DCOS CLI Guide

Introduction

The DCOS Command Line Interface (CLI) is a utility for managing your DCOS installation. Early access features include:

1. The ability to search for, install and uninstall datacenter services from the DCOS package registry.
2. The ability to administer the DCOS init process, Marathon.
3. The ability to install and uninstall subcommands that extend the CLI by adding functionality.

The following instructions illustrate how to use the DCOS CLI for early access users.

Getting Started

The DCOS CLI (or `dcos`) ships with five basic commands:

```
Command line utility for the Mesosphere Datacenter Operating System (DCOS). The Mesosphere
DCOS is a distributed operating system built around Apache Mesos. This utility provides tools
for easy management of a DCOS installation.

Available DCOS commands in '/Users/me/workspace/dcos-cli/cli/env':

       config               Get and set DCOS command line options
       help                 Display command line usage information
       marathon             Deploy and manage applications on the DCOS
       package              Install and manage DCOS software packages
       subcommand           Install and manage DCOS CLI Subcommands

Get detailed command description with 'dcos <command> --help'.
```

In this document we'll walk through some common actions you might perform. First, let's go over some useful high-level information.

Getting Help

`dcos help` shows a list of all available subcommands. Running `dcos <subcommand> --help` shows detailed help for that subcommand, e.g. `dcos package --help`.

Configuration

A default configuration file is created during setup in `~/.dcos/dcos.toml`. If you'd like to override the configuration file for your session, simply set the environment variable `DCOS_CONFIG` to point to the new configuration file. In addition, the CLI comes with a subcommand called `dcos config` for reporting and modifying the values in your local configuration file.

Recipes

Installing a Datacenter Service

Datacenter services are popular distributed applications that can be easily installed on your DCOS cluster. We'll walk through the example of installing Apache Spark, an in-memory engine for data processing.

First, let's update our local cache of the package repository to make sure we have the latest available versions of packages:

```
$ dcos package update
Updating source [https://github.com/mesosphere/universe/archive/ea.zip]
Validating package definitions...
OK
```

If we aren't sure what the exact name of the package is, we can use `dcos package search` to search for it (alternatively `dcos package describe spark` would show us the same information):

```
$ dcos package search "big data"
[
  {
    "packages": [
      {
        "currentVersion": "1.3.0-SNAPSHOT",
        "description": "Spark is a fast and general cluster computing system for Big
Data",
        "framework": false,
        "name": "spark",
        "tags": [
          "mesosphere",
          "framework",
          "bigdata"
        ],
```

```
        "versions": [
          "1.3.0-SNAPSHOT"
        ]
      }
    ],
    "source": "https://github.com/mesosphere/universe/archive/ea.zip"
  }
]
```

