Container Orchestration Test Plans:

Test #1: Kill a running service task to watch Marathon rescheduling behavior

Step 1: Review Marathon App Definition nginx.json:

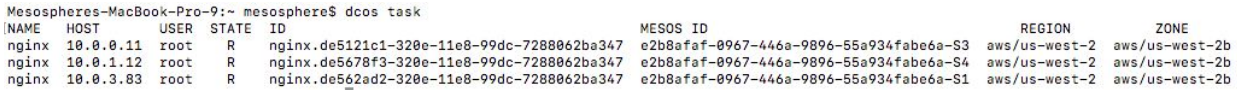
```
{
  "id": "/nginx-example",
  "backoffFactor": 1.15,
  "backoffSeconds": 1,
  "container": {
    "portMappings": [
      {
        "containerPort": 80,
        "hostPort": 0,
        "labels": {
          "VIP_0": "/nginx-example:80"
        },
        "protocol": "tcp",
        "servicePort": 10101,
        "name": "nginx-example"
      }
    ],
    "type": "DOCKER",
    "volumes": [],
    "docker": {
      "image": "nginx",
      "forcePullImage": false,
      "privileged": false,
      "parameters": []
    }
  },
  "cpus": 0.1,
  "disk": 0,
  "healthChecks": [
    {
      "gracePeriodSeconds": 5,
      "intervalSeconds": 10,
      "maxConsecutiveFailures": 2,
      "portIndex": 0,
      "timeoutSeconds": 10,
      "delaySeconds": 5,
      "protocol": "MESOS_TCP",
      "portName": "nginx-example"
    }
  ],
  "instances": 3,
  "maxLaunchDelaySeconds": 30,
  "mem": 128,
  "gpus": 0,
  "networks": [
    {
      "mode": "container/bridge"
    }
  ],
  "requirePorts": false,
  "upgradeStrategy": {
    "maximumOverCapacity": 1,
    "minimumHealthCapacity": 1
  },
  "killSelection": "YOUNGEST_FIRST",
  "unreachableStrategy": {
    "inactiveAfterSeconds": 1,
    "expungeAfterSeconds": 5
  },
  "fetch": [],
  "constraints": []
}
```

Step 2: Deploy nginx.json application definition

```
dcos marathon app add nginx.json
```

Step 3: Use the DC/OS CLI to observe running tasks

```
dcos task
```



```
Mesospheres-MacBook-Pro-9:~ mesosphere$ dcos task
NAME   HOST       USER  STATE  ID                                           MESOS ID                                     REGION        ZONE
nginx  10.0.0.11  root    R    nginx.de5121c1-320e-11e8-99dc-7288062ba347  e2b8afaf-0967-446a-9896-55a934fabe6a-S3  aws/us-west-2  aws/us-west-2b
nginx  10.0.1.12  root    R    nginx.de5678f3-320e-11e8-99dc-7288062ba347  e2b8afaf-0967-446a-9896-55a934fabe6a-S4  aws/us-west-2  aws/us-west-2b
nginx  10.0.3.83  root    R    nginx.de562ad2-320e-11e8-99dc-7288062ba347  e2b8afaf-0967-446a-9896-55a934fabe6a-S1  aws/us-west-2  aws/us-west-2b
```

Step 4: Use the DC/OS CLI to kill a running container

dcos marathon task kill **<ID>**

Step 5: Observe rescheduling behavior on DC/OS UI

- Task reschedule to another node

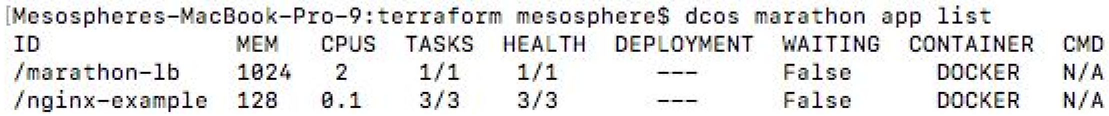
Test #2: Kill a running application to watch Marathon rescheduling

Step 1: Deploy Marathon App Definition nginx.json:

If not already deployed, deploy the nginx.json application definition from Test #1

