This repository | Search Pull requests Issues Marketplace Gist

dcos / **examples** Watch 67 Star 71 Fork 88

Code Issues 9 Pull requests 5 Projects 0 Insights

Branch: master **examples** / registry / **1.8** / Create new file Upload files Find file History

joerg84 Refactor example folder structure.		Latest commit 0795049 on Apr 5
..		
img	Refactor example folder structure.	2 months ago
README.md	Refactor example folder structure.	2 months ago

README.md

How to use Docker Registry on DC/OS

The Docker registry is a stateless, highly scalable server side application that stores and lets you distribute Docker images. This DC/OS package provides a private registry that can be made available to any other component in the system through a virtual IP with the default of `registry.marathon.l4lb.thisdcos.directory:5000` .

Note that this package will install with the default parameters, but in order to provide secure communication, Docker registry requires a valid TLS certificate and key to work properly and secure connections between the engines and registry hosting the cache. Follow this document to learn how to configure a download location for the certificate and key in the Advanced Installation “security” section.

In case you'd like to configure your nodes to use the Docker Registry without TLS security and skip all configuration related to TLS security and certificates, please check out the section Using an insecure Docker registry for details on how to configure the Docker engine in the nodes in your DC/OS cluster to work with an insecure registry.

- Estimated time for completion: up to 30 minutes
- Target audience:
- Operators
- Application admins
- Scope: You'll learn how to install the Docker registry and how to use it to push and pull docker images, and to act as a proxy for another registry.

Table of Contents:

Prerequisites

- A running DC/OS 1.8 cluster with at least a bootstrap node, a master and an agent.
- SSH access to the nodes

Creating a valid TLS certificate

The Docker registry requires a TLS certificate and a TLS key to operate. This certificate will also need to be installed in the nodes that need to access the Docker registry. This package uses two downloadable URIs to store the certificate and key. The certificate and key will be shared with the application from a host with a running web server. For the purposes of this example, we will generate and host the certificate in the bootstrap node of the cluster, leveraging the fact that it already runs an NGINX web server by default. Any other web server accessible from the private nodes of the cluster could also be used.

Note that above means that the location of your secrets will be OPEN in the bootstrap node to download, at least during package installation, so this process is not considered adequate for production setups. Consider making these files available only during the process of booting up the registry, and then moving them outside of the “downloadable” path for security.

A production install should count with proper valid certificates, or use a third party certificate authority such as Let’s Encrypt. During this tutorial, we will use self-signed certs as outlined in the Docker docs.

Create a self-signed certificate

To create a self-signed certificate in the *bootstrap* node with the correct common name and subject alternative names do as follows.

Open a terminal in the *bootstrap* node, and generate a certificate in the `/genconf/serve` directory of your DC/OS installation. Note that this assumes that your installation directory for DC/OS in the bootstrap node is `~` . If your installation path differs, please generate these files in the adequate `/genconf/serve` location):

```
$ cd ~/genconf/serve   #use the /genconf/serve location of your install
$ openssl req -newkey rsa:4096 -nodes -sha256 -keyout domain.key -x509 -days 365 -out domain.crt

Generating a 4096 bit RSA private key
.............................................................................................................
.................................................................++
writing new private key to 'domain.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:NY
Locality Name (eg, city) [Default City]:NYC
Organization Name (eg, company) [Default Company Ltd]:mycompany
Organizational Unit Name (eg, section) []:myorg
Common Name (eg, your name or your servers hostname) []:registry.marathon.l4lb.thisdcos.directory
Email Address []:myemailaddress@mesosphere.com
```

After executing this command, your certificate and key should be available in the current directory:

```
# ls -la domain.*
-rw-r--r--. 1 root root 2130 Oct 21 07:24 domain.crt
-rw-r--r--. 1 root root 3272 Oct 21 07:24 domain.key
# cat domain.crt
-----BEGIN CERTIFICATE-----
MIIF9zCCA9+gAwIBAgIJAJEcQ0sL/gcJMA0GCSqGSIb3DQEBCwUAMIGRMQswCQYD
VQQGEwJVUzELMAkGA1UECAwCTlkxDDAKBgNVBAcMA05ZQzETMBEGA1UECgwKbWVz
(...)
bFot/v8ZSth2VHyo2+jxeXUGtbfa1hrvd0YbTLjdsnnZcQoDcVv7VTerJkMNoCrn
0ymy4CLVZ8lRrW6VTHDAe+MAapN8QUYPxJcy1p0ohhlgOYITR8mraWLe2eC/sWym
Ecp0VKA7+x7f2jBwzSIK8k5HJI/GCzJwre5RGTcYaIMxseQtVk+Mi6BJTg==
-----END CERTIFICATE-----
```

These `registry-certificate` and `registry-key` files will be used by the application to facilitate authentication to agents that request to download images from the registry.

Copy certificate to all agents

The next step to facilitate that authentication is to add the certificate to all of the agents in the cluster, so that whenever any of them wants to download an image from the registry, they can use this certificate to validate the connection. This is achieved by distributing the `domain.crt` file to each of the agents running Docker, and saving it in all agent nodes in the cluster as:

```
/etc/docker/certs.d/registry.marathon.l4lb.thisdcos.directory:5000/ca.crt
```

We will upload the certificate file to one of the *MASTER* nodes, and then execute a loop to open an SSH connection to each of the agents, and add the certificate to the agent's trusted certificate list in the path above. Note that for this to work, the Master node must have a valid SSH key/ID stored allowing it to connect to other agents. If you have a working key in a `.pem` file in your computer that is valid for your agent nodes, you can copy it to the MASTER node with:

```
$ scp -i my_key.pem my_key.pem  centos@master:~
#where "centos" is a valid user to log into the master node, and
#"master" is the IP address or name of your MASTER node
```

Also, the nodes in your DC/OS cluster should be configured to allow sudo from non-tty. In order to enable that, some CentOS and RedHat nodes require to edit the `/etc/sudoers` file and remove or comment-out the line:

```
Defaults requiretty
```

Otherwise, you'll run into this error when running the SSH command loops below:

`sudo: sorry, you must have a tty to run sudo`

Alternatively, you can also use other methods to copy this file to the right location in every agent of the cluster, such as a configuration management tool or script. The files could also come from a file share mounted to all of the nodes, like Azure Files.

So, SSH to one of your Master nodes and execute the steps described in the following.

