MESOSPHERE

DC/OS Metrics & ELK Integration

Overview of DC/OS Metrics and ELK Integration

October, 2018

Arthur Johnson,

Solutions Architect, Mesosphere

ajohnson@mesosphere.io

Getting Started: DCOS-Metrics

Assumption

This brief document deals with DCOS-Metrics, a pre-1.12 metrics/reporting/monitoring mechanism within a DC/OS cluster solution.

Core metrics are derived and delivered by DCOS-metrics and sub-components which provide cluster usage/performance data. Additional packages for scraping data, mapping, and management may be utilized. A Core requirement is integration with ELK, however, Grafana may also be used as a intermediary for monitoring/reporting.

Objective

Provide live-near-term metrics on the health of a DC/OS cluster reporting on usage, availability, and error reporting prior to events causing outages. Report on capacity/utilization of a cluster environment to provide short and long-term perspective on the overall health and capacity of a given cluster.

Recommendation

DC/OS-Metrics is being replaced by Telegraph in DC/OS 1.12 and thus will not be maintained long-term, moving forward. Deploying Groundwork monitoring agents on DC/OS nodes is the best recommendation as this agent will provide the same insight that DCOS-metrics would reveal on a per/node perspective. There are additional metrics which may be useful but can be applied or automated on an as needed basis.

In addition to this, it should be vetted if something like Grafana may provide additional useful perspective for the existing clusters running on DC/OS 1.11.x+. This is ongoing work. Additional effort and development of extra tools should be carefully considered as the monitoring API for DC/OS will change with the next GA release (1.12.x)

Logging and Metrics

- DC/OS Network Metrics: exposes networking related metrics
    - `Dcos-networking_api.service`
- DC/OS Diagnostics (3DT): also known as the “Distributed Diagnostics Tool”. Aggregates and presents component health
    - `Dcos-3dt.service; dcos-3dt.socket`
- DC/OS Log: exposes node, component, and container logs
    - `Dcos-log-master.service`
    - `Dcos-log-agent.service`
- DC/OS Metrics: exposes node, container, and application metrics
    - `Dcos-metrics-master.service`
    - `Dcos-metrics-agent.service`
- DC/OS History: a historical cache of system state to present usage statistics in the UI
    - `Dcos-history.service`

DC/OS Metrics

Available on all hosts, both masters and agents. Available on each host at the following endpoint: /system/v1/metrics/v0

Metrics are appended with structured data in the format of:

- Agent-id
- Framework-id
- Task-id

Applications will discover endpoints via an environment variable `$STATSD_UDP_HOST or $STATSD_UDP_PORT`

Three types of metrics:

- Host (node specific information)
- Container (cgroup allocation from tasks running in either UCR or Docker containers)
- Application (metrics about apps running in UCR)

Host (Node) Metrics:

Metric	Purpose/Description
cpu.cores	% of cores used
cpu.idle	% of idle CPUs
cpu.system	% of system used
cpu.total	% of CPUs used
cpu.user	% of CPUs used by the user space
cpu.wait	% of idle while waiting for an operation to complete
Load.1min; load.5min; load.15min	Load averages for 1, 5, 15 minutes
memory.buffers	# of memory buffers
Memory cached	Amount of cached memory
memory.free	Amount of free memory (in bytes)
memory.total	Total amount of memory (bytes)
process.count	# of processes running
Swap.free; swap.total; swap.used	Total amount of swap, free swap, used swap
system.uptime	System uptime and load average

Filesystem & Network Metrics

Metric	Purpose/Description
filesystem.capacity.(free|total|used)	Total capacity, available capacity, used capacity
filesystem.inode(fr	Inode reporting

ee|total|used)	
Network	
network.in.(bytes|dropped|errors|packets)	# of bytes downloaded, dropped, errors, packets
network.out.(bypte s|dropped|errors|packets)	# of bytes uploaded, dropped, errors, packets