Step 2: Use the DC/OS CLI to observe running applications

```
dcos marathon app list
```



```
[Mesospheres-MacBook-Pro-9:terraform mesosphere$ dcos marathon app list
 ID              MEM    CPUS  TASKS  HEALTH  DEPLOYMENT  WAITING  CONTAINER  CMD
 /marathon-lb    1024    2     1/1    1/1       ---      False      DOCKER   N/A
 /nginx-example  128    0.1    3/3    3/3       ---      False      DOCKER   N/A
```

Step 3: Use the DC/OS CLI to kill a running application

```
dcos marathon app kill nginx-example
```

Step 4: Observe rescheduling behavior in DC/OS UI

- All three instances will be killed simultaneously
- Observe reschedule by looking at the HOST IP in the GUI
- Containers are rescheduled to another available node

Test #3 Expose a service using Marathon-LB

Step 1: Add Labels to Application Definition

```
{
  "labels": {
    "HAPROXY_GROUP": "external",
    "HAPROXY_0_VHOST": "<PUBLIC_NODE_IP>"
  },
```

Step 2: Access Service

```
http://<PUBLIC_NODE_IP>
```

Test #4 Service Discovery using VIPs

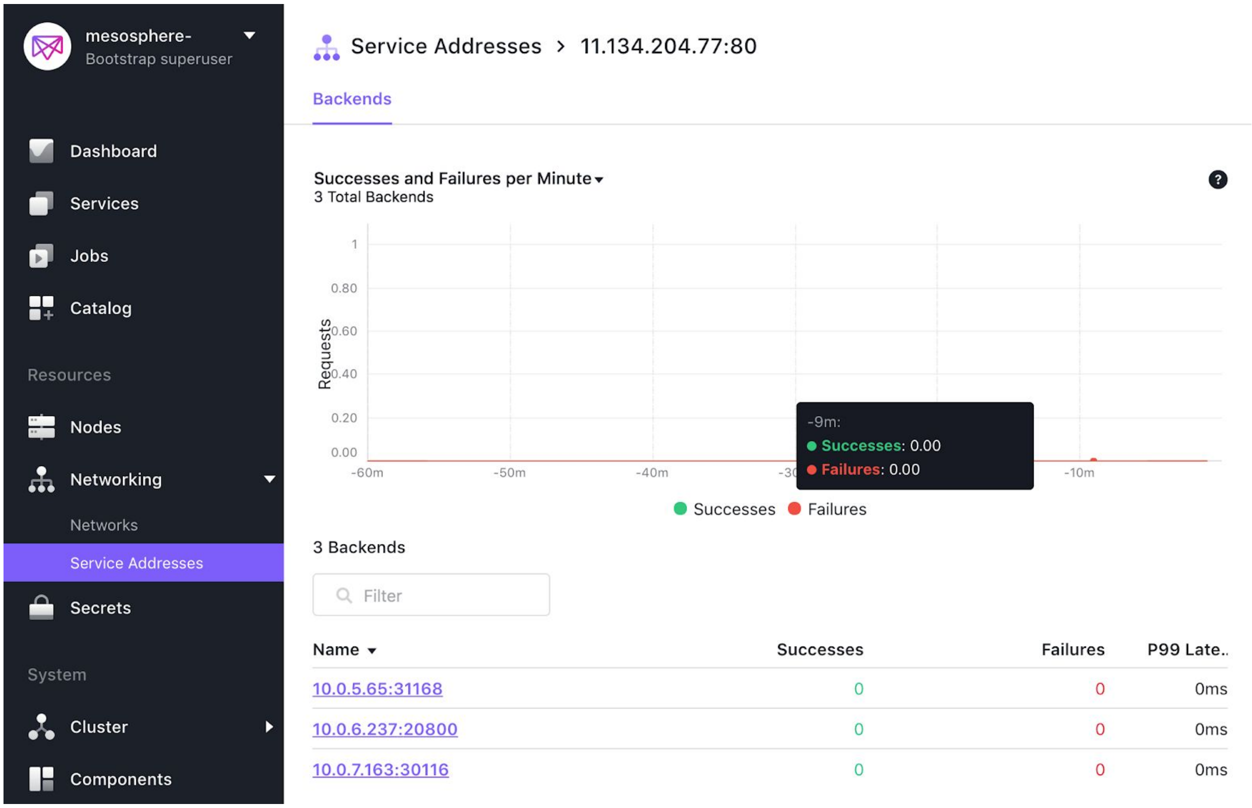
Step 1: Review and Deploy Marathon App Definition nginx.json:

If not already deployed, deploy the nginx.json application definition from Test #1. Take a look under the “labels” parameter to see the usage of a Name-based VIP (Virtual IP Address)

```
"labels": {
          "VIP_0": "/nginx-example:80",
```

Step 2: View L4 Minuteman Service Addresses (Named-Based VIPs) in the UI

- Note that the existing # of service addresses correspond to the instance count in your nginx-example.json application definition
- There is currently no load traffic, we can optionallyl generate this in a next step



3 Backends

Name	Successes	Failures	P99 Late..
10.0.5.65:31168	0	0	0ms
10.0.6.237:20800	0	0	0ms
10.0.7.163:30116	0	0	0ms

Optional Step 1: Review load generator app definition nginx-load.json

```
{
  "id": "/nginx-load",
  "backoffFactor": 1,
  "backoffSeconds": 1,
  "cmd": "curl nginx-example.marathon.l4lb.thisdcos.directory && curl
nginx-example.marathon.l4lb.thisdcos.directory && curl
nginx-example.marathon.l4lb.thisdcos.directory && sleep $(( $RANDOM %
10 ))",
  "container": {
    "type": "MESOS",
    "volumes": []
  },
  "cpus": 0.1,
  "disk": 0,
  "instances": 4,
  "maxLaunchDelaySeconds": 1,
  "mem": 256,
  "gpus": 0,
  "networks": [
    {
      "mode": "host"
    }
  ],
  "portDefinitions": [],
  "requirePorts": true,
  "upgradeStrategy": {
    "maximumOverCapacity": 1,
    "minimumHealthCapacity": 1
  },
  "killSelection": "YOUNGEST_FIRST",
  "unreachableStrategy": {
    "inactiveAfterSeconds": 300,
    "expungeAfterSeconds": 600
  },
  "healthChecks": [],
  "fetch": [],
  "constraints": []
}
```

Optional Step 2: Deploy nginx-load.json app definition

```
dcos marathon app add nginx-load.json
```

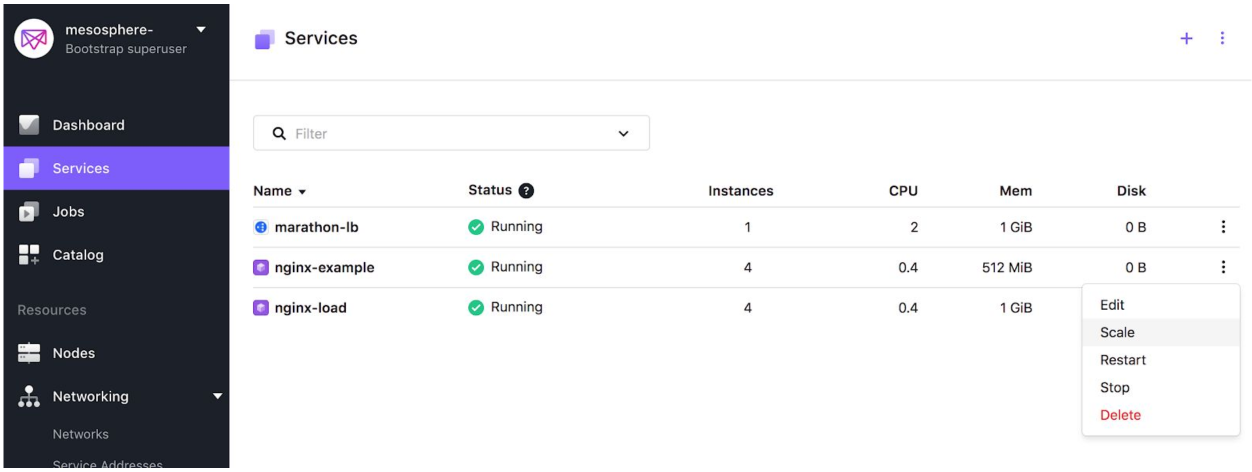
NOTE: You will see the status flapping between ‘READY’ and ‘RECOVERING’. This is normal behavior of the load generator command. (see cmd parameter in step 3)

Step 3: Scale nginx-example service

CLI:

```
dcos marathon app update <APP_ID> instances=<TOTAL_DESIRED_INSTANCES>
```

