluster and serving and one is elected leader- from a master: `curl -fsSL http://localhost:8181/exhibitor/v1/cluster/status |jq .`
Navstar	`dcos-navstar.service`	DC/OS	Mesos master; inet_gethost; erlang port mapping daemon (epmd)		Networking overlay for DC/OS; manages IPv4 Layer 4 load balancing	Loss of Navstar will impact the internal networking of agents/masters and components within the DC/OS solution/cluster.	Ensure that filesystems on the node with Navstar issues is not full. Investigate the logs of navstar: `journalctl -flu dcos-navstar`
CockRoachDB	`dcos-cockroach.service`	DC/OS	-		Distributed SQL database with strongly consistent key-value store.	Losing or issues with cockroachdb will affect IAM (bouncer) and thusly will create cascade events within the cluster. Logging in and authentication are obvious side-effects.	`journalctl -flu dcos-cockroach`

COMPONENTS AND TROUBLESHOOTING BASICS, II

Component	Service Name	Layer	Dependencies	Subcomponent	Description	Impact to Environment	Where to begin	
Bouncer	`dcos-bouncer.service`	DC/OS	mesos.master		IAM access control to components and services within DC/OS. Can also be integrated with external services like LDAP	Authenication and control issues within the environment. Mesos and Marathon may also be integrated with bouncer and when bouncer is lost access to these services will not be possible. Requests and interaction with adminrouter will also be affected.	`journalctl -flu dcos-bouncer`	
Marathon	`dcos-marathon.service`	DC/OS		Service Marathons	Marathon is the main container orchestration component (engine) for DC/OS. It handles management of long running tasks in the cluster. Marathon can also manage child-Marathon tasks in the environment, frequently called Service Marathons.	As a meta-framework in DC/OS, any issue adversly affecting Marathon could potentially affect all tasks (sometimes more than 1000s) running on the cluster. Without Marathon there can be substantial degradation in the cluster and loss of service.	Monitor Marathon from the DC/OS UI: click "Services" tab and select Marathon; inspect logs for Marathon from boot time: `journalctl -u dcos-marathon -b`	
Master	`dcos-master.service`	DC/OS	Zookeeper	cluster agents, tasks, communication, overall cluster	DC/OS distributed systems kernel.	Losing masters in the DC/OS cluster is a critical event and can potentially impact the entire cluster and all tasks running on the cluster. Most production clusters have a minimum of 3 master nodes which can sustain the failure of one node.	View the Mesos Master interface directly at: http://<master-ip-addr>/mesos; gather logs from the start of Mesos Master processes: `journalctl -u mesos-master -b`	
Slave	`dcos-mesos-slave.service`	DC/OS	Tasks running on agents		DC/OS distributed systems kernel agent.	Loss of an agent within DC/OS may cause the agent to become unregistered with the DC/OS masters. It may also negatively affect the tasks running on the agent but existing tasks, executed by an executor should continue running.	Check to make sure the slave process is running on the agent node. Connect to the mesos-agent UI: http://<agent-ip-addr>:5051/slave; investigate logs for the agent service: `journalctl -u dcos-mesos-slave -b`	

COMPONENTS AND TROUBLESHOOTING BASICS, III

Component	Service Name	Layer	Dependencies	Subcomponent	Description	Impact to Environment	Where to begin	
Slave	`dcos-mesos-slave.service`	DC/OS	Tasks running on agents		DC/OS distributed systems kernel agent.	Loss of an agent within DC/OS may cause the agent to become unregistered with the DC/OS masters. It may also negatively affect the tasks running on the agent but existing tasks, executed by an executor should continue running.	Check to make sure the slave process is running on the agent node. Connect to the mesos-agent UI: http://<agent-ip-addr>:5051/slave; investigate logs for the agent service: `journalctl -u dcos-mesos-slave -b`	
Slave Public	`dcos-mesos-slave-public.s`	DC/OS	Tasks running on public agents		DC/OS distributed systems kernel agent on public nodes within a cluster	Loss of an agent within DC/OS may cause the agent to become unregistered with the DC/OS masters. It may also negatively affect the tasks running on the agent but existing tasks, executed by an executor should continue running.	Check to make sure the slave process is running on the agent node. Connect to the mesos-agent UI: `http://<agent-ip-addr>:5051/slave`; investigate logs for the agent service: `journalctl -u dcos-mesos-slave -b`	
Mesos-DNS	`dcos-mesos-dns.service`	DC/OS	mesos.master; /etc/resolv.conf; detect_ip		Mesos-DNS provides domain name based service discovery for tasks running with a DC/OS environment.	Issues with Mesos-DNS will affect service discovery and the ability for apps to resolve names correctly within the cluster. This may manifest itself as service/app outages.	You can verify if name resolution is working within the cluster by performing nslooksups and/or dig @ commands from a master or agent node in the cluster, looking for service names. Additionally, investigate the logs for mesos-dns: `journalctl -u dcos-mesos-dns -b`	
Spartan	`dcos-spartan.service`	DC/OS	mesos-dns		Spartan forwards DNS requests to multiple DNS servers.	Name resolution issues will arise with loss of Spartan.	Investigate the logs for Spartan: `journalctl -u dcos-spartan -b`	