Container Metrics

Metric	Purpose/Description
CPU Usage	
cpus.limit	# of CPU shares allocated
cpus.system.time	Total of CPU time spent in kernel mode in seconds
cpus.throttled.time	Total time, in seconds, that CPU was throttled
cpus.user.time	Total CPU time spent in user space/mode
Disk Metrics	
disk.limit|used	Capacity limit and used in bytes
Memor Metrics	
mem.limit|total	Memory limit for a container (task), memory of a process in RAM
Network Metrics	
net.rx.bytes|dropped|errors|packets	Received for bytes, dropped, errors, packets
net.tx.bytes|dropped|errors|packets	Sent bytes, dropped, errors,
Mesos Dimensions	

mesos_id	Id of a node
cluster_id	Id of a mesos cluster
container_id	Id of a container
executor_id	Id of a task executor
executor_name	Name of the executor
framework_id	…
framework_name	….
framework_principal	Principal of the framework
framework_role	...
hostname	Actually the IP address of the node
labels	Key-value pairs describing the metric
task_id|name	ID and name of the task

The statistics collector collects DC/OS statistics for the following services:

1. Mesos Master statistics
2. Mesos Slave (Agent) statistics
3. Marathon Master Statistics
4. Zookeeper Statistics

Key Metrics to Trap

Mesos Master Resources

Metric	Descripton
master/mem_percent	% of allocated memory
master/mem_total	Memory in MB
master/cpu_percent	% of allocated CPUs
master/cpus_total	Total number of CPUs
master/uptime	Uptime in hours
master/elected	Which master is identified as the elected leader

Mesos Slaves

Metric	Description
slaves/disconnected	# of slaves disconnected
slaves/active	# of active slaves
slaves/connected	# of of connected slaves
slaves/removals	# of slaves removed, for maintenance or other reasons

Mesos Master Task(s)

Metric	Description
task_killed	# of killed tasks
task_lost	# of lost tasks

master/messages_kill_task	# of kill task messages
master/message_decline_offers	# of offers declined

Mesos Agent Task(s)

Metric	Description
slave/mem_percent	% of allocated memory
slave/mem_total	Total memory in MB
slave/cpus_percent	% of allocated CPUs
slave/cpus_total	Total number of CPUs
slave/uptime_hours	…
slave/registered	Identify that the slave is registered with the master(s)
slave/task_killed	# of killed tasks
slave/tasks_lost	# of tasks lost per slave

Marathon Metrics

Metric	Description
IncomingOffers	# of incoming offers for tasks
ResourceOffers	# of resource offers
SlaveLost	# of Marathon slaves lost
statusUpdate	Marathon status updates
KillTasks	# of killed tasks
NumTasks	# of tasks
uptime	Marathon uptime

MarathonCPUUsage	Consumption of CPU by Marathon
MarathonMemUsage	Consumption of Mem by Marathon

Zookeeper Metrics

Metric	Description
NumConnections	# of active ZK connections
IsServerOK	ZK server state, currently a leader or follower in the cluster
NodeCount	# of total data registers in ZK (how many are in the cluster)
AvgLatency	ZK average latency measured by ZK (set threshold)
ZKCpuUsage	Consumption of ZK CPU
ZKMemUsage	Consumption of ZK Mem (per znode)

Source documentation: https://github.com/soabase/exhibitor/wiki/REST-Introduction

GET cluster status:
curl http://leader.mesos:8181/exhibitor/v1/cluster/status |jq .

GET cluster state:
`curl http://leader.mesos:8181/exhibitor/v1/cluster/state |jq .`

GET cluster state / machine:
Usage: `curl http://leader.mesos:8181/exhibitor/v1/cluster/state/{hostname} |jq .`
`curl http://leader.mesos:8181/exhibitor/v1/cluster/state/172.31.28.211 |jq .`

GET log / machine:
Usage: `curl http://leader.mesos:8181/exhibitor/v1/cluster/log/{hostname} |jq .`
`curl http://leader.mesos:8181/exhibitor/v1/cluster/log/172.31.28.211 |jq .`

GET stop/start Zookeeper:
Usage: `curl http://leader.mesos:8181/exhibitor/v1/cluster/restart`

GET list of Zookeeper servers:
`curl http://leader.mesos:8181/exhibitor/v1/cluster/list |jq .`

GET systemState
`curl http://leader.mesos:8181/exhibitor/v1/config/get-state |jq .`