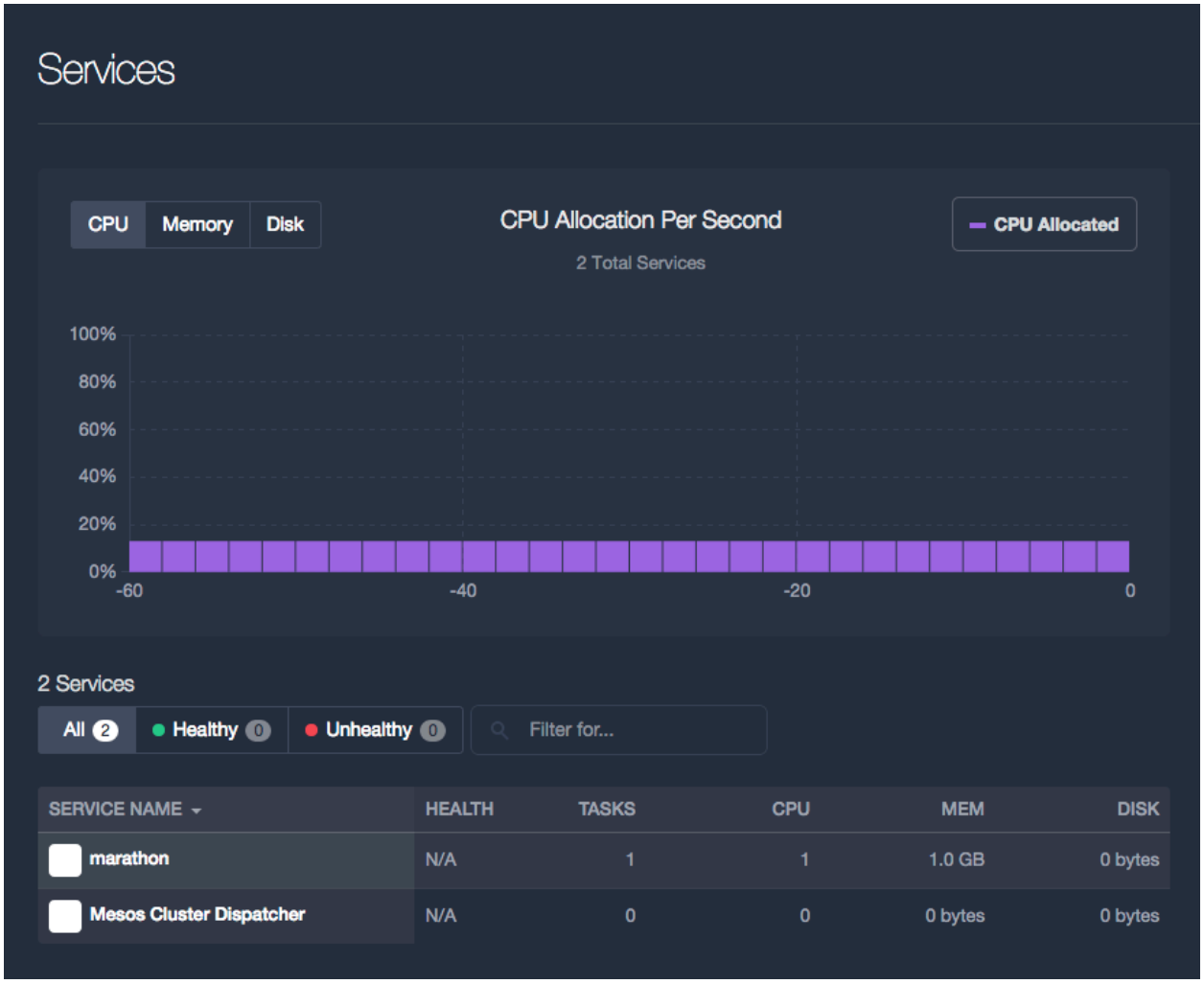
This looks good, so let's install it! Packages come with sensible defaults so it's just one command to install:

```
$ dcos package install spark
Installing package [spark] version [1.3.0-SNAPSHOT]
Installing CLI subcommand for package [spark]
```

We can double check that this has been installed with `dcos package list-installed`:

```
$ dcos package list-installed
[
  {
    "appId": "/spark",
    "description": "Spark is a fast and general cluster computing system for Big
Data",
    "maintainer": "support@mesosphere.io",
    "name": "spark",
    "packageSource": "https://github.com/mesosphere/universe/archive/ea.zip",
    "registryVersion": "0.1.0-alpha",
    "scm": "https://github.com/apache/spark.git",
    "tags": [
      "mesosphere",
      "framework",
      "bigdata"
    ],
    "version": "1.3.0-SNAPSHOT"
  }
]
```

Now, let's hop over to the DCOS UI, where we can see a new service has been registered:



Once we're done playing around with big data, it's as easy to uninstall:

```
$ dcos package uninstall spark
```

There are no errors, so the uninstall worked without a problem. We can verify by checking what's installed again (in this case, nothing):

```
$ dcos package list-installed
[]
```

Deploying a Custom Application

One of the most useful aspects of letting DCOS run your datacenter is that you have just one cluster for both your own applications and popular open source systems like Spark.

Using the `dcos marathon` subcommand, we can easily deploy custom applications to a DCOS cluster. This subcommand works with the same JSON application definitions that the Marathon REST API uses.

In this example we'll use a minimal example application that uses Docker to run three instances of a Python webserver. This is our application JSON. It specifies the container image to use, the command to run and the resources required by the application:

```
$ cat definition.json
{
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "superguenter/demo-app"
    }
  },
  "cmd":  "python -m SimpleHTTPServer $PORT",
  "id": "demo",
  "cpus": 0.01,
  "mem": 256,
  "ports": [3000]
}
```

More examples of application definitions are available in the Marathon docs.