COMPONENTS AND TROUBLESHOOTING BASICS, IV

Component	Service Name	Layer	Dependencies	Subcomponent	Description	Impact to Environment	Where to begin
NTP	`ntpd.service`	Linux	Linux kernel; ntpd; time servers		Keeps time constant on hosts within an environment.	Time drift is a major disruptor or incident causing issue with DC/OS clusters. Time must be synchronized across the cluster in order for components, masters, and slaves to work correctly.	`journalctl -u ntpd.service; ntptime; timedatectl`
Kernel Parameters		Linux	assorted /etc configuration files		Kernel parameters drive how the host operating system runs, responds to needs, and ultimately performs to manage workloads on the system.	Improperly configured Kernel(s) on hosts within a DC/OS cluster can have many negative impacts on running, capacity, and performance.	Systools.

DC/OS Troubleshooting Strategy

APPLICATION DEPLOYMENT

JSON Unit for Application Definition:

```
{
  "id": "/app-server",
  "description": "Appication-v1.",
    "cpus": 0.5,
    "mem": 32,
    "disk": 0,
    "instances": 1,
    "container": {
      "type": "DOCKER",
      "volumes": [],
      "docker": {
        "image": "code/app-server"
      }
  },
}
```

Deploy with: `dcos marathon app add app-server.json`

Elements of the application, which can affect running and stability:

- `“mem”: 32,`
- `“image”: “code/app-server”`
- `“container”: { “type”: “DOCKER”,...}`
- `“healthchecks”: [],`

DC/OS Troubleshooting Strategy

APPLICATION DEBUGGING

- Application logs are the first logical place to look
- Look in the mesos-agent logs for reasons why the app may have stopped
- While running, check sandbox, sandbox logs on the agent where the app is running
- Monitor the app deploy, start, run state from the DC/OS UI
- More…

Topic Four

Diagnostics and Tools for Debugging

TOOLS FOR TROUBLESHOOTING

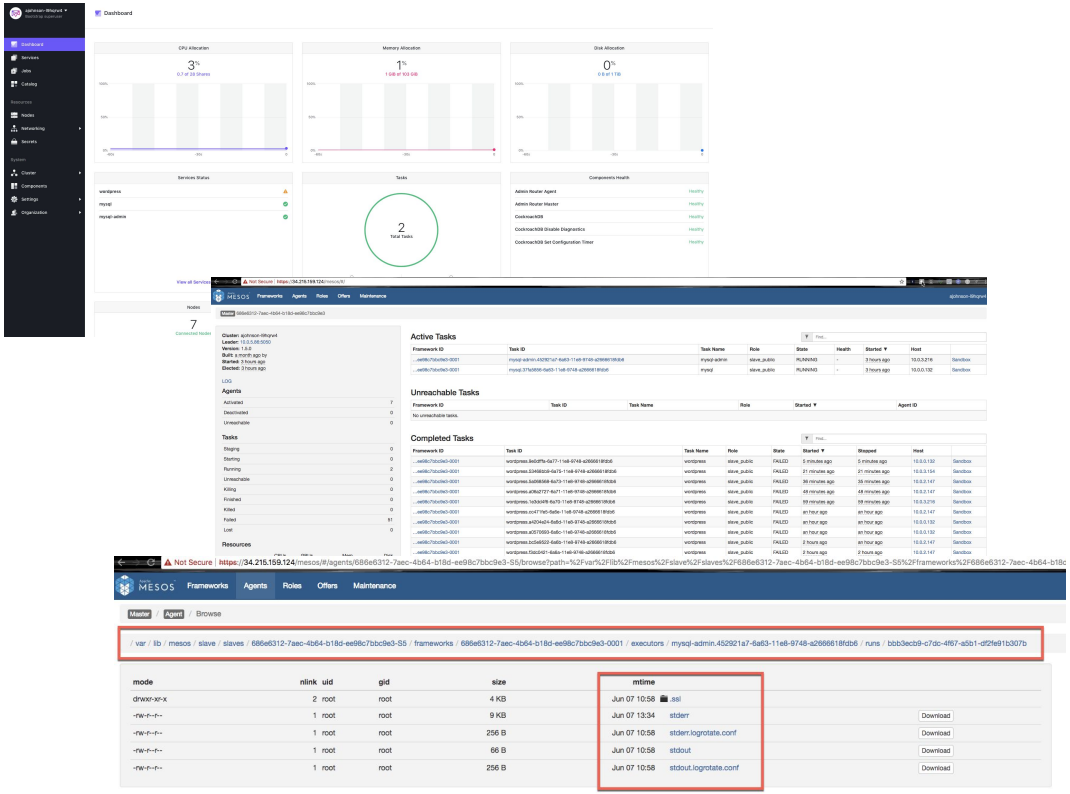
Tools for debugging DC/OS and applications on DC/OS:

- DC/OS UIs and CLI
- Logs - both from DC/OS and host operating systems
- DC/OS Metrics
- HTTP Endpoints
- Community resources
- Interactive debugging of tasks and infrastructure
- Other tools
- DC/OS Debugging Documentation
  - https://docs.mesosphere.com/1.11/monitoring/debugging/
  -

TROUBLESHOOTING: DC/OS UI

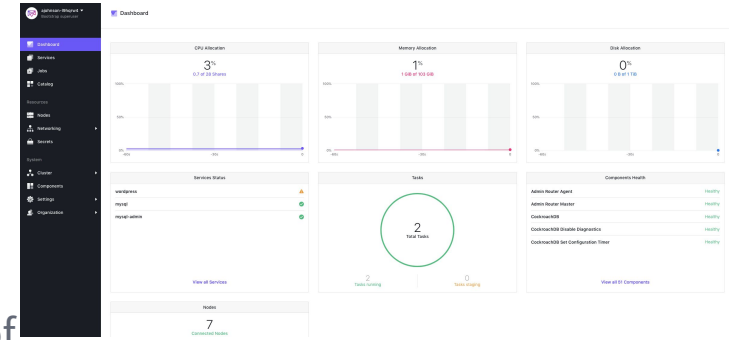
The DC/OS UI is a wise place to begin troubleshooting issues. It allows easy access to data such as:

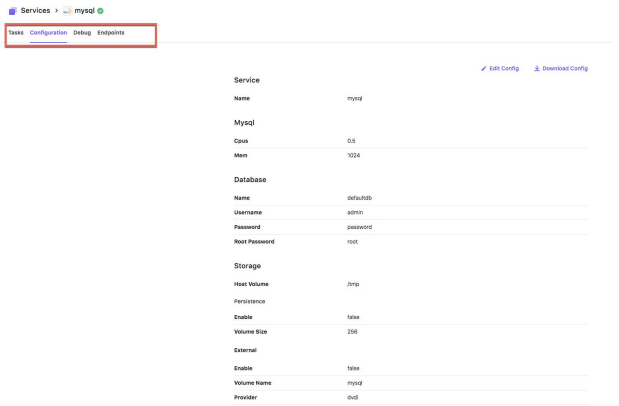
- Resource allocation
  - An overview of available resources within the cluster
- Task logs
  - Provides insight into task failures
- Task debugging information
  - Details from most recent task offers and why a task did start

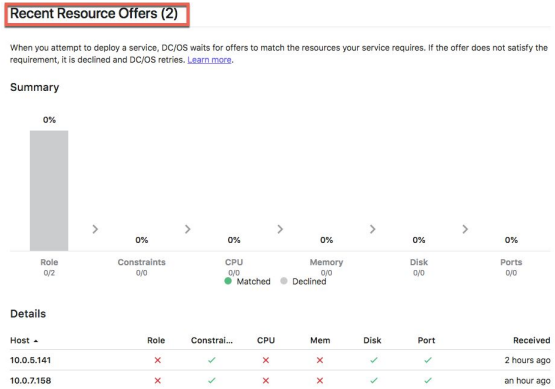


DC/OS UI

- Provides near real-time update on system status
- Provides view of services, configuration, and file data of services running
- Resource offers shown for tasks in deployment, or delayed deployment for debugging purposes