Accessing Metrics via Curl on Clusters

Metrics Producers HTTP API Producer Specification

Base Structure

`http://<hostname>:<port>/system/v<version>/metrics/v<version>/<request>`

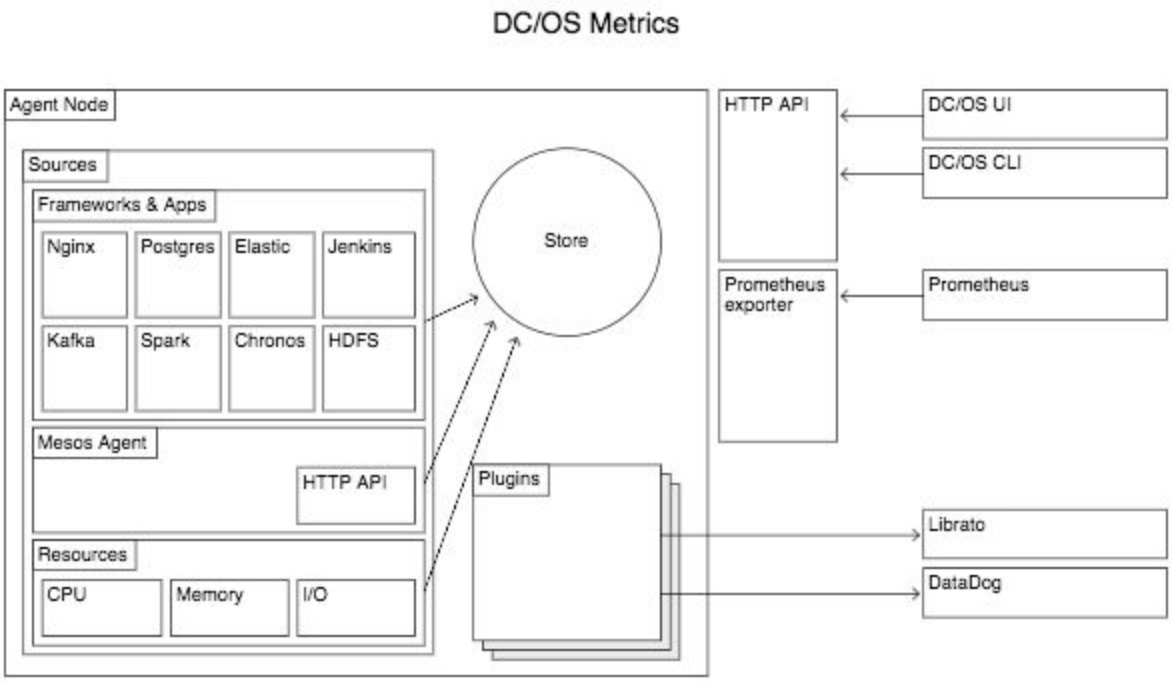
Node Endpoints

- `/system/metrics/v0/node`

Containers and App Endpoints

- `/system/metrics/v0/containers`
- `/system/metrics/v0/containers/<id>`
- `/system/metrics/v0/containers/<id>/app`
- `/system/metrics/v0/containers/<id>/app/<metric-id>`

Accessing Metrics - DCOS-metrics



Statistics for current leader: `/system/v1/metrics/v0`

Statistics for agent(s):
`/system/v1/agent/{agent_id}/metrics/v0`

All metrics via API require authentication and by use of a token. The token can be retrieved by:

```
curl -kv POST http://leader.mesos/acs/api/v1/auth/login -d '{"uid":
"bootstrapuser", "password": "deleteme"}' -H 'Content-Type: application/json'
```

Then

```
export token=<token-id>
```

Primary Routes for API Calls:
Masters: `/system/v1/metrics/v0`
Agents: `/system/v1/agent/{agent_id}/metrics/v0`

Cluster Version (Mesos):

```
curl -H "Authorization: token=$token" http://leader.mesos/mesos/version |jq .
```

Ping the cluster master:

```
curl -H "Authorization: token=$token"
http://leader.mesos/system/v1/metrics/v0/ping  |jq .
```

To get node metrics:

```
curl -H "Authorization: token=$token"
http://leader.mesos/system/v1/metrics/v0/node |jq
```

