



DC/OS Demo

Webex, April 19th & 24th, 2018

Arthur Johnson,
Solutions Architect, Mesosphere
ajohnson@mesosphere.io

Note: most of these demos will run on a small nominal cluster. The last demo, Arangodb, requires at least 3 private agents and 2 public agents.

Demo I: Install CLI; Discover UI; Launch simple Hello-World

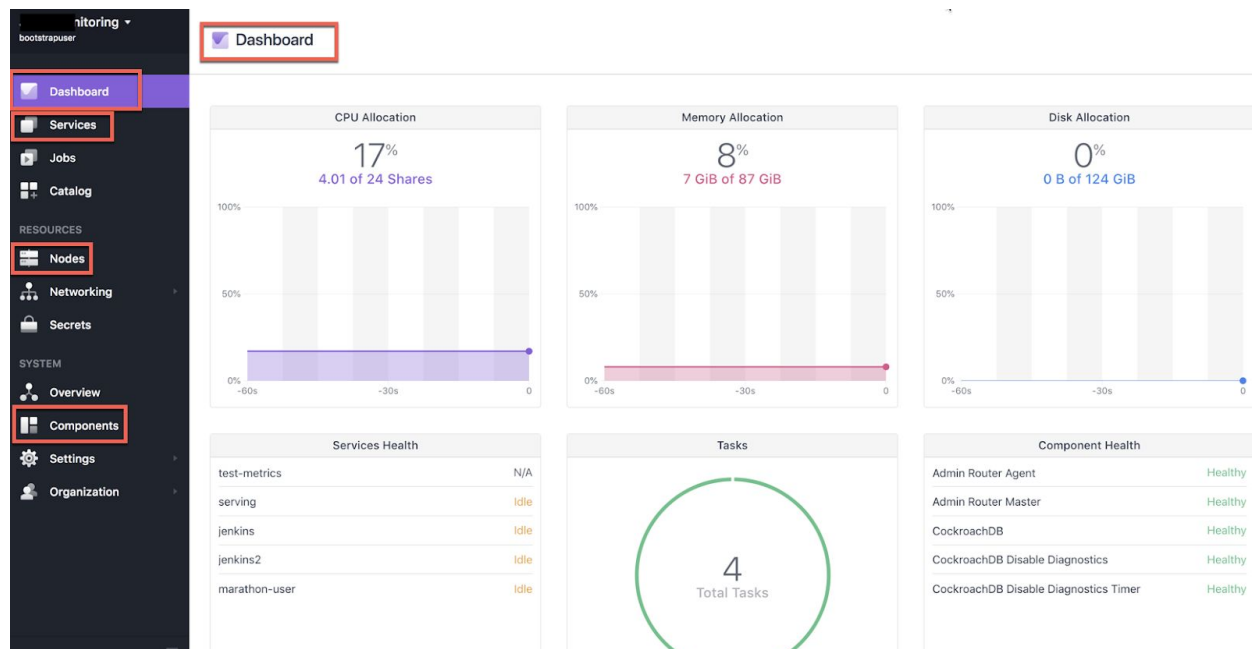
1. To install the DC/OS CLI -- you must first have an installed cluster and know what the UI address (master IP ADDR) is and the credentials to connect to the environment. The default credentials are username: bootstrap user; password: delete-me

Install the CLI

```
curl https://downloads.dcos.io/binaries/cli/darwin/x86-64/dcos-1.11/dcos -o dcos &&
sudo mv dcos /usr/local/bin &&
sudo chmod +x /usr/local/bin/dcos &&
dcos config set core.dcos_url https://<master-ip-address> &&
dcos auth login &&
dcos
```

2. Login and Discover the DC/OS UI-- Investigate the following:
 - a. Dashboard
 - b. Nodes
 - c. Components
 - d. Metrics for currently configured environment

DC/OS UI



3. Create a simple Hello-world app in JSON and run it on the cluster via CLI

- a. Create a file with the Hello-World JSON- this will be created where the DCOS-CLI was installed (e.g. Master server). E.g. "vi hello-world.json"

Hello-World JSON

```
{  
  "id": "/hello-world",  
  "cmd": "while [ true ]; do echo -n 'Hello Marathon: ' ; date; sleep 5; done",  
  "cpus": 0.25,  
  "mem": 10.0,  
  "instances": 1  
}
```

- b. Run your Hello-World app:

```
dcos marathon app add ./hello-world.json
```

- c. List the App from the CLI

```
dcos marathon app list
```

- d. Verify you can find your app in the UI- under "Services". It should also appear as a running task from the Dashboard

