DC/OS Universe

DCOS Universe is an Open Source repository that stores the meta-data for DCOS Packages including all of the components that comprise a DCOS service such as Docker Images and HTTP Artifacts. This guide details the process to build, install, and configure a DC/OS local Universe. The official documentation for this can be found in the Mesosphere documentation, under Administration > Installing DC/OS > Deploying a local Universe.

This document assumes a basic understanding of DC/OS, Docker, and Linux. The document details the build and configuration of a DC/OS local Universe in AWS; these steps could be easily adapted for other environments.

Summary

The DC/OS local Universe, when installed with the default configuration, consists of the following:

- A Docker image (mesosphere/universe), modified with the specific set of DC/OS packages specified during the build process.

- A Linux systemd service which starts a Docker container from the mesosphere/universe image as an nginx server, and serves all non-Docker content (HTTP artifacts) necessary for Cosmos and the packages in the Universe, including:
    - A Universe repo zip or json file depending on the version of DCOS
    - Icons and images for the DC/OS Universe UI
    - Other non-Docker resources required for the specified DC/OS packages

  This is served on port 8082 (http) by default on the local Universe server.

- A second Linux systemd service which starts a Docker container from the mesosphere/universe image as a private Docker registry, and serves all Docker images necessary for the specified DC/OS packages. This is served on port 5000 on the local Universe server.

These are the high level steps to get a local Universe up and running:

- Build the local Universe:
    - Stand up a Linux server to build the local Universe
    - Configure and install all of the prereqs on the build server

    - Determine which Universe packages are to be built into the local Universe
    - Determine where the local Universe will be hosted, and customize the Universe built scripts accordingly
    - Build a 'universe-base' Docker image, which serves as the baseline for the Docker image
    - Add all of the specific content to the Docker image
    - Export ('save') the Docker image to a tar file
    - (optional) Gzip the tar file.
- Host the local Universe:
    - Copy the Universe tar archive to the hosting server(s)
    - Load the Universe tar archive into Docker
    - Option1 - Configure and start the two systemd services which run the Universe image with two different commands
    - Option 2 - Run the two Docker Containers inside a DCOS cluster on Marathon to serve the Universe REPO files and the HTTP Artifacts and Docker Images
- Configure the Docker daemon on all agents nodes to trust the private Docker registry

Notes:

This document is based off the official documentation, but supplements it with modifications to perform the following:

- The official documents assume that the local Universe will be hosted on the DC/OS master(s). This document details how to host it externally, which consists of the following modifications:
    - Modify the certificates to match the hostname of hosting server (instead of master.mesos)
    - Modify the repo (zip file) build process to point to the hosting server (instead of master.mesos) for both Docker and HTTP content.
    - **In order to accommodate the above, you must know the hostname for the hosting server prior to beginning the build process.**

Building the Local Universe (Build Server)

Note: this document details one way to build the local Universe on a standard CentOS 7 AMI; similar steps could be followed for other Linux environments, but are not detailed here. Specifically, this document specifies the following versions:

- CentOS 7 (x64)
- Python 3.4
- Docker 1.11.2

The build process does not need the OverlayFS module to be configured.

System Requirements

The local Universe should be built on a Linux machine which meets the following system requirements:

- Standard Linux packages/programs:
  - **openssl**: for generating SSL certs for use by Docker private registry
  - **zip**
  - **make**: to run the build process
  - **git**: to download the core Universe repository (hosted on Github)
- Python requirements:
  - **python34**: Python3 is necessary to run the build scripts; python34 is an easy way to get this up and running (and is included in EPEL, which greatly simplifies installation)
  - **Python34-pip**: primarily used to install jsonschema
  - **jsonschema**: used during the build process to validate that repo files meet the necessary JSON schema requirements
- Docker: for the purposes of this document, we use Docker 1.11.2, which is the same version recommended for DC/OS 1.8.x. Other Docker versions likely work, but the commands may require modification.
  - If you are building a large universe, you may need to increase the Docker daemon base max device size (the default is 10G). Because the Local Universe image includes a Docker registry with all of your Docker images in it, it can be fairly large, so you can increase the max size (detailed at the end of the "Building the Build Server (AWS)" section)