Get node metrics on another node, e.g. Agent:

```
curl -H "Authorization: token=$token"
http://leader.mesos/system/v1/agent/aa8bf8b3-bbcb-4e2c-9676-55b1d04703ac-S4/me
trics/v0/node |jq
```

Viewing running containers on a node:

```
curl -H "Authorization: token=$token"
http://leader.mesos/system/v1/agent/aa8bf8b3-bbcb-4e2c-9676-55b1d04703ac-S4/me
trics/v0/containers |jq
```

DC/OS metrics listens for statsd metrics

The metrics collectors run on every node in the cluster and provide the following methods of ingesting metrics:

- Periodically polling the Mesos HTTP APIs for cluster state and metrics about running containers and cluster services
- Periodically polling the node's operating system for system-level resource utilization (CPU, memory, disk, networks)
- Listening on TCP port 8124 for metrics being sent from the Mesos metrics module

The metrics collectors then transform their metrics to fit a common message format, at which point they are broadcast to one or more configured *producers*. More information about the available producers and their implementation can be found in PRODUCERS.md.

A *collector* refers to code that queries the underlying operating system, Mesos, and/or DC/OS to collect metrics. Each collector then transforms collected metrics into a `MetricsMessage`, or common format that can be broadcast to one or more producers.

Included Collectors

As part of the dcos-metrics project, we've included several metrics collectors to help you get metrics from various sources on the agent into a common format, and eventually off the node and into the metrics store of your choice.

Framework

Listens on TCP port 8124 for metrics being sent from the Mesos metrics module. Allows applications running in containers to ship metrics (in DogStatsD format) to the listener by using the environment variables `STATSD_UDP_HOST` and `STATSD_UDP_PORT`.

Sample API ROUTES

GET system health:

```
curl -X GET -H "Authorization: token=$token"
http://leader.mesos/system/health/v1 |jq .
```

System Health Report:

```
curl -X GET -H "Authorization: token=$token"
http:/localhost/system/health/v1/report |jq .
```

→ System health by nodes:

```
curl -X GET -H "Authorization: token=$token"
http:/localhost/system/health/v1/nodes |jq .
```

→ System health by units (DC/OS Components)

```
curl -X GET -H "Authorization: token=$token"
http:/localhost/system/health/v1/units |jq .
-0- is a healthy state
```

→ Version of Mesos-dns:

```
curl -X GET -H "Authorization: token=$token"
http://localhost/mesos_dns/v1/version
```

→ Configuration of Mesos-dns:

```
curl -X GET -H "Authorization: token=$token"
http://localhost/mesos_dns/v1/config |jq .
```

→ Hosts in the Mesos domain

```
curl -X GET -H "Authorization: token=$token"
http://localhost/mesos_dns/v1/hosts/nginx.marathon.mesos
```

→ Resolve names in the marathon.mesos domain:

```
curl -X GET -H "Authorization: token=$token"
http://localhost/mesos_dns/v1/services/_nginx._tcp.marathon.mesos
```

→

Mesos (master):
→ State
```
curl -X GET http://master.mesos:5050/state |jq .
```

→ Definition of roles in the cluster
```
curl -X GET http://master.mesos:5050/roles |jq .
```

→ Mesos - Slaves
```
curl -X GET http://master.mesos:5050/slaves |jq .
```

→ State-summary:
```
curl -X GET http://master.mesos:5050/state-summary |jq .
```

→ Tasks
```
curl -X GET http://master.mesos:5050/tasks |jq .
```

→ Metrics Snapshot
```
curl -X GET http://master.mesos:5050/metrics/snapshot |jq .
```

→ System stats as seen from Mesos
```
curl -X GET http://master.mesos:5050/system/stats.json |jq .
```

→ Mesos Version
```
curl -X GET http://master.mesos:5050/version |jq .
```

Mesos (agent):
→ Metrics Snapshot (per agent)
```
curl -X GET http://172.31.28.82:5051/metrics/snapshot |jq .
```
Where IPADDR is <private-IP-of-agent>

→ Mesos Agent Flags
```
curl -X GET http://172.31.28.82:5051/flags |jq .
```

→ Monitor Statistics
```
curl -X GET http://172.31.28.82:5051/monitor/statistics |jq .
```