Add the key that was copied to the Master in the step above to the SSH keychain:

```
$ eval `ssh-agent -s` && ssh-add ~/my_key.pem
Agent pid 10638
Identity added: /home/centos/my_key.pem (/home/centos/my_key.pem)
```

Make sure that `jq` is installed:

```
$ sudo yum install -y epel-release && sudo yum install -y jq
```

Copy the `domain.crt` file from your terminal to the master node:

```
$ export BOOTSTRAP_IP=[your bootstrap node IP address]
$ export BOOTSTRAP_PORT=[your bootstrap node's TCP port]
$ curl -O $BOOTSTRAP_IP:$BOOTSTRAP_PORT/domain.crt
```

Find out and store the list of your agent nodes IP addresses:

```
$ MESOS_AGENTS=$(curl -sS master.mesos:5050/slaves | jq '.slaves[] | .hostname' | tr -d '"');
```

Configure your agents to accept liberal TCP connections:

```
$ for i in $MESOS_AGENTS; do ssh "$i" -oStrictHostKeyChecking=no "sudo sysctl -w net.netfilter.nf_conntrack_t
```

Create a temporary `/etc/privateregistry/certs` directory in your agents:

```
$ for i in $MESOS_AGENTS; do ssh "$i" -oStrictHostKeyChecking=no "sudo mkdir --parent /etc/privateregistry/ce
```

Copy the certificate and key to your home directory in the agents:

```
$ for i in $MESOS_AGENTS; do scp -o StrictHostKeyChecking=no ./domain.* "$i":~/; done
```

Move the certificate and key files to the temporary directory:

```
$ for i in $MESOS_AGENTS; do ssh "$i" -oStrictHostKeyChecking=no "sudo mv ./domain.* /etc/privateregistry/cer
```

Create the directory for holding the certificates of the registry that we will create in DC/OS:

```
$ for i in $MESOS_AGENTS; do ssh "$i" -oStrictHostKeyChecking=no "sudo mkdir --parent /etc/docker/certs.d/reg
```

Copy the certificate and key files to the directory of the DC/OS registry:

```
$ for i in $MESOS_AGENTS; do ssh "$i" -oStrictHostKeyChecking=no "sudo cp /etc/privateregistry/certs/domain.c
```

Restart the docker daemon:

```
$ for i in $MESOS_AGENTS; do ssh "$i" -oStrictHostKeyChecking=no "sudo systemctl restart docker"; done
```

Optionally, for additional security you can modify permissions as so:

```
$ for i in $MESOS_AGENTS; do ssh "$i" -oStrictHostKeyChecking=no 'sudo chown -R root:root /etc/docker/certs.d
$ for i in $MESOS_AGENTS; do ssh "$i" -oStrictHostKeyChecking=no 'sudo chmod 444 -R /etc/docker/certs.d/regis
$ for i in $MESOS_AGENTS; do ssh "$i" -oStrictHostKeyChecking=no 'sudo chmod 400 -R /etc/docker/certs.d/regis
$ for i in $MESOS_AGENTS; do ssh "$i" -oStrictHostKeyChecking=no 'sudo systemctl restart docker'; done
```

The package can now be installed from the Universe, using the bootstrap's node IP address and TCP port to download the certificate and key file.

Insecure

In case you'd like to run an insecure registry without using any TLS certificates, you can configure the nodes in your DC/OS cluster to work without certificates or security.

This basically tells the Docker engine in each node to entirely disregard security for your registry. While this is relatively easy to configure the daemon in this way, it is very insecure. It does expose your registry to trivial MITM. Only use this solution for isolated testing or in a tightly controlled, air-gapped environment.

Run this in all agent nodes of your cluster:

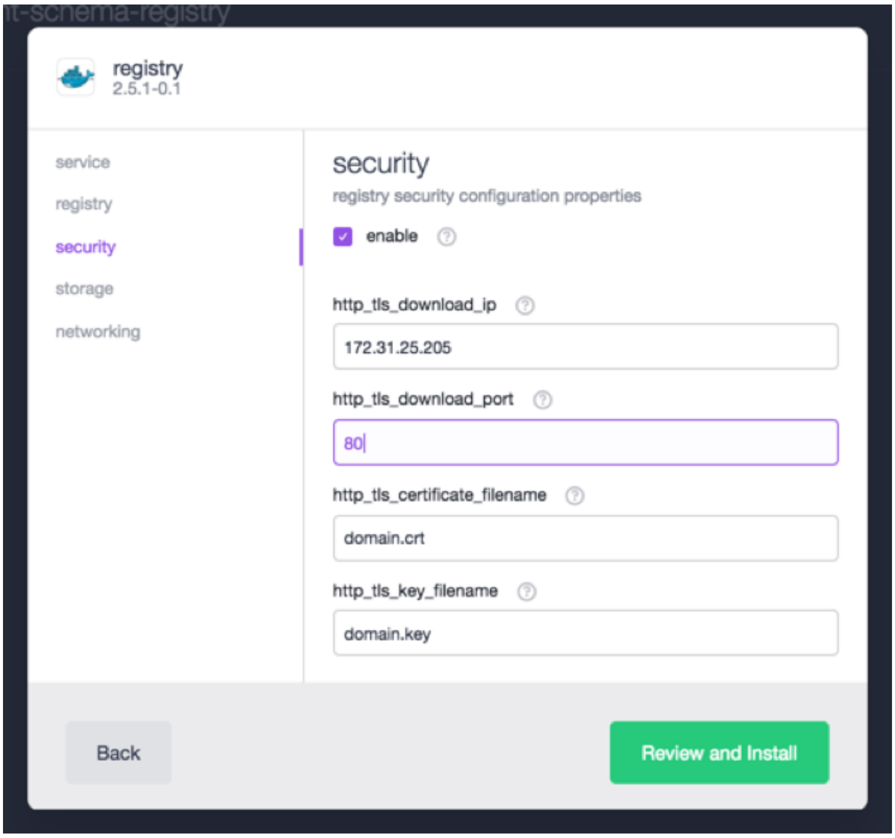
```
$ sudo tee /etc/systemd/system/docker.service.d/override.conf  <<-'EOF'
[Service]
EnvironmentFile=-/etc/sysconfig/docker
EnvironmentFile=-/etc/sysconfig/docker-storage
EnvironmentFile=-/etc/sysconfig/docker-network
ExecStart=
ExecStart=/usr/bin/docker daemon -H fd:// $OPTIONS \
         $DOCKER_STORAGE_OPTIONS \
         $DOCKER_NETWORK_OPTIONS \
         $BLOCK_REGISTRY \
         $INSECURE_REGISTRY \
         --storage-driver=overlay \
         --insecure-registry registry.marathon.l4lb.thisdcos.directory:5000
EOF

systemctl daemon-reload
systemctl restart docker
```

For more information on running without security, please check the Docker documentation.