Building the Build Server (AWS)

1. Launch an AWS EC2 instance.
    a. From the EC2 console, choose AWS Marketplace
    b. Search for “CentOS” and choose the “CentOS 7 (x86_64) - with Updates HVM” AMI
    c. Choose an instance type (I usually use t2.xlarge, but any decently-sized one should work)
    d. Ensure that the instance is reachable (via SSH), and can reach the Internet. For example:
        i. VPC which can reach the Internet
        ii. Subnet which has an IGP
        iii. Auto-assign public IP
    e. Add sufficient storage. The more Universe packages you include, the more storage you need (to be expanded). I usually use about 60GB, SSD.
    f. Tag it however you want
    g. In security groups, ensure that SSH is allowed in, at the very least.
    h. Ensure that you have the SSH private key necessary to connect to your instance

2. SSH into your build server

```
ssh -i <path-to-private-key> centos@<public-ip-address>
```

3. Install the Linux packages

```
sudo yum install -y git make zip openssl
```

4. On CentOS, install EPEL (which contains the Python3 packages we’re going to use):

```
sudo yum install -y epel-release
```

5. Download IUS repo *.rpm and and set up IUS Repo

```
curl -LO https://centos7.iuscommunity.org/ius-release.rpm
sudo rpm -ivh ius-release.rpm
```

6. Install Python3.4

```
sudo yum install -y python36u
```

7. Install PIP for Python3.4

```
sudo yum install -y python36u-pip
```

8. Set up symlink for python3:

```
sudo ln -s /bin/python36m /bin/python3
```

9. Configure yum with the Docker repository for CentOS 7

```
sudo tee /etc/yum.repos.d/docker.repo <<-'EOF'
[dockerrepo]
name=Docker Repository
baseurl=https://yum.dockerproject.org/repo/main/centos/7/
enabled=1
gpgcheck=1
gpgkey=https://yum.dockerproject.org/gpg
EOF
```

10. Install Docker 1.11.2

```
sudo yum install -y docker-engine-1.11.2
```

11. Enable the Docker service

```
sudo systemctl enable docker
```

12. Start the Docker daemon

```
sudo systemctl start docker
```

13. (Optional) Modify the Docker daemon base device max size
    a. Locate your docker systemd unit (it may be in /etc/systemd/system/docker.service, or in a file located in /etc/systemd/system/docker.service.d)
    b. Edit the docker systemd unit file, and add this flag to the dockerd ExecStart line:

```
--storage-opt dm.basesize=<desired-size>G
```

    c. For example:

```
[Service]
Restart=always
StartLimitInterval=0
RestartSec=15
ExecStartPre=-/sbin/ip link del docker0
ExecStart=
ExecStart=/usr/bin/dockerd --graph=/var/lib/docker --storage-driver=overlay
--storage-opt dm.basesize=40G
```

    d. Restart the Docker daemon

```
sudo systemctl restart docker
```

Prerequisite Installation Summary

This set of commands will install all of the prereqs indicated above on a CentOS server (aside from the optional Daemon max size configuration):

```
# Install git, make, zip, and openssl
sudo yum install -y git make zip openssl

# Install epel-release repository
sudo yum install -y epel-release

# Install python3.4 and python3.4 pip
sudo yum install -y python34 python34-pip

# Install jsonschema for python3 for root
sudo pip3 install jsonschema

# Configure Docker Repo (should be compatible with both centos and rhel)
sudo tee /etc/yum.repos.d/docker.repo <<-'EOF'
[dockerrepo]
name=Docker Repository
baseurl=https://yum.dockerproject.org/repo/main/centos/7/
enabled=1
gpgcheck=1
gpgkey=https://yum.dockerproject.org/gpg
EOF

# Install Docker engine
sudo yum install -y docker-engine-1.11.2

# Configure Docker service and start Docker
sudo systemctl enable docker
sudo systemctl start docker
```

Clone the Mesosphere Universe repository

Use git to clone the Mesosphere Universe repository to your local system

```
git clone https://github.com/mesosphere/universe.git --branch version-3.x
```