→ Agent State

```
curl -X GET http://172.31.28.82:5051/state |jq .
```

→ System Statistics (per agent):

```
curl -X GET http://172.31.28.82:5051/system/stats.json |jq .
```

→ Mesos Agent Version

```
curl -X GET http://172.31.28.82:5051/version |jq .
```

ELK Integration with DC/OS

Application Logging, Monitoring, Reporting and Telemetry

Description:

Ability of the DC/OS platform to provide system and application logs, reports and telemetry for operators and users

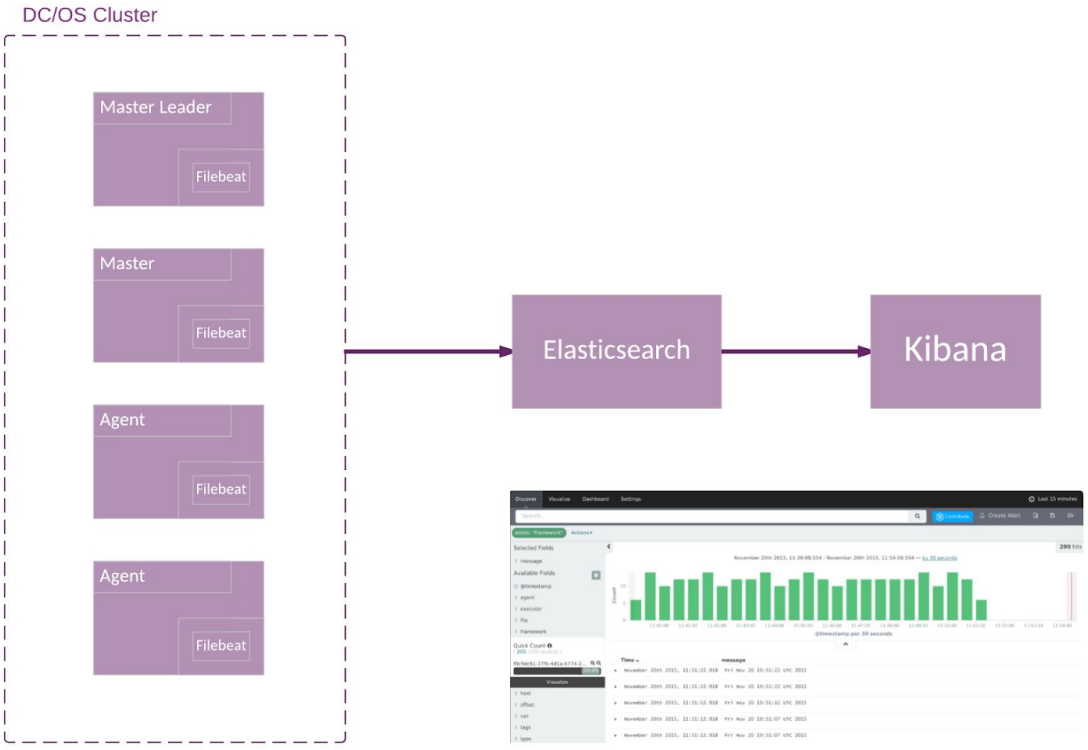
Solution:

DC/OC has 3 main logs: Mesos logs, Service logs, and Task logs. Mesos logs are system logs with information on health of nodes (Master and Agents). You can access system logs by using either one of the following commands: **dcos node log** or **journalctl**. You can access logs without ssh into private agent nodes by dcos node log method. Service logs contain information on components such as Admin Router, cosmos, Marathon, Mesos DNS, and Metronome. Use command **dcos service log** to access these information. Task logs are generated by Mesos task runs in its own "sandbox", or a directory on the host machine. Files generated by a task appear in the sandbox, and Mesos itself creates stdout and stderr files that capture these streams from the task. You can access these logs with command **dcos task log**.

Monitoring is needed to add intelligence to logs information and generate alerts. Marathon exposes metrics via REST API. DC/OS UI provide health check status (System -> Components). Logs can also be extracted to log repositories and analytics engines. For log management and reporting, Splunk or ELK integration are available out of the box. Mesosphere does not provide plugins for Splunk or ELK, but these are standard industry log management solutions.