Install

Log into DC/OS, go to Universe, and select the Registry package from Universe. Select `Advanced Installation` . Enter the bootstrap node's IP address and port in the `Advanced Installation` :



security

registry security configuration properties

☑ enable

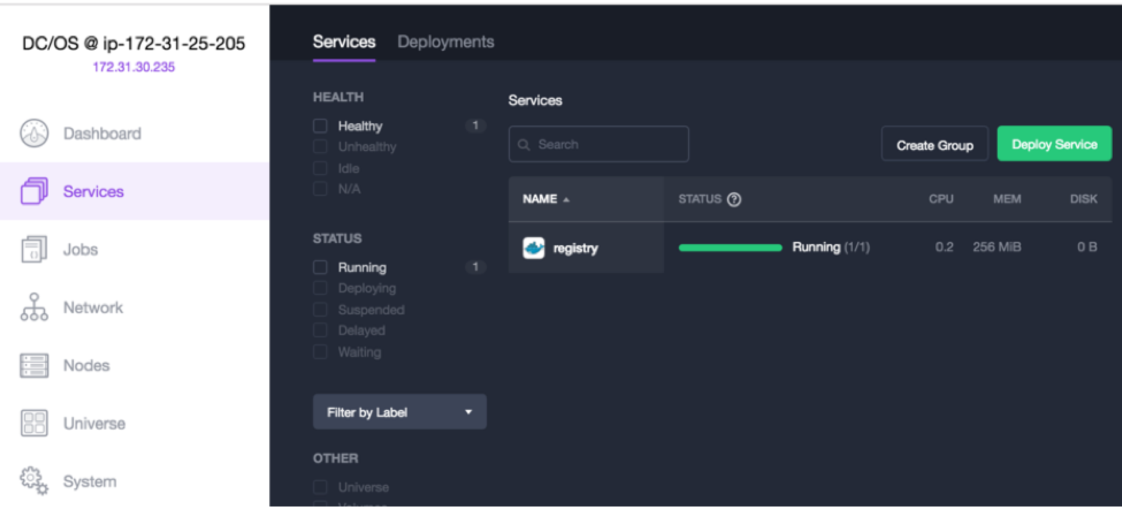
http_tls_download_ip: 172.31.25.205

http_tls_download_port: 80

http_tls_certificate_filename: domain.crt

http_tls_key_filename: domain.key

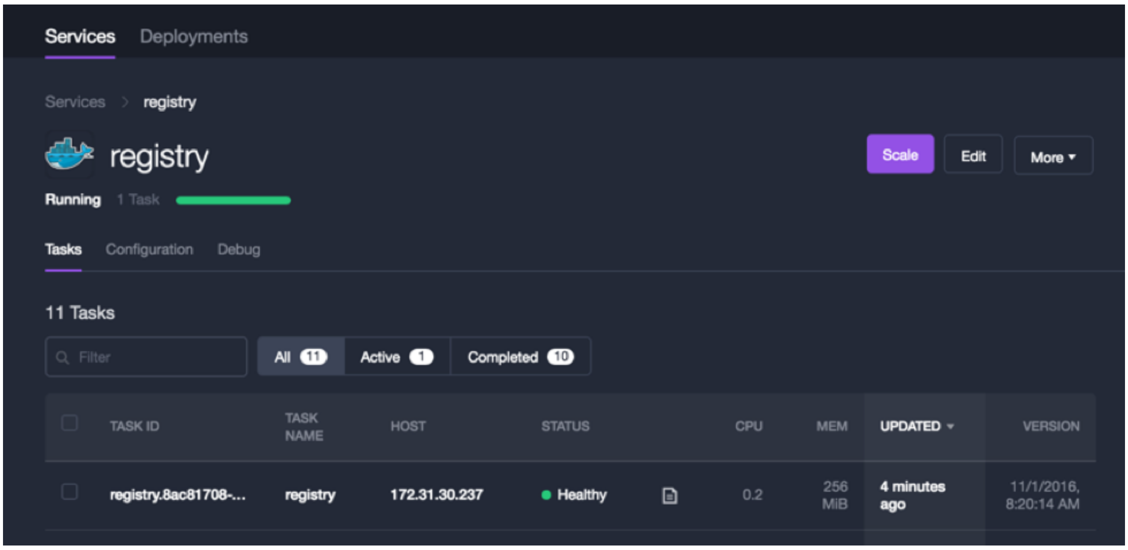
After installation, the package will be running under the `Services` tab:



Services

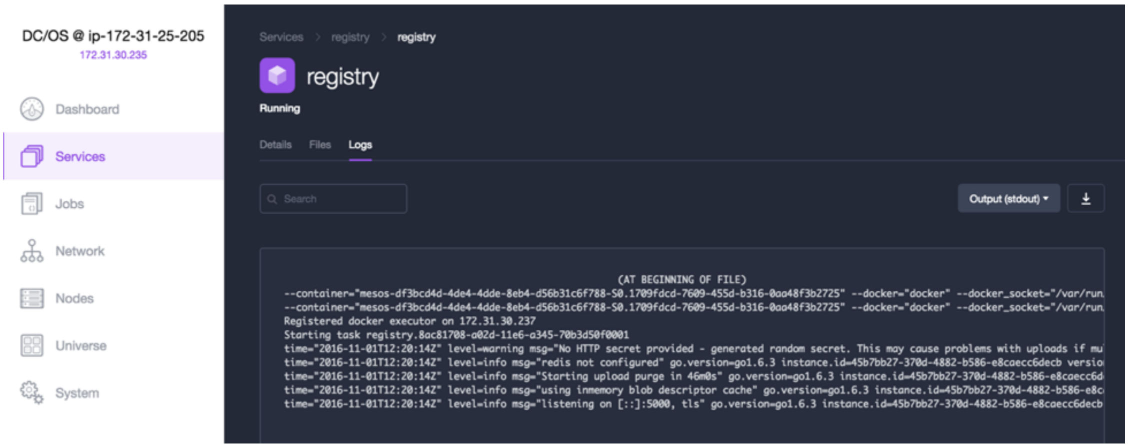
NAME	STATUS	CPU	MEM	DISK
registry	Running (1/1)	0.2	256 MiB	0 B

You can check the package instance’s correct functioning by clicking on the package name and looking at the task’s logs:



□	TASK ID	TASK NAME	HOST	STATUS		CPU	MEM	UPDATED ▾	VERSION
□	registry.8ac81708-...	registry	172.31.30.237	● Healthy	🗎	0.2	256 MiB	4 minutes ago	11/1/2016, 8:20:14 AM