To add this application:

```
$ dcos marathon app add definition.json
```

If this is added successfully, there will be no output. `dcos marathon app list` shows this new application:

```
$ dcos marathon app list
[
  {
    "args": null,
    "backoffFactor": 1.15,
    "backoffSeconds": 1,
    "cmd": "python -m SimpleHTTPServer $PORT",
    "constraints": [],
    "container": {
      "docker": {
        "image": "superguenter/demo-app",
        "parameters": [],
```

```
        "privileged": false
      },
      "type": "DOCKER",
      "volumes": []
    },
    "cpus": 0.01,
    "dependencies": [],
    "deployments": [],
    "disk": 0.0,
    "env": {},
    "executor": "",
    "healthChecks": [],
    "id": "/demo",
    "instances": 1,
    "labels": {},
    "maxLaunchDelaySeconds": 3600,
    "mem": 256.0,
    "ports": [
      3000
    ],
    "requirePorts": false,
    "storeUrls": [],
    "tasksHealthy": 0,
    "tasksRunning": 1,
    "tasksStaged": 0,
    "tasksUnhealthy": 0,
    "upgradeStrategy": {
      "maximumOverCapacity": 1.0,
      "minimumHealthCapacity": 1.0
    },
    "uris": [],
    "user": null,
    "version": "2015-04-06T04:50:23.809Z"
  }
]
```


Since we didn't specify a number of instances, it has started running just one instance. We can update that (or any other property) easily. In this example, we scale the application with one command to run on 3 instances:

```
$ dcos marathon app update demo instances=3
Created deployment 805cea1d-e4b1-4f17-a195-91c8d35c54ba
```

This has created a deployment. Deployments in Marathon occur whenever the current state of an application does not match its desired state. More documentation about deployments is available in the Marathon docs. While the deployment is active, it will show up in the current list of deployments:

```
$ dcos marathon deployment list
[
  {
```

```
    "affectedApps": [
      "/demo"
    ],
    "currentActions": [
      {
        "action": "ScaleApplication",
        "app": "/demo"
      }
    ],
    "currentStep": 1,
    "id": "9f95316e-f983-4dcf-a015-da6bd77cb397",
    "steps": [
      [
        {
          "action": "ScaleApplication",
          "app": "/demo"
        }
      ]
    ],
    "totalSteps": 1,
    "version": "2015-04-06T04:59:15.770Z"
  }
]
```

Each time the application's definition changes, a new version is saved by Marathon. You can see the available versions of the demo application like this:

```
$ dcos marathon app version list demo
[
  "2015-04-06T04:59:15.770Z",
  "2015-04-06T04:58:53.180Z"
]
```

This version can be used to inspect an application definition at a certain time:

```
$ dcos marathon app show --app-version="2015-04-06T04:58:53.180Z" demo
{
  "args": null,
  "backoffFactor": 1.15,
  "backoffSeconds": 1,
  "cmd": "python -m SimpleHTTPServer $PORT",
  "constraints": [],
  "container": {
    "docker": {
      "image": "superguenter/demo-app",
      "network": null,
      "parameters": [],
      "portMappings": null,
      "privileged": false
    },
    "type": "DOCKER",
```

```
    "volumes": []
  },
  "cpus": 0.01,
  "dependencies": [],
  "disk": 0.0,
  "env": {},
  "executor": "",
  "healthChecks": [],
  "id": "/demo",
  "instances": 10,
  "labels": {},
  "maxLaunchDelaySeconds": 3600,
  "mem": 256.0,
  "ports": [
    3000
  ],
  "requirePorts": false,
  "storeUrls": [],
  "upgradeStrategy": {
    "maximumOverCapacity": 1.0,
    "minimumHealthCapacity": 1.0
  },
  "uris": [],
  "user": null,
  "version": "2015-04-06T04:58:53.180Z"
}
```


Now that we're done testing, we can optionally stop our application manually:

```
$ dcos marathon app stop demo
Created deployment fd32b7d2-cb70-4b54-ba62-b57feeb2c0a6
```

This will reduce the number of running instances to 0. We can then remove the application:

```
$ dcos marathon app remove demo
```

The DCOS Package Registry

The [Mesosphere Universe] defines an open specification for the representation and transfer of DCOS package metadata.

The DCOS CLI provides a subcommand called `dcos package` for interacting with remote package metadata sources, installing and uninstalling DCOS packages, and listing what's currently installed.