Demo II: Launch a Docker Container to DC/OS

1. In this demo we are going to run a Docker container on DC/OS. For ease of use, we'll repurpose the JSON below and also repurpose a content file called "master.zip". Both of these are available from Mesosphere's Github. We have the following details on the container and content:
 - a. Container will be nginx
 - b. Content is pulled from Mesosphere's Github

2. On the master server, create a JSON definition (file) for the nginx container

Nginx.json

```
{  
  "nginx": {  
    "cpus": 1,  
    "mem": 1024,  
    "bridge": true,  
  
    "contentUrl": "https://github.com/mesosphere/hello-nginx/archive/master.zip",  
    "contentDir": "hello-nginx-master/"  
  }  
}
```

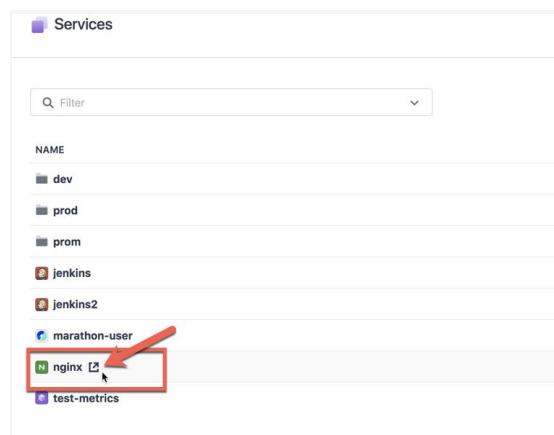
3. Launch your nginx based Container

```
dcos package install nginx --options=nginx.json
```

4. Check that the app was launched from both the CLI and the UI:

```
dcos marathon app list
```

View the App in the UI -- go to: Services->Find your nginx→ and click the "pop-out" option to see the container serve a page:



Demo III: Check out the Mesos UI

Take some time to discover and explore the UI for Mesos. The UI for Mesos will be at the following service endpoint:

`http://<master-ip-Address>/mesos`

The screenshot shows the Mesos UI interface. The top navigation bar includes 'Frameworks', 'Agents', 'Roles', 'Offers', and 'Maintenance'. The left sidebar contains 'Cluster' information, 'LOG', 'Agents' (highlighted), 'Tasks' (highlighted), and 'Resources' (highlighted). The main content area displays 'Active Tasks' and 'Unreachable Tasks' tables. The 'Active Tasks' table has columns: Framework ID, Task ID, Task Name, Role, State, Started, Host, and a 'Sandbox' column (highlighted). The 'Unreachable Tasks' table is currently empty. The 'Completed Tasks' table is also visible at the bottom.

Framework ID	Task ID	Task Name	Role	State	Started	Host	Sandbox
4b4277dc-21ea-41a8-8a1a-2fb741d24ec6-0000	nginx.bc5f61c0-4677-11e8-b2f3-5299e0d4b7c9	nginx	slave_public	RUNNING	24 minutes ago	172.31.1.12	Sandbox
4b4277dc-21ea-41a8-8a1a-2fb741d24ec6-0000	dev_spark_history.9e0ab75f-44e2-11e8-b2f3-5299e0d4b7c9	history.spark.dev	slave_public	RUNNING	2 days ago	172.31.0.100	Sandbox
4b4277dc-21ea-41a8-8a1a-2fb741d24ec6-0000	dev_spark_dispatcher.8ba4c92d-44e2-11e8-b2f3-5299e0d4b7c9	dispatcher.spark.dev	slave_public	RUNNING	2 days ago	172.31.0.100	Sandbox
4b4277dc-21ea-41a8-8a1a-2fb741d24ec6-0000	prom_statsd-exporter.48b49ca8-2544-11e8-8500-52ae8b1816ae	statsd-exporter.prom	slave_public	RUNNING	a month ago	172.31.5.99	Sandbox
4b4277dc-21ea-41a8-8a1a-2fb741d24ec6-0000	test-metrics.d896cff-2215-11e8-8d02-5299e0d4b7c9	test-metrics	slave_public	RUNNING	2 months ago	172.31.1.12	Sandbox

Framework ID	Task ID	Task Name	Role	Started	Agent ID
No unreachable tasks.					