Customize the Universe Scripts

The local Universe build process essentially consist of the following files and scripts:

- **universe/docker/local-universe/Makefile**: Makefile which has the following targets:
    - clean: removes all certificates and generated tarfiles
    - certs: generates the self-signed certificate and key used by the private Docker registry
    - base: builds the 'universe-base' image
    - local-universe: adds specific content to the Docker image, and exports it

- **universe/docker/local-universe/Dockerfile.base**: Dockerfile used to generate the universe-base image ("base" target in Makefile)

- **universe/scripts/local-universe.py**: Python script used to populate the Docker image with Universe packages and content ("local-universe" target in Makefile)

All paths are relative to the directory into which the Mesosphere repository was cloned.

Choosing Packages (recommended)

The packages that are included in the local Universe build are specified in the "local-universe" target of **universe/docker/local-universe/Makefile**. There are three ways to configure this:

- '--selected' flag: this will generate a local Universe image with all packages that are currently specified as "Selected" (see section on "Selected" packages). ***This is the default option***.
- '--include=' flag: this will generate a local Universe image with all packages that are explicitly listed. ***This is the recommended option.***
- no flag: this will generate a local Universe with the latest version of every package in the Mesosphere Universe. ***This is not recommended.***

The recommended path is to modify **universe/docker/local-universe/Makefile**. Specifically, update the "local-universe" target, and replace "--selected" with a specific set of desired packages. For example:

Replace:

```
--selected                                          &&
```

with:

```
--include="marathon,marathon-lb,hello-world"              &&      \
```

"Selected" and Available Packages

- In order to get a list of all available packages, you can run this command:

```
for alpha in $(ls universe/repo/packages);
do ls -1 universe/repo/packages/${alpha}/; done

for alpha in $(ls universe/repo/packages/*);
do ls -1 universe/repo/packages/${alpha}/;
done
```

- By default, all "Selected" packages are included (all packages which have the "selected" value set to **true** in the latest version of **package.json**). For example:
    - universe/repo/packages/M/marathon/8/package.json has "selected":true, so it **marathon** is a "Selected" app (as of version 8)
    - universe/repo/packages/M/marathon-lb/17/package.json does not have a "selected" field, so **marathon-lb** is not a "Selected" app (as of version 17)

    - universe/repo/packages/H/hello-world/5/package.json has "selected":false, so **hello-world** is not a "Selected" app (as of version 5).

- There's no easy way to get a full list of all "Selected" packages. Create a python script called 'selected.py' with this content, run it with the full path to the 'packages' directory, and it should list all "Selected" packages:

```
#!/usr/bin/env python3

import pathlib
import json
import sys

def list_selected_packages(packages_path):
    for letter_path in pathlib.Path(packages_path).iterdir():
        assert len(letter_path.name) == 1 and letter_path.name.isupper()

        for package_path in letter_path.iterdir():
            largest_revision = max(
                package_path.iterdir(),
                key=lambda revision: int(revision.name))

            json_path = largest_revision / 'package.json'
            with json_path.open(encoding='utf-8') as json_file:
                if json.load(json_file).get('selected',False):
                    print(package_path.name)


list_selected_packages(sys.argv[1])
```

For example:

```
$ python3 selected.py ~/universe/repo/packages
arangodb3
artifactory
cassandra
confluent-kafka
dse
elastic
gitlab
jenkins
marathon
spark
```

PGP Server Change (Optional)

The Dockerfile uses pgp.mit.edu as a PGP key server; occasionally, this particular server has load issues, so the server can be replaced with a public pool with this command:

```
cd <universe>/docker/local-universe
sed -i -e 's/pgp.mit.edu/p80.pool.sks-keyservers.net/' Dockerfile.base
```

Certificates and Links (Optional)

*Note: This document uses **ec2-10-10-10-10.us-west-2.compute.amazonaws.com** as the DNS name for the alternative location for the local Universe to be hosted. The DNS name MUST be resolvable by all DC/OS agent nodes. Additionally, in order for icons to show up properly in DC/OS, the DNS name must also be resolvable by the browser used to navigate DC/OS.*