Look at the logs and check the correct functioning:



```
(AT BEGINNING OF FILE)
--container="mesos-df3bcd4d-4de4-4dde-8eb4-d56b31c6f788-S0.1709fdcd-7609-455d-b316-0aa48f3b2725" --docker="docker" --docker_socket="/var/run
--container="mesos-df3bcd4d-4de4-4dde-8eb4-d56b31c6f788-S0.1709fdcd-7609-455d-b316-0aa48f3b2725" --docker="docker" --docker_socket="/var/run
Registered docker executor on 172.31.30.237
Starting task registry.8ac81708-a02d-11e6-a345-70b3d50f0001
time="2016-11-01T12:20:14Z" level=warning msg="No HTTP secret provided - generated random secret. This may cause problems with uploads if mu
time="2016-11-01T12:20:14Z" level=info msg="redis not configured" go.version=go1.6.3 instance.id=45b7bb27-370d-4882-b586-e8caecc6decb versio
time="2016-11-01T12:20:14Z" level=info msg="Starting upload purge in 46m0s" go.version=go1.6.3 instance.id=45b7bb27-370d-4882-b586-e8caecc6d
time="2016-11-01T12:20:14Z" level=info msg="using inmemory blob descriptor cache" go.version=go1.6.3 instance.id=45b7bb27-370d-4882-b586-e8c
time="2016-11-01T12:20:14Z" level=info msg="listening on [::]:5000, tls" go.version=go1.6.3 instance.id=45b7bb27-370d-4882-b586-e8caecc6decb
```

Use

Now that the Docker registry is installed and running, we can push and pull images from it.

Push an image to the local registry

Log into one of the nodes of the cluster, download any image from Docker Hub, and push it to the local registry:

```
# docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
43c265008fae: Pull complete
e4c030a565b1: Pull complete
685b7631c1ce: Pull complete
Digest: sha256:dedbce721065b2bcfae35d2b0690857bb6c3b4b7dd48bfe7fc7b53693731beff
Status: Downloaded newer image for nginx:latest
# docker tag nginx registry.marathon.l4lb.thisdcos.directory:5000/nginx
# docker push registry.marathon.l4lb.thisdcos.directory:5000/nginx
The push refers to a repository [registry.marathon.l4lb.thisdcos.directory:5000/nginx]
bc1394447d64: Pushed
6591c6f92a7b: Pushed
f96222d75c55: Pushed
latest: digest: sha256:dedbce721065b2bcfae35d2b0690857bb6c3b4b7dd48bfe7fc7b53693731beff size: 948
```

Check the contents of the local registry

```
# curl --insecure https://registry.marathon.l4lb.thisdcos.directory:5000/v2/_catalog
```

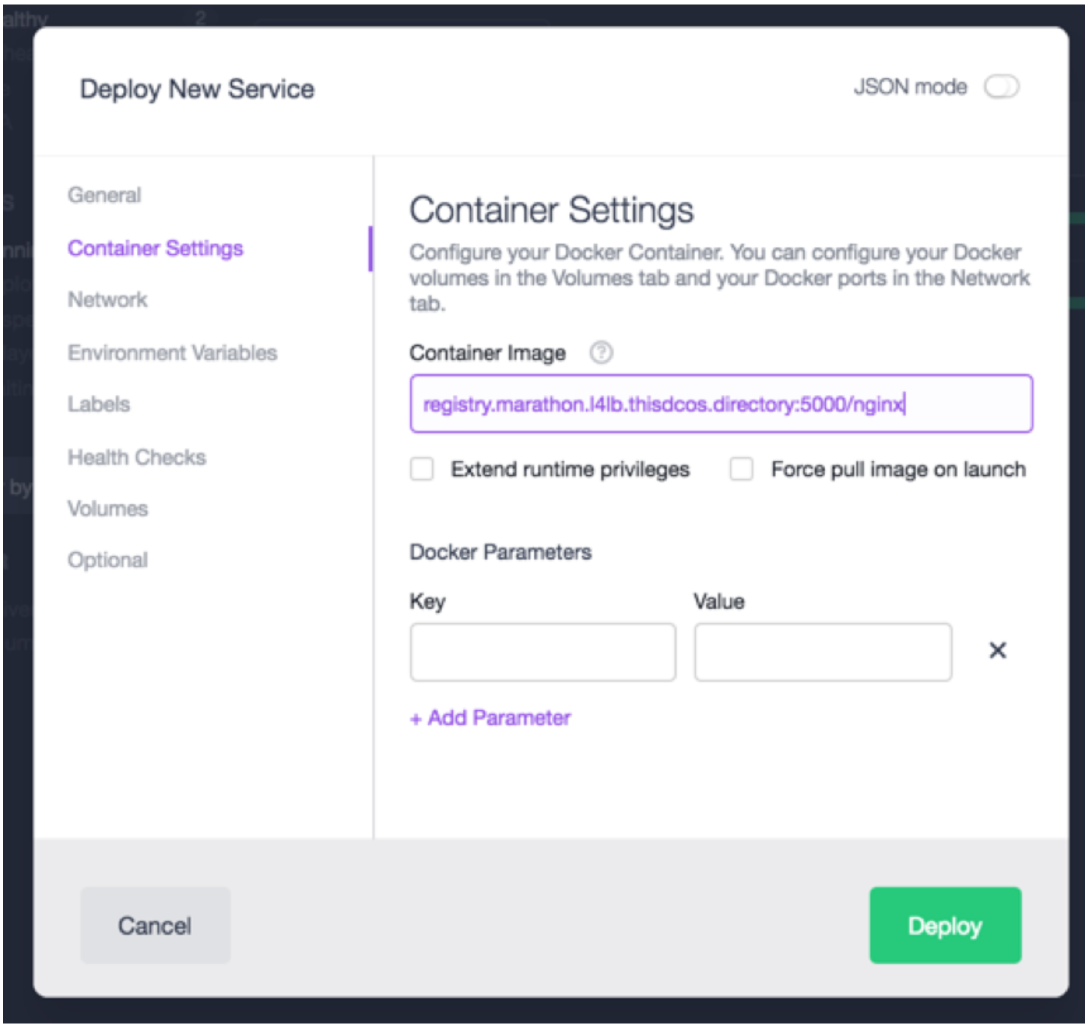
```
{"repositories":["nginx"]}
```

Pull image from local registry

The Docker registry at `registry.marathon.l4lb.thisdcos.directory:5000` is now available for use in your Marathon applications. It can be used to launch applications from the GUI or the CLI interfaces.

From the Marathon UI

Simply use `registry.marathon.l4lb.thisdcos.directory:5000` as a prefix in the `Container Settings` / `Container Image` field of the `Services` form:



Deploy New Service

Container Settings

Configure your Docker Container. You can configure your Docker volumes in the Volumes tab and your Docker ports in the Network tab.