Framework ID	Task ID	Task Name	Role	State	Started	Stopped	Host	Sandbox
4b4277dc-21ea-41a8-8a1a-2fb741d24ec6-0000	dev_tensorflow_serving.41b7afdd-3d38-11e8-b2f3-5299e0d4b7c9	serving.tensorflow.dev	slave_public	FAILED		2 weeks ago	172.31.6.214	Sandbox

Demo IV: Implement health checks for a Tomcat container

As in the previous applications, launched via dcos-cli by add an app JSON, create an instance of Tomcat by using the below tomcat JSON:

```
{
  "id": "/tomcat",
  "instances": 1,
  "cpus": 1,
  "mem": 512,
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "tomcat:8.5",
      "network": "BRIDGE",
      "portMappings": [
        { "protocol": "tcp", "hostPort": 80, "containerPort": 8080 }
      ]
    }
  }
}
```

```

    }
  },
  "requirePorts": true,
  "acceptedResourceRoles": [
    "slave_public"
  ],
  "env": {
    "JAVA_OPTS": "-Xms256m -Xmx256m"
  },
  "healthChecks": [
    {
      "gracePeriodSeconds": 120,
      "intervalSeconds": 30,
      "maxConsecutiveFailures": 3,
      "path": "/",
      "portIndex": 0,
      "protocol": "HTTP",
      "timeoutSeconds": 5
    }
  ]
}

```

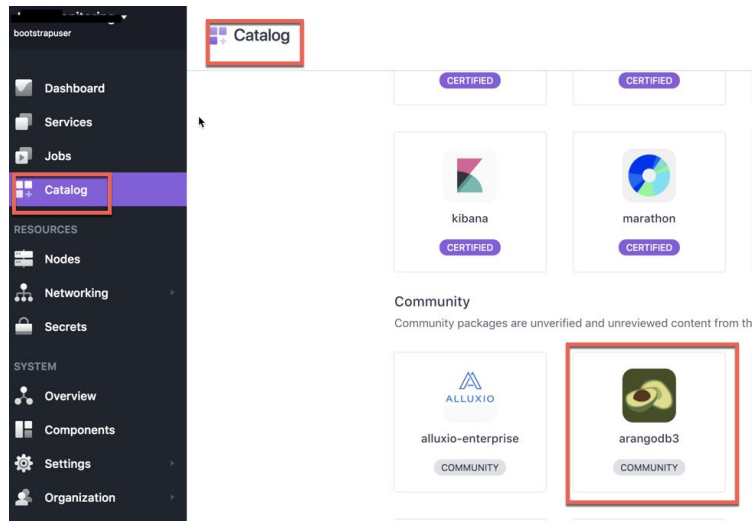
The last segment of the JSON is for “healthchecks”. The defined properties are considered to be default and can be tweaked to meet the expectations for an application environment.

After the application is running, attempt to get it to fail health checks by demanding more from the environment than is physically possible -- one way to do this is to ask DC/OS to run multiple instances of this application, e.g. 10. You can do this by ‘scaling the application’. Since the cluster does not have enough resources it cannot fulfil the deployment request and will mark the application as ‘unhealthy’. Find in the system where you can see it listed as ‘unhealthy’.

Demo V: Install a package from the Catalog: Arangodb

Install ArangoDB3 from the Universe/Catalog. Arangodb is a community offering that is available within the DC/OS catalog starting in version 1.9 of DC/OS. This demo does not go beyond installing and bringing the framework online. It does include verifying that the application did start and can be reached by the UI.

1. From the DC/OS UI, install the arangodb package from the Catalog:



2. Monitor from the DC/OS UI installation progress. This can take a few minutes or more.
3. Once the service is installed you should be able to open the interface to ArangoDB and browse it's capabilities from within the DC/OS UI
4. Connect to ArangoDB from within DC/OS and deploy the ArangoDB Mesos HAProxy. This can be done on the master server where you have DC/OS installed:
 - a. Connect to the master
5. Clone the arangodb-mesos-haproxy repo

git clone <https://github.com/arangodb/arangodb-mesos-haproxy>

6. Add the arangodb-mesos-haproxy to the cluster
dcos marathon app add marathon.json

7. Review the marathon.json that is part of the Proxy

Marathon.json

```
{
  "id": "/arangodb-proxy",
  "cmd": "nodejs /configurator.js arangodb3",
  "cpus": 1,
  "mem": 128,
```

```
"disk": 0,
"instances": 3,
"container": {
  "type": "DOCKER",
  "volumes": [],
  "docker": {
    "image": "arangodb/arangodb-mesos-haproxy",
    "network": "BRIDGE",
    "portMappings": [
      {
        "containerPort": 80,
        "hostPort": 8529,
        "protocol": "tcp",
        "labels": {}
      }
    ],
    "privileged": false,
    "parameters": [],
    "forcePullImage": true
  }
},
"requirePorts": true
}
```