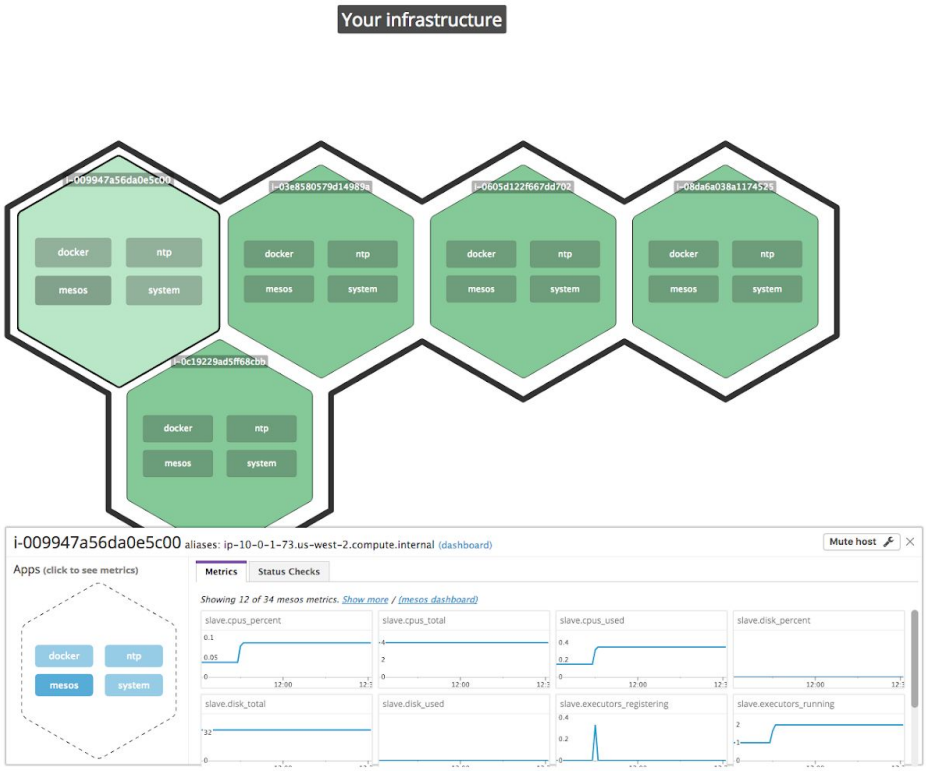
Status of Jobs running in DC/OS (via Metronome) and additional debugging information about waiting and failed Jobs can be retrieved from Splunk or ELK after they have been configured to receive journald logs from the DC/OS masters and agents.

Log management with ELK



Performance and System Utilization Monitoring

System utilization and performance metrics for the DC/OS cluster can be obtained using third-party monitoring tools such as Datadog or Sysdig. Datadog or Sysdig are available as Universe packages that can be deployed on the agent nodes in the cluster. Agents can be deployed manually to the master nodes to gather insight into master metrics.



For application metrics and performance reporting an open source leveraging cAdvisor (container performance monitoring), InfluxDB (data persistence) and Grafana (visualization) can be used as a potential solution.

Appendix I: Monitoring; Telemetry for Chargeback/Showback

Mesos master and agent nodes expose metrics through the API endpoint as JSON objects. The list of metrics that can be queried through endpoints for masters and nodes are listed here: http://mesos.apache.org/documentation/latest/monitoring/

Procedure:

https://docs.mesosphere.com/1.12/monitoring/logging/aggregating/elk/
https://docs.mesosphere.com/1.12/monitoring/logging/aggregating/filter-elk/

References and Additional Reading:

https://mesosphere.com/blog/2015/07/22/new-dcos-cli-commands-for-logs-and-ssh/
https://docs.mesosphere.com/1.11/overview/telemetry/
https://mesosphere.com/blog/2017/01/26/monitoring-dcos-cadvisor-influxdb-grafana/

Appendix II - Details - Data Sources

- Metrics reference for 1.9:

https://docs.mesosphere.com/1.9/metrics/reference/

For more information, see the dcos-metrics repository.

Node

Container

Dimensions

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

- Performance Monitoring:

https://docs.mesosphere.com/1.9/monitoring/performance-monitoring/