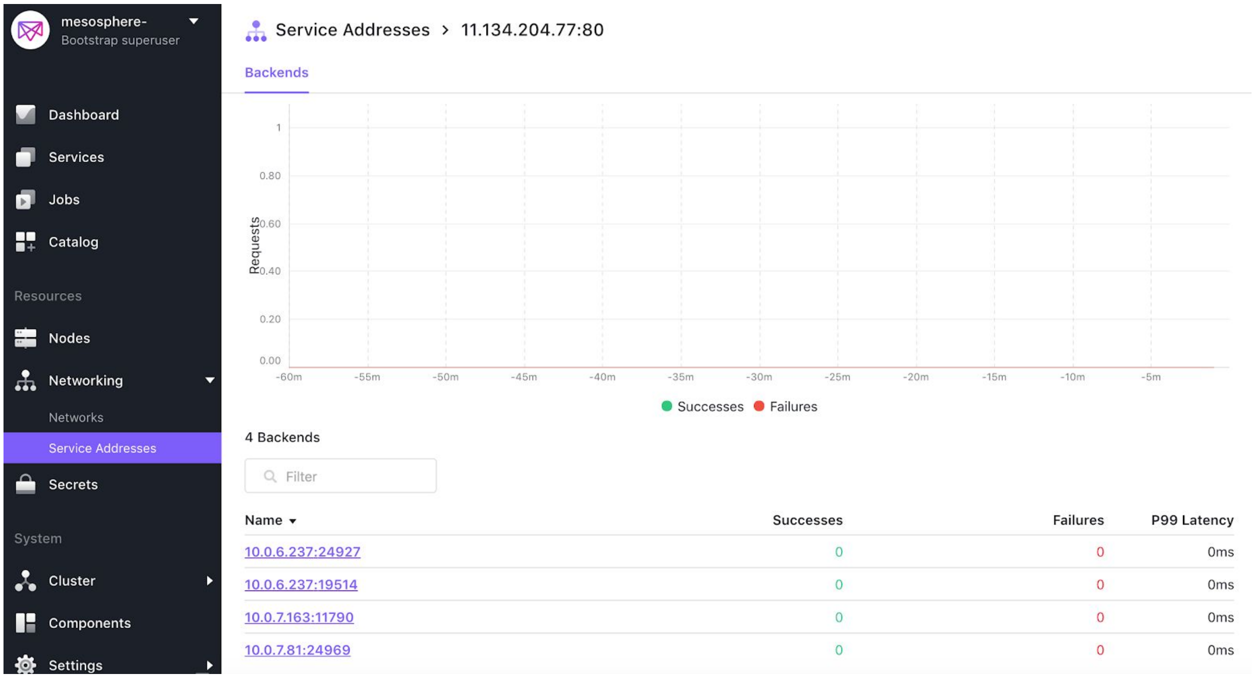
GUI:



Name	Status	Instances	CPU	Mem	Disk
marathon-lb	Running	1	2	1 GiB	0 B
nginx-example	Running	4	0.4	512 MiB	0 B
nginx-load	Running	4	0.4	1 GiB	

Step 4: Return to Service Addresses tab in the UI and observe Service Discovery

- nginx-example scaled from 3-4 instances and added to the backend pool
- If you followed the optional steps you should also see a load generated against these backends, load-balanced in round-robin



4 Backends

Name	Successes	Failures	P99 Latency
10.0.6.237:24927	0	0	0ms
10.0.6.237:19514	0	0	0ms
10.0.7.163:11790	0	0	0ms
10.0.7.81:24969	0	0	0ms