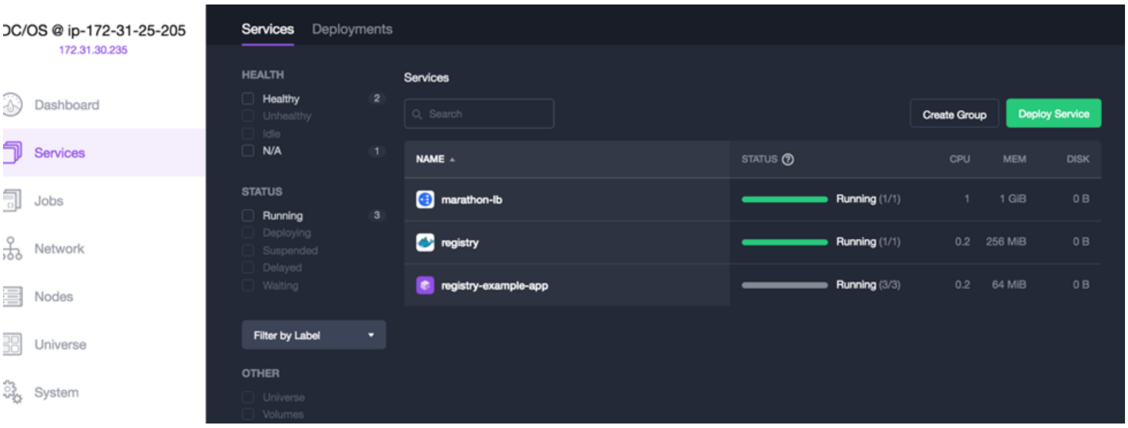
Container Image: registry.marathon.l4lb.thisdcos.directory:5000/nginx

□ Extend runtime privileges □ Force pull image on launch

Docker Parameters

Key	Value

Launch your application normally and check it's working correctly in the "Services" tab:



DC/OS @ ip-172-31-25-205
172.31.30.235

Services

NAME	STATUS	CPU	MEM	DISK
marathon-lb	Running (1/1)	1	1 GiB	0 B
registry	Running (1/1)	0.2	256 MiB	0 B
registry-example-app	Running (3/3)	0.2	64 MiB	0 B

From the CLI

Check that your DC/OS CLI is connected properly to the DC/OS cluster:

```
$ dcos node
   HOSTNAME          IP                          ID
172.31.30.236  172.31.30.236  df3bcd4d-4de4-4dde-8eb4-d56b31c6f788-S1
172.31.30.237  172.31.30.237  df3bcd4d-4de4-4dde-8eb4-d56b31c6f788-S0
```

Write a JSON file that uses the image from the local registry:

```
$cat registry-example-app.json
{
  "volumes": null,
  "id": "/registry-example-app",
  "cmd": null,
  "args": null,
  "user": null,
  "env": null,
  "instances": 3,
  "cpus": 0.2,
  "mem": 64,
  "disk": 0,
  "gpus": 0,
  "executor": null,
  "constraints": null,
  "fetch": null,
  "storeUrls": null,
  "backoffSeconds": 1,
  "backoffFactor": 1.15,
  "maxLaunchDelaySeconds": 3600,
  "container": {
    "docker": {
      "image": "registry.marathon.l4lb.thisdcos.directory:5000/nginx",
      "forcePullImage": false,
      "privileged": false,
      "portMappings": [
        {
          "containerPort": 80,
          "protocol": "tcp",
          "name": "registry-example-app",
          "servicePort": 10101,
          "labels": {
            "VIP_0": "/registry-example-app:80"
          }
        }
      ],
      "network": "BRIDGE"
    }
  },
  "healthChecks": null,
  "readinessChecks": null,
  "dependencies": null,
  "upgradeStrategy": {
    "minimumHealthCapacity": 1,
    "maximumOverCapacity": 1
  },
  "labels": {
    "HAPROXY_GROUP": "external"
  },
  "acceptedResourceRoles": null,
  "residency": null,
  "secrets": null,
  "taskKillGracePeriodSeconds": null,
  "portDefinitions": [
    {
      "port": 10101,
      "protocol": "tcp",
      "labels": {}
    }
  ],
  "requirePorts": false
}
```

Load that JSON into Marathon:

```
$ dcos marathon app add registry-example-app.json
```

Check that the application is running properly:

```
$ dcos marathon app list
ID                     MEM   CPUS  TASKS  HEALTH  DEPLOYMENT  CONTAINER  CMD
/marathon-lb           1024   1     1/1    1/1       ---         DOCKER   ['sse', '-m', 'http://master.mesos:8
/registry              256   0.2    1/1    1/1       ---         DOCKER   None
/registry-example-app   64   0.2    3/3    ---       ---         DOCKER   None
```

Storage options

Running the registry in a production environment will require that each machine in the cluster has an external volume mounted at the same location. External volumes can be backed by any number of systems, including NFS, CIFS, Ceph, and others. This will allow the registry to persist data to the external volume while still being able to run on any agent in the cluster, preventing against outages due to machine failure.

If you already have a mount point, great! Enter the path in the "Advanced Installation" options of the Universe UI. Alternatively you can use the CLI, by creating an `options.json` file that resembles the following example:

```
$ cat options.json
{
    "service": {
        "name": "registry-prod",
        "cpus": 2.0,
        "mem": 4096
    },
    "storage": {
        "host-volume": "/mnt/registry"
    }
}
```

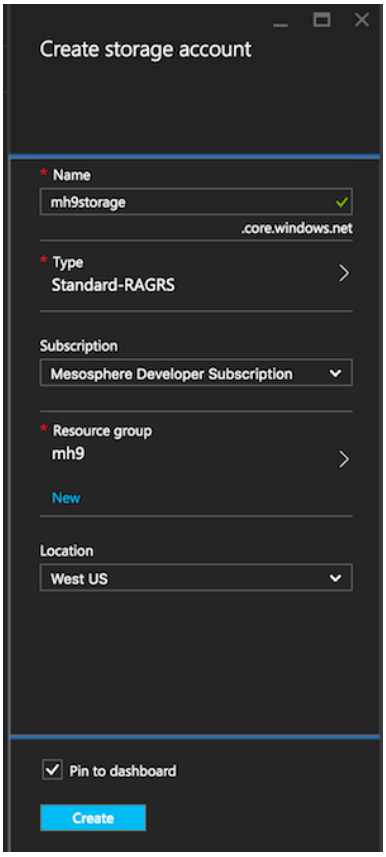
Then, install the Docker registry by running the following command:

```
$ dcos package install registry --options=options.json
```

If you don't have a file share set up and are looking for a solution, continue to the next section for instructions on how to set up a shares using CIFS on Microsoft Azure or NFS on Amazon EFS.