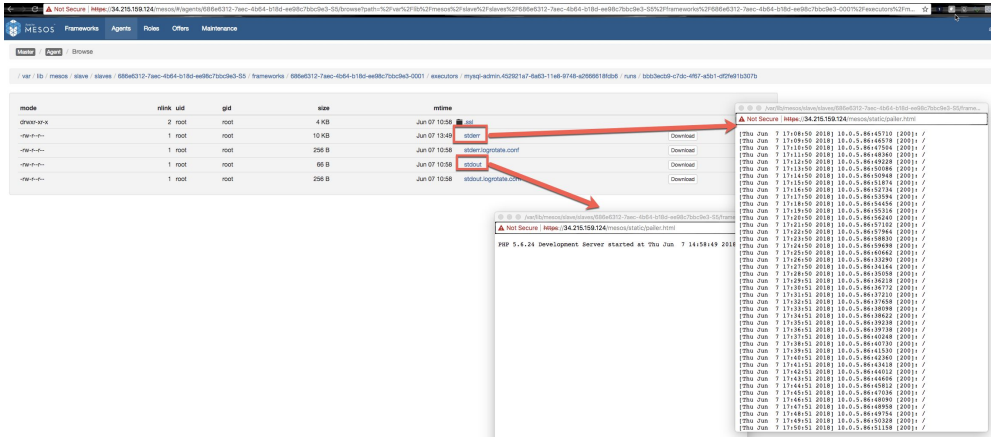






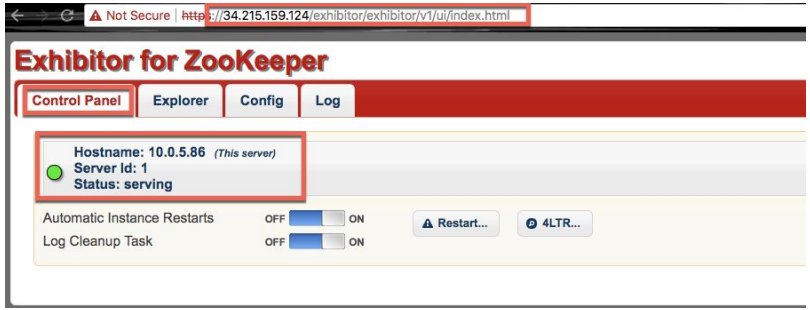
MESOS UI

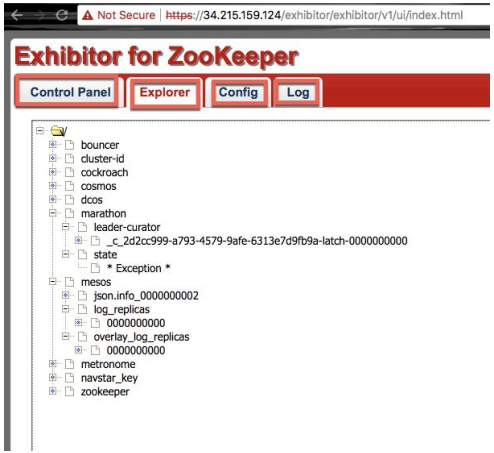
- Mesos UI is available at, http://<masterIPAddr>/mesos
- Shows active, unreachable, and completed tasks within the cluster
- By selecting a <task> you can view the “sandbox” data, including ‘stderr’ and ‘stdout’