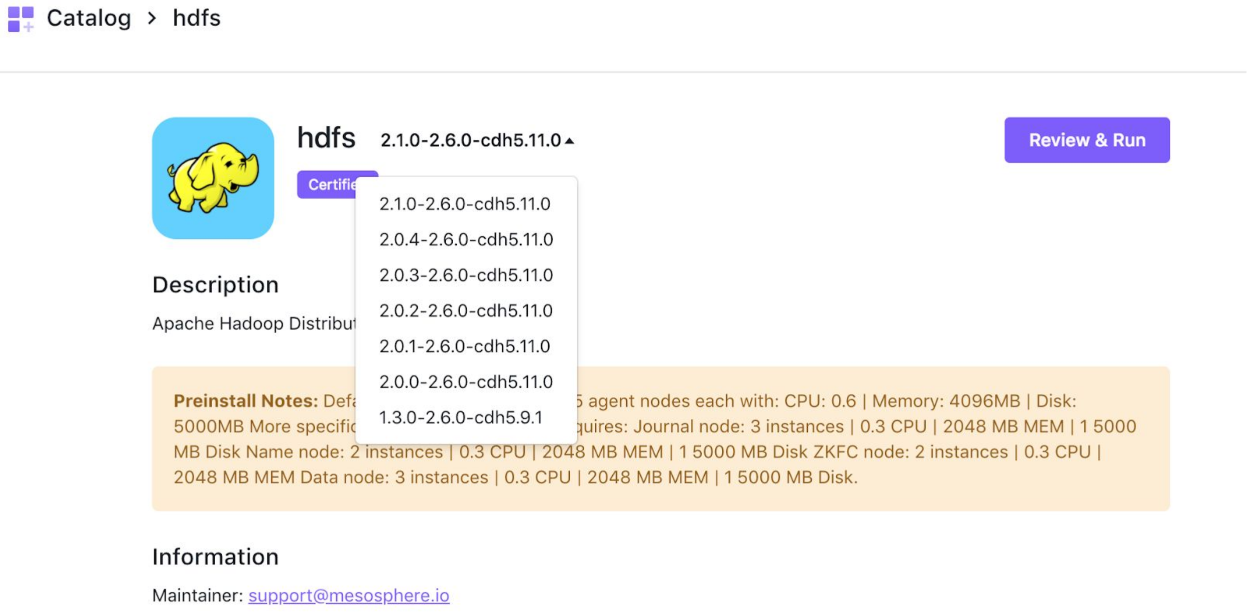
Data Services Test Plans:

Test #1 Deploy HA Certified Data Service

- Mesosphere Certified Catalog packages are built to be highly available and production ready by default
  - See Catalog Packages for a full list of existing Certified/Community packages

GUI Method:

Step 1: Navigate to the Catalog → HDFS Service → Select HDFS version → Review & Run