Creating a CIFS file share on Microsoft Azure

First, you need to create a Storage Account in the **same resource group** in which you've launched your DC/OS cluster. In this particular example, let's create the storage account `mh9storage` in the resource group `mh9` :



Create storage account

* Name
mh9storage
.core.windows.net

* Type
Standard-RAGRS

Subscription
Mesosphere Developer Subscription

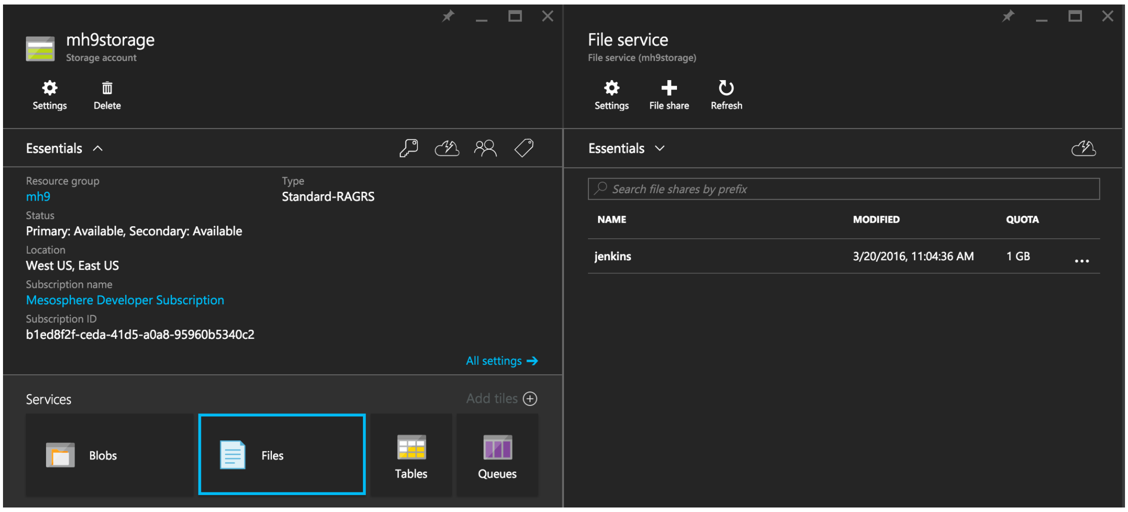
* Resource group
mh9
New

Location
West US

☑ Pin to dashboard

Create

Now, create a file share. In the example shown here it's called `registry` :



mh9storage
Storage account

Essentials

Resource group
mh9

Type
Standard-RAGRS

Status
Primary: Available, Secondary: Available

Location
West US, East US

Subscription name
Mesosphere Developer Subscription

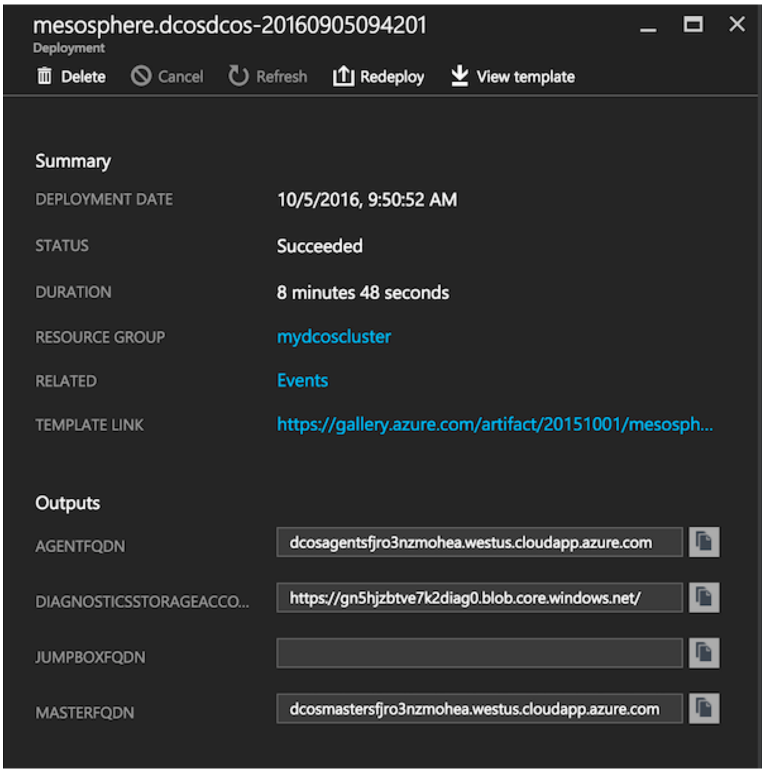
Subscription ID
b1ed8f2f-ceda-41d5-a0a8-95960b5340c2

File service
File service (mh9storage)

NAME	MODIFIED	QUOTA
jenkins	3/20/2016, 11:04:36 AM	1 GB

Mounting an Azure CIFS file share on Ubuntu

Log into the DC/OS master node. To determine the master, look up `MASTERFQDN` in the `Outputs` section of the deployment in Azure:



mesosphere.dcosdcos-20160905094201
Deployment

Summary

DEPLOYMENT DATE 10/5/2016, 9:50:52 AM

STATUS Succeeded

DURATION 8 minutes 48 seconds

RESOURCE GROUP mydcoscluster

RELATED Events

TEMPLATE LINK https://gallery.azure.com/artifact/20151001/mesosph...

Outputs

AGENTFQDN dcosagentsfjro3nzmohea.westus.cloudapp.azure.com

DIAGNOSTICSSTORAGEACCO... https://gn5hjzbtve7k2diag0.blob.core.windows.net/

JUMPBOXFQDN

MASTERFQDN dcosmastersfjro3nzmohea.westus.cloudapp.azure.com

Next, add the private SSH key locally:

```
$ ssh-add ~/.ssh/azure
Identity added: /Users/mhausenblas/.ssh/azure (/Users/mhausenblas/.ssh/azure)
```

Next, if you haven't already, tunnel the master node using the following command (note that the `-L 8000:localhost:80` is forwarding port `8000` from your local machine to port `80` on the remote host:

```
$ ssh azureuser@dcosmastersfjro3nzmohea.westus.cloudapp.azure.com -A -p 2200 -L 8000:localhost:80
```

On this node you can now mount the File Share we created in the previous step.