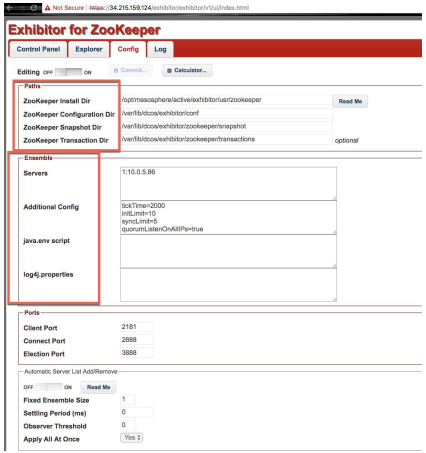


DC/OS Troubleshooting Tools

ZOOKEEPER/EXHIBITOR UI



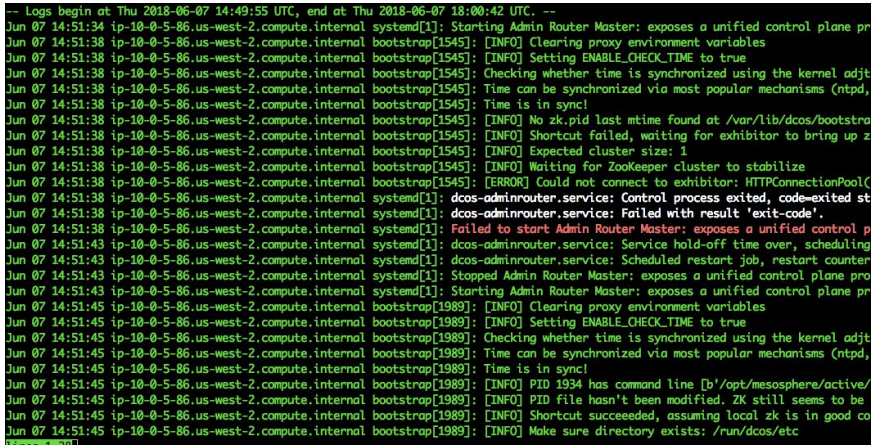




- Exhibitor UI shows status of Zookeeper communication within the cluster
- Identify which are followers and who is the leader in the cluster
- Other options show explorer for data, configuration of exhibitor/zookeeper and log files

TOOLS: LOGS

- View System units and dcos-services with journalctl
- Cmd: sudo journalctl -u dcos-adminrouter -b



Various sources for logs, both application and cluster related:

- Task/Application logs
- Mesos-Agents
- Mesos-Masters
- Service Scheduler(s) (e.g., Marathon)
- System logs
  - Documentation for system logs:
    - https://docs.mesosphere.com/1.11/monitoring/logging/#system-logs

TOOLS: METRICS

There are three main sources for Metrics within DC/OS

1. DC/OS Metrics-> endpoint exposing combined metrics from tasks/containers, nodes, and applications
2. Mesos Metrics endpoint-> Mesos specific metrics
3. Marathon Metrics-> Marathon specific metrics

- DC/OS Metrics:
  - https://github.com/dcos/dcos-metrics
- Mesos Metrics:
  - http://mesos.apache.org/documentation/latest/monitoring/
- Marathon Metrics:
  - https://mesosphere.github.io/marathon/docs/metrics.html
  -

DC/OS Troubleshooting Tools

DCOS DIAGNOSTICS - CLI

1. `dcos node diagnostics`
2. `dcos node diagnostics --list`
    a. `--json`
    b. `--status`

Topic Five

Adventures in Troubleshooting

POTENTIAL ISSUES

Infrastructure:

- Time and date out of sync within the cluster
- Missing prerequisite packages on the underlying operating environment
- Root disk on operating environment full or filling too quickly
- TCP ports blocked between master and agent nodes

Application:

- Inappropriate resource definition (too much or too little, resulting in OOM symptoms)
- Inadequate resources to deploy and start tasks
- Incorrect usage of Catalog packages, e.g. Marathon-LB working with external load balancer issues
- Incorrect JSON definitions resulting in app starting and killing loop

TROUBLESHOOTING: USE CASE I

Troubleshooting Use Case I: DC/OS Networking
Problem Statement: When new deployments are happening for large services (400 instances +) it takes too long for Minuteman to converge the vips.
Data Gathered/Analysis Conducted: <Enter data-analysis here>
Potential Solution: <Enter solution here>

TROUBLESHOOTING: USE CASE II

Troubleshooting Use Case II: Marathon Performance
Problem Statement: There have been issues with services deployed that would contain “500 errors” and containers would crash. Due to marathon backing off from rescheduling, these services would end up without any running containers. What are some possible workarounds and recommendations for backoff tuning?
Data Gathered/Analysis Conducted: <Enter data-analysis here>
Potential Solution: <Enter solution here>

TROUBLESHOOTING: USE CASE III

Troubleshooting Use Case III: Capacity-Performance Analysis
Problem Statement: Cluster is currently X in size: How do you plan capacity properly? What do you need to actively monitor to ensure available resources and stability?
Data Gathered/Analysis Conducted: <Enter data-analysis here>
Potential Solution: <Enter solution here>

TROUBLESHOOTING: USE CASE IV

Troubleshooting Use Case IV: Performance Analysis
Problem Statement: There are currently 32 teams / services on the cluster and the concern is, all teams try to schedule deployments in a short period of time, there might be a bottleneck. Which metrics should be monitored for the scheduler to ensure it's appropriately tuned and can support the workload?
Data Gathered/Analysis Conducted: <Enter data-analysis here>
Potential Solution: <Enter solution here>

MESOSPHERE