Description

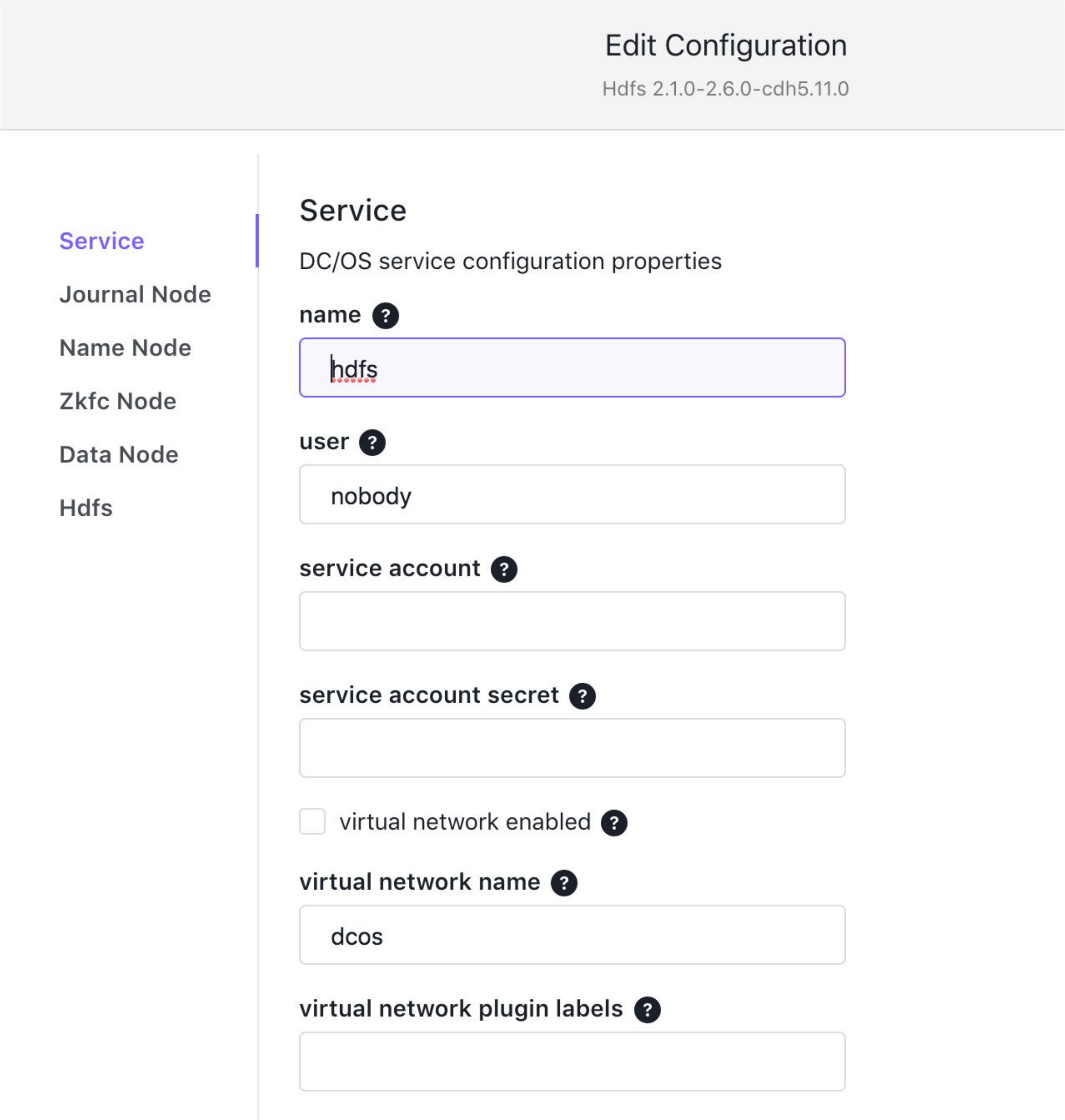
Apache Hadoop Distribut

Preinstall Notes: Defa 5 agent nodes each with: CPU: 0.6 | Memory: 4096MB | Disk: 5000MB More specific quires: Journal node: 3 instances | 0.3 CPU | 2048 MB MEM | 1 5000 MB Disk Name node: 2 instances | 0.3 CPU | 2048 MB MEM | 1 5000 MB Disk ZKFC node: 2 instances | 0.3 CPU | 2048 MB MEM Data node: 3 instances | 0.3 CPU | 2048 MB MEM | 1 5000 MB Disk.

Information

Maintainer: support@mesosphere.io

Step 2: Review default HDFS configuration and make any parameter changes necessary (i.e. storage, node count, CPU, memory, HDFS-specific config) → Review & Install