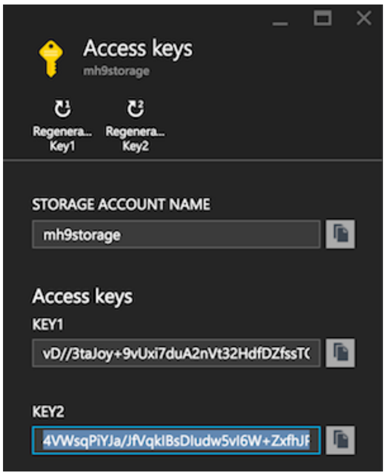
First, let's make sure that the CIFS mount utils are available:

```
$ sudo apt-get update && sudo apt-get -y install cifs-utils
```

And now we can mount the file share:

```
azureuser@dcosmastersfjro3nzmohea:~$ sudo mkdir -p /mnt/registry
azureuser@dcosmastersfjro3nzmohea:~$ sudo mount -t cifs    \
  //mh9storage.file.core.windows.net/registry /mnt/registry \
  -o vers=3.0,username=REDACTED,password=REDACTED,dir_mode=0777,file_mode=0777
```

Be sure to replace the `REDACTED` value for the `username` and `password` options with your username and password. Note that the value for `password` is `KEY2` from `Access keys`, as shown here:



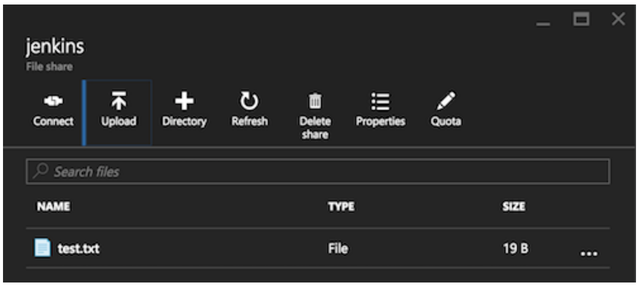
STORAGE ACCOUNT NAME
mh9storage

Access keys

KEY1
vD//3taJoy+9vUxi7duA2nVt32HdfDZfssT

KEY2
4VWsqPiYJa/JfVqklBsDludw5vl6W+ZxfhJ

To check if the file share works, we upload a test file via the Azure portal:



NAME	TYPE	SIZE
test.txt	File	19 B

If all is well, you should be able to list the contents of the mounted file share on the DC/OS master node:

```
azureuser@dcosmastersfjro3nzmohea:~$ ls -al /mnt/registry
total 1
-rwxrwxrwx 1 root root 19 Mar 20 11:21 test.txt
```

Finally, using the `pssh` tool, configure each of the DC/OS agents to mount the file share.

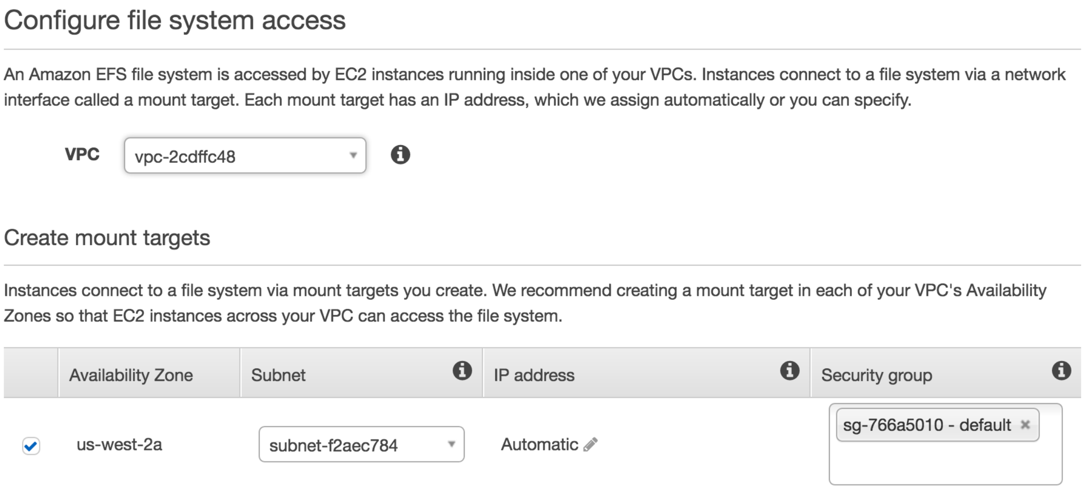
```
$ sudo apt-get install pssh
$ cat pssh_agents
10.0.3.226
10.0.3.227
10.0.3.228
10.0.3.229
10.0.3.230
10.0.7.0

$ parallel-ssh -O StrictHostKeyChecking=no -l azureuser -h pssh_agents "if [ ! -d "/mnt/registry" ]; then mkd
$ parallel-ssh -O StrictHostKeyChecking=no -l azureuser -h pssh_agents "mount -t cifs //mh9storage.file.core.
```

Creating an NFS file share with Amazon EFS

To start, open the Amazon EFS console, click `Create file system` and then `Create file system`. Make sure you are in the same availability zone as as your DC/OS cluster.

Select the VPC of your DC/OS cluster and click `Next Step`:



Configure file system access

An Amazon EFS file system is accessed by EC2 instances running inside one of your VPCs. Instances connect to a file system via a network interface called a mount target. Each mount target has an IP address, which we assign automatically or you can specify.