To accomplish all of this, the package CLI is configured with:

- How to contact the cluster's Marathon instance.
- The location of the local package cache.
- The source for fetching package metadata. Package sources are specified as URLs. If multiple sources are configured, then packages are resolved from those sources in the order they are listed in the config. By default for DCOS Early Access one package source is configured: https://github.com/mesosphere/universe/archive/ea.zip

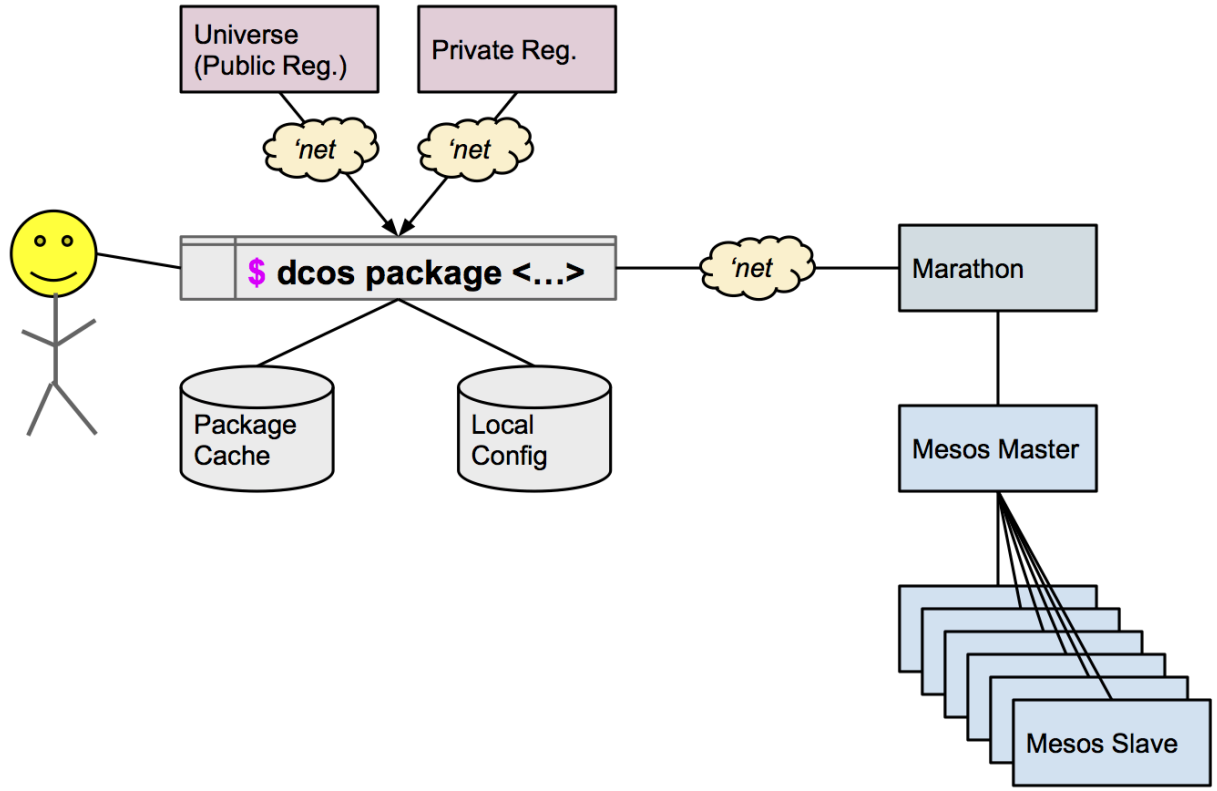


Figure 1: Simplified interaction between the Package CLI, package registry, and Marathon.

`dcos package install <your-package>` is within reach! Given the Marathon app definition for your own software, a sufficiently motivated EAP participant could add a package definition for it to an internally hosted registry.

Changing DCOS Cluster

You may want to use the `dcos` utility to administer multiple DCOS clusters. You can inspect the current configuration with `dcos config`:

```
$ dcos config show
marathon.host=ea-test.us-west-2.elb.amazonaws.com
marathon.port=8080
package.cache=tmp/cache
package.sources=['https://github.com/mesosphere/universe/archive/ea.zip']
```

To set the `marathon.host` property to the new DCOS master:

```
$ dcos config set marathon.host ea-test-2.us-west-2.elb.amazonaws.com
```

This will be persisted to the DCOS configuration file too. You can verify that this value has been updated:

```
$ dcos config show marathon.host
ea-test-2.us-west-2.elb.amazonaws.com
```

Known Issues

There is currently no tabular output. Each command defaults to JSON and if you want to query for certain parts you can, for example, use jq. The next version of the CLI may offer a tabular output mode for ease of use and a JSON mode for easy programmability.

Work is ongoing to integrate the Mesos CLI project and to provide one unified command line tool. For now, the two utilities are separate.

Specifying a git branch other than `master` for `git://` package sources is currently unsupported.

Upgrading or Uninstalling

To uninstall the CLI, simply remove the folder specified when you installed it. If you didn't change the command provided in the installation guide, this will simply be `~/.dcos-cli`.

To upgrade, simply remove the folder and re-run the installation script.