Edit Configuration

Hdfs 2.1.0-2.6.0-cdh5.11.0

Service

DC/OS service configuration properties

name

hdfs

user

nobody

service account

service account secret

□ virtual network enabled

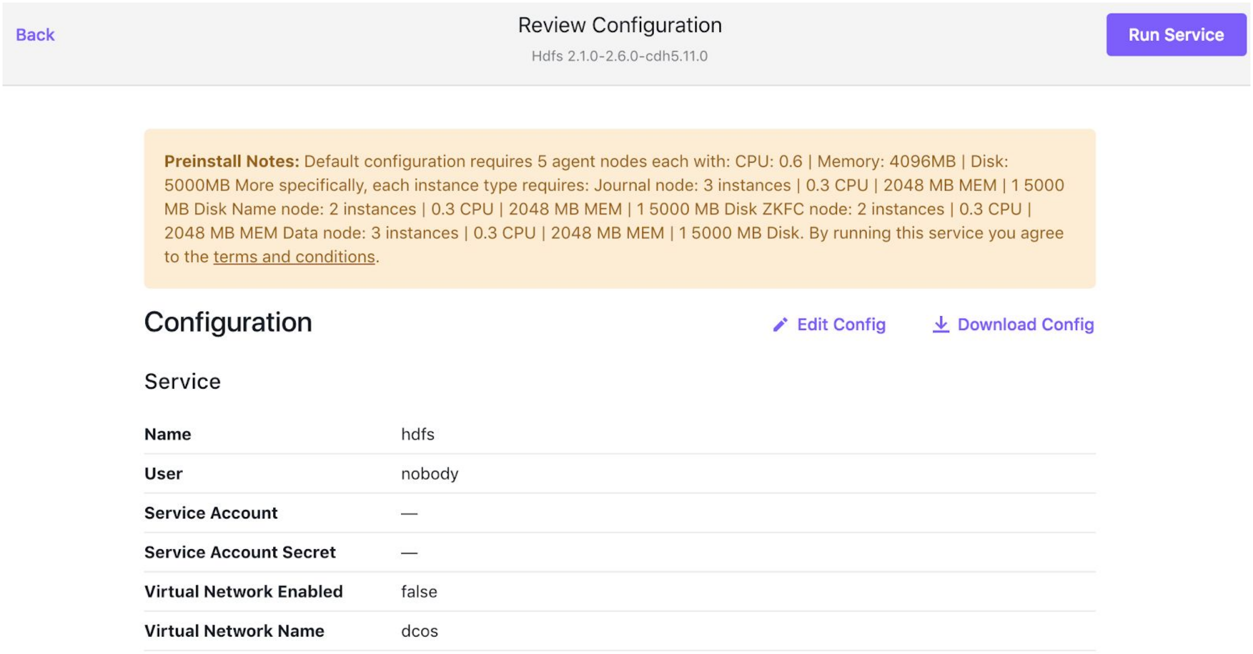
virtual network name

dcos

virtual network plugin labels

Step 3: Review Configuration and Run Service

- Note that you can also download any custom config for future re-use



Preinstall Notes: Default configuration requires 5 agent nodes each with: CPU: 0.6 | Memory: 4096MB | Disk: 5000MB More specifically, each instance type requires: Journal node: 3 instances | 0.3 CPU | 2048 MB MEM | 1 5000 MB Disk Name node: 2 instances | 0.3 CPU | 2048 MB MEM | 1 5000 MB Disk ZKFC node: 2 instances | 0.3 CPU | 2048 MB MEM Data node: 3 instances | 0.3 CPU | 2048 MB MEM | 1 5000 MB Disk. By running this service you agree to the terms and conditions.

Configuration

Service

Name	hdfs
User	nobody
Service Account	—
Service Account Secret	—
Virtual Network Enabled	false
Virtual Network Name	dcos

Step 4: View deployment in the GUI

CLI Method:

Step 1: Use DC/OS CLI to search for the HDFS package

```
dcos package search hdfs
```

Step 2: Install HDFS Package using DC/OS CLI

```
dcos package install hdfs --package-version=<package_version>
```

```
Note: It is possible to pass a custom configuration by using the
--options=<options.json> flag
```

Step 3: View deployment in the GUI

Test #2 Run a Spark HDFS Job

Access SMACK stack Github repo

[Github: SMACK Stack Tutorial]

- Full tutorial with step by step instructions are provided in PDF

- Deployment of HDFS + Spark + Kafka tutorial guides a reader through a simple example of running a Spark job that reads a file from the HDFS service and from a Kafka queue.

NOTE: This tutorial will require at least 10 private agent nodes (m4.xlarge) to complete

Test #3 Upgrading Certified Data Service

Prerequisites:

- Enterprise DC/OS 1.10 or newer
- A DC/OS SDK-based Service with a version greater than 2.0.0-x
- The DC/OS CLI installed and available
- The service's subcommand available and installed on your local machine
  - You can install just the subcommand CLI by running dcos package install --cli <service-name>.

Step #1: If you are running an older version of the subcommand CLI that doesn't have the update command, uninstall and reinstall your CLI.

```
dcos package uninstall --cli <service-name>
dcos package install --cli <service-name>
```

Step #2: View available Upgrade/Downgrade version options

```
dcos <service-name> update package-versions
```

Step #3: Update CLI subcommand to new version

```
dcos package uninstall --cli <service-name>

dcos package install --cli <service-name>
--package-version="<package-version>"
```

Step #4: Initiate upgrade

```
dcos <service-name> update start
--package-version="<package-version>"
```

NOTE: If you are missing mandatory configuration parameters, the update command will return an error.