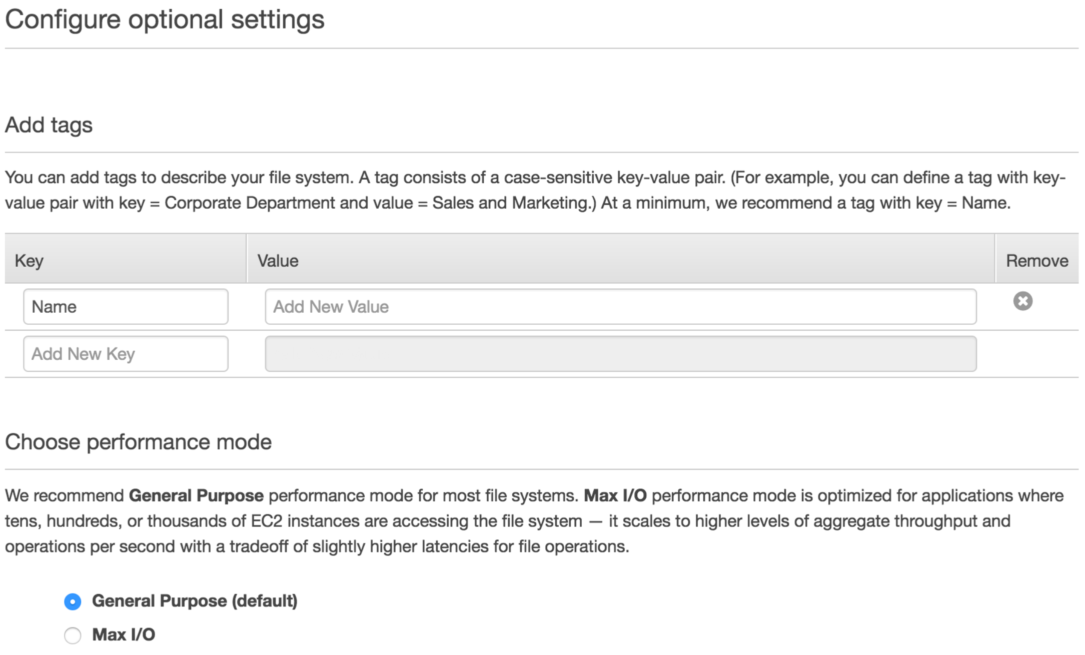
VPC vpc-2cdffc48

Create mount targets

Instances connect to a file system via mount targets you create. We recommend creating a mount target in each of your VPC's Availability Zones so that EC2 instances across your VPC can access the file system.

	Availability Zone	Subnet	IP address	Security group
☑	us-west-2a	subnet-f2aec784	Automatic	sg-766a5010 - default ×

Optional settings can be left blank, or you can add tags to the volume if desired. Click `Next Step` :



Configure optional settings

Add tags

You can add tags to describe your file system. A tag consists of a case-sensitive key-value pair. (For example, you can define a tag with key-value pair with key = Corporate Department and value = Sales and Marketing.) At a minimum, we recommend a tag with key = Name.

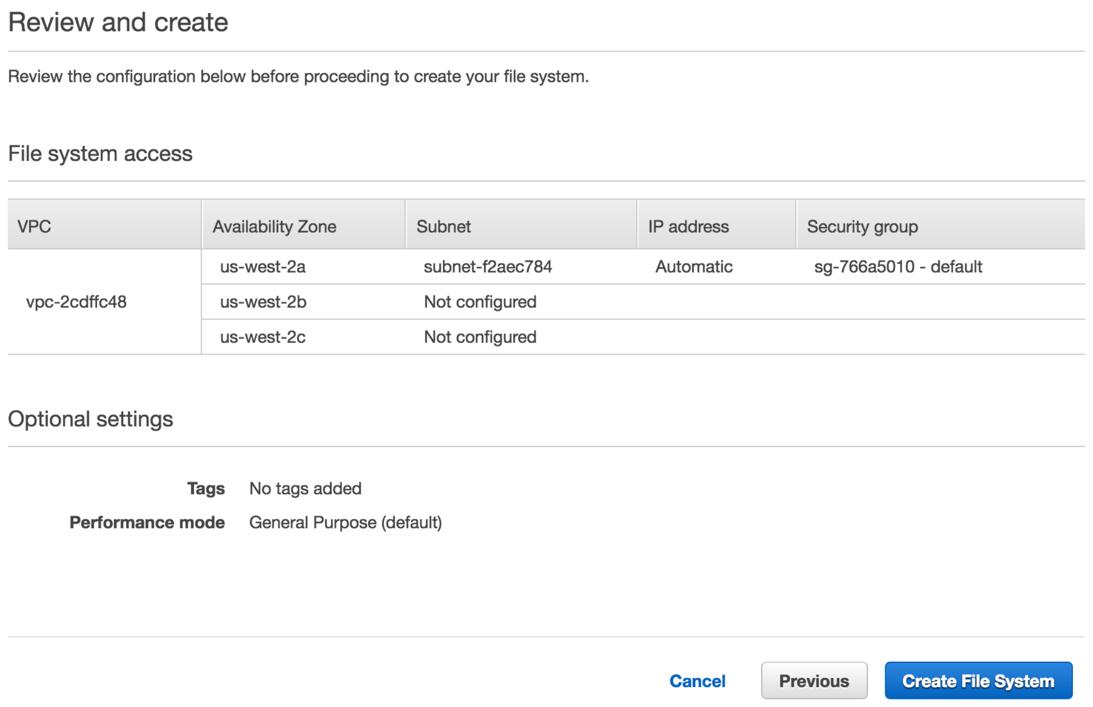
Key	Value	Remove
Name	Add New Value	⊗
Add New Key		

Choose performance mode

We recommend **General Purpose** performance mode for most file systems. **Max I/O** performance mode is optimized for applications where tens, hundreds, or thousands of EC2 instances are accessing the file system — it scales to higher levels of aggregate throughput and operations per second with a tradeoff of slightly higher latencies for file operations.

● **General Purpose (default)**
○ **Max I/O**

You will see a `Review and create` screen. Double check that the appropriate availability zone is selected, then click `Create File System` :



Review and create

Review the configuration below before proceeding to create your file system.

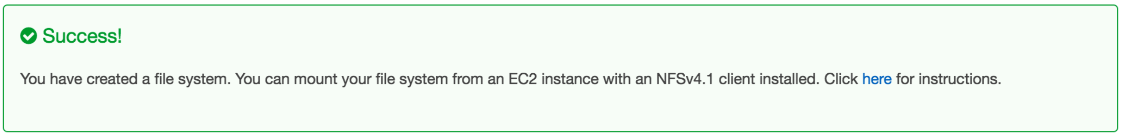
File system access

VPC	Availability Zone	Subnet	IP address	Security group
vpc-2cdffc48	us-west-2a	subnet-f2aec784	Automatic	sg-766a5010 - default
	us-west-2b	Not configured		
	us-west-2c	Not configured		

Optional settings

Tags	No tags added
Performance mode	General Purpose (default)

Once your EFS volume has been created, Amazon provides a link (click `here`) on instructions for mounting on Amazon, Red Hat, and SuSE Linux:



Success!

You have created a file system. You can mount your file system from an EC2 instance with an NFSv4.1 client installed. Click here for instructions.

See below for instructions on mounting an NFS volume on CoreOS.

Mounting an NFS file share on CoreOS

First, get the link to the EFS NFS fileshare you created in the previous step, replacing `xxxxxxxx` with your unique EFS ID:

```
$ echo $(curl -s http://169.254.169.254/latest/meta-data/placement/availability-zone).fs-xxxxxxxx.efs.us-west
```

Next, follow our documentation for mounting NFS volumes to mount your EFS NFS filesystem on each of your DC/OS agents.

Uninstall

To uninstall the Docker registry using the DC/OS CLI, run the following command:

```
$ dcos package uninstall registry
```

© 2017 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub API Training Shop Blog About