Step #5: Monitor Upgrade status

dcos <service> --name=<service-name> update status

Test #4 Updating Data Service Configurations

Step #1: Fetch full configuration of a service
dcos <service-name> describe > options.json

Step #2: Make any configuration changes

- Scaling example: Increase Kafka default broker count from default 3 → 4

Step #3: Update Configuration

dcos <service-name> update start --options=options.json

Step #4: Monitor Update status

dcos <service-name> update status

See Advanced Update Actions for more useful update commands reference

[WIP - To be added in V2] Test #5 Traditional Database Services using Local Persistent Drives

See DC/OS Storage: Persistent Volumes

GUI Method:

CLI Method:

DC/OS Security Test Plan:

Test #1 Role Based Access Control

Step 1: Make sure DC/OS Enterprise CLI is installed

```
dcos package install dcos-enterprise-cli --cli --yes
```

Step 2: Create group a and add users 1 & 2 using the DC/OS CLI

```
dcos security org groups create groupa
dcos security org users create -d User1 -p User1 User1
dcos security org users create -d User2 -p User2 User2
dcos security org groups add_user groupa User1
dcos security org groups add_user groupa User2
```

Step 3: Create group b and add users 3 & 4

```
dcos security org groups create groupb
dcos security org users create -d User3 -p User3 User3
dcos security org users create -d User4 -p User4 User4
dcos security org groups add_user groupb User3
dcos security org groups add_user groupb User4
```

Step 4: Create permission to access native Marathon instance using API method

```
curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
'{"description":""}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:adminrouter:service:marathon
```

Step 5: Give permission to native Marathon instance

```
curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
'{"description":"Give permission to groups"}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:adminrouter:service:marathon/grou
ps/groupa/full

curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
'{"description":"Give permission to groups"}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:adminrouter:service:marathon/grou
ps/groupb/full
```

Step 6: Create permission to the Mesos agent UI and API

```
curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
'{"description":"Create permission"}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:adminrouter:ops:slave
```

Step 7: Give permission to Mesos agent UI and API

```
curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
'{"description":"Give permission"}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:adminrouter:ops:slave/groups/grou
pa/full

curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
'{"description":"Give permission"}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:adminrouter:ops:slave/groups/grou
pb/full
```

Step 8: Create permission to launch DC/OS services

NOTE: groupa and groupb only have access to launch services in their respective team group folder (e.g. /groupa/postgres)

```
curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
'{"description":"Create permission"}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:service:marathon:marathon:service
s:groupa

curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
'{"description":"Create permission"}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:service:marathon:marathon:service
s:groupb
```

Step 9: Give permission to launch DC/OS services

```
curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
'{"description":"Give permission"}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:service:marathon:marathon:service
s:groupa/groups/groupa/full

curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
'{"description":"Give permission"}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:service:marathon:marathon:service
s:groupb/groups/groupb/full
```

Step 10: Create permission to launch packages from the DC/OS Universe

Note: groupa and groupb only have access to launch services in their respective team group folder (e.g. /Grouop_A/postgres)

```
curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
'{"description":"Create permission"}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:adminrouter:package
```

Step 11: Give permission to launch packages from the DC/OS Universe

```
curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
'{"description":"Give permission"}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:adminrouter:package/groups/groupa
/full
```

```
curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
'{"description":"Give permission"}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:adminrouter:package/groups/groupb
/full
```

Step 12: Create permission to the Mesos master UI and API

```
curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
'{"description":"Create permission"}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:adminrouter:ops:mesos
```

Step 13: Give permission to the Mesos master UI and API

```
curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
'{"description":"Give permission"}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:adminrouter:ops:mesos/groups/grou
pa/full
```

```
curl -X PUT -k -H "Authorization: token=$(dcos config show
core.dcos_acs_token)" -H "Content-Type: application/json" -d
```

```
'{"description":"Give permission"}' $(dcos config show
core.dcos_url)/acs/api/v1/acls/dcos:adminrouter:ops:mesos/groups/grou
pb/full
```

Walkthrough Workflow:

1. Show Superuser full view
2. Show locked-down user view
3. Login to groupa/groupb personas and test deploy nginx-example.json into root Marathon folder and watch it fail.
4. Retry the deployment into the group (i.e. /groupa/nginx-example.json) folder and watch it deploy successfully
5. Test deployment of catalog package into root folder and watch it fail
6. Retry the deployment into the group (i.e. /groupa/kafka) folder and watch it deploy successfully

Demo #2 LDAP Integration

- We will demo this integration using our own AD server to show functionality
- If time permits we can explore this further after initial few weeks of tackling tasks above