The default installation process generates self-signed certificates and assumes that the local Universe will be hosted on the DC/OS masters (on whatever 'master.mesos' resolves to in DC/OS). In order to change this behavior, several things must happen:

- To host local Universe somewhere other than on the DC/OS master(s), all links within the repo must point to somewhere other than 'master.mesos'. This consists of two lines in **universe/scripts/local-universe.py**:

  ```
  HTTP_ROOT = "http://master.mesos:8082/"
  DOCKER_ROOT = "master.mesos:5000"
  ```

  Replace 'master.mesos' with the DNS name for the local Universe host or load balancer. For example:

  ```
  HTTP_ROOT = "http://ec2-10-10-10-10.us-west-2.compute.amazonaws.com:8082/"
  DOCKER_ROOT = "ec2-10-10-10-10.us-west-2.compute.amazonaws.com:5000"
  ```

- If the local Universe is hosted elsewhere, then the self-signed certificate that is generated by the build process must be generated with the correct hostname. This is controlled by this line in **universe/docker/local-universe/Makefile** (in the "certs" target):

  ```
  -subj "/CN=master.mesos"
  ```

  Replace 'master.mesos' with the DNS name for the local Universe host or load balancer. For example:

  ```
  -subj "/CN=ec2-10-10-10-10.us-west-2.compute.amazonaws.com"
  ```

- If you have your own SSL private key and certificate, these can be injected into the Docker image. This is one simple way to do so, with minimal modification to the build process (overwrites the self-signed certificate and key with your custom certificate and key). (There are many other ways to accomplish this - this method just allows the rest of the Makefile harness to continue to operate transparently).

1. Within the **universe/docker/local-universe** directory, create a directory called **certs_custom** (**universe/docker/local-universe/certs-custom**).
2. Place your SSL certificate at <path>/certs-custom/domain.crt, with 644 permissions (owned by root:root)
3. Place your SSL private key at <path>/certs-custom/domain.key, again with 644 permissions (owned by root:root)
4. Modify **Dockerfile.base** by updating COPY certs with the new certs-custom location:

```
# COPY certs /certs
COPY certs-custom /certs
# COPY certs/domain.crt /usr/share/nginx/html/certs/domain.crt
COPY certs-custom/domain.crt /usr/share/nginx/html/certs/domain.crt
```

Building the local Universe

Once all of the above configuration options have been made, the actual process to build the local Universe is relatively simple.

1. Build the universe-base image. This will create a 'certs' subdirectory, create a new SSL private key and certificate, and then use Dockerfile.base to create a new Docker image tagged 'universe-base'

```
cd universe/docker/local-universe
sudo make base
```

2. Build the local Universe with your desired packages. This will run the local-universe.py script (which loads content into the universe-base image), and then save it to a tar file, which is gzip'd up.

```
sudo make local-universe
```

3. You should now have a local-universe.tar.gz file, which should be backed up externally, and then copied to wherever you are hosting your local Universe (either the DC/OS masters or a separate server):
    - local-universe.tar.gz

4. Additionally, back up and/or store these two files (from the **universe/docker/local-universe** directory):
    - dcos-local-universe-http.service
    - dcos-local-universe-registry.service

Running the Local Universe

The process to deploy a local universe is relatively well documented in the official docs (Administration > Installing DC/OS > Deploying a local Universe). This section of the document summarizes the official documentation, which should be taken as authoritative, and indicates differences between running local Universe on DC/OS masters vs. other Linux servers.

These are the high-level steps to run the local Universe:

1. If local Universe is not going to be hosted on the DC/OS master cluster, configure the Linux servers with the appropriate system requirements.
2. On each of the servers hosting local Universe, load the local-universe.tar.gz file (which contains the single custom mesosphere/universe image into Docker
3. Configure systemd services that will start two instances of the mesosphere/universe Docker image: one to serve as a local Docker private registry, one that will serve as an HTTP server serving all non-Docker content (repository, associated resources)

System Requirements (non-DC/OS masters)

The system requirements to run a DC/OS local universe are a subset of those required to install DC/OS. Specifically, there are four main requirements:

- A Linux server, with sufficient system resources (TBD), that supports systemd services, with SSH access and permissions to run sudo
- Docker daemon
- Open TCP port 8082 from all master and agent nodes in the DC/OS infrastructure, for HTTP
- Open TCP port 5000 from all agent nodes in the DC/OS infrastructure, for Docker private registry

Assuming that you're working with CentOS 7 (same system as examples above), this will setup the Docker repository:

```
sudo tee /etc/yum.repos.d/docker.repo <<-'EOF'
[dockerrepo]
name=Docker Repository
baseurl=https://yum.dockerproject.org/repo/main/centos/7/
enabled=1
gpgcheck=1
gpgkey=https://yum.dockerproject.org/gpg
EOF
```

Then Docker can be installed and run with these commands:

```
sudo yum install -y docker-engine-1.11.2
sudo systemctl enable docker
sudo systemctl start docker
```

Load Balancer System Requirements

In addition to hosting registry external to DC/OS, it may be possible to host multiple instances behind a load balancer. This has not been tested. Specifically, running multiple copies of the same Docker private registry behind a load balancer may have unexpected issues.

If a load balancer is to be used in front of multiple local Universe servers, then the certificate and all links in the Certificates and Links (Optional) section should have the DNS name for the load balancer, rather than the DNS name for a specific local Universe.

In addition, the following ports should be load balanced:

- 5000: private Docker registry
- 8082: nginx http port

Installing and Running Local Universe

Once a system has been set up with the necessary prerequisites, the following steps should be followed to install and run a local Universe (these same steps apply to DC/OS masters, if the local Universe is run from the master cluster).

1. Either from your local Universe build server, or from wherever your three files (local-universe.tar.gz, dcos-local-universe-http.service, and dcos-local-universe-registry.service) are served, transfer (via SCP or other) them to your local Universe server (in this example, to the home directory (~)):

```
scp local-universe.tar.gz <user>@<hostname>:~
scp dcos-local-universe-http.service <user>@<hostname>:~
scp dcos-local-universe-registry.service <user>@<hostname>:~
```

2. SSH into your local Universe server:

```
ssh <user>@<hostname>
```

3. On the local Universe server, copy the systemd service files to the systemd service directory:

```
sudo cp dcos-local-universe-http.service /etc/systemd/system/
sudo cp dcos-local-universe-registry.service /etc/systemd/system/
```

4. Load the custom mesosphere/universe Docker image into the local Docker instance:

```
sudo sh -c "docker load < local-universe.tar.gz"
```

5. Reload the systemd daemon, and enable and start the two dcos-local-universe services:

```
sudo systemctl daemon-reload
sudo systemctl enable dcos-local-universe-http
sudo systemctl enable dcos-local-universe-registry
sudo systemctl start dcos-local-universe-http
sudo systemctl start dcos-local-universe-registry
```

6. Verify that two Docker services are running, one for the http instance and one for the registry instance:

```
sudo docker ps
```

All of the above steps should be completed for every local Universe server (either all of the DC/OS master nodes, or every external local Universe server).

Using / Connecting to the Local Universe

In order to actually use the newly-setup local Universe, two additional steps must be taken:

- Configure the DC/OS master cluster to connect to the new local Universe, so that the packages in the local Universe can be utilized
- Configure all DC/OS agents to trust the private Docker registry in the local Universe, so that the agents can run Docker images from the private Docker registry.

This section of the document details both steps.

Configuring DC/OS Master Cluster with the new Local Universe

1. Ensure that you have the DC/OS command line installed and configured to connect to your local Universe (instructions for this are available in the official Mesosphere docs, at Usage > CLI > Installing the CLI).

2. From the DC/OS command line, run this command (replacing "<hostname>" with the DNS name for your local Universe, and "<universe-name>" with the name you want to show up in the DC/OS UI

dcos package repo add <universe-name> http://<hostname>:8082/repo

3. In the DC/OS UI, your new Local Universe should show up in the "Package Repositories" page (System > Settings > Package Repositories)

4. Optionally, remove the default Universe with this command:

dcos package repo remove Universe

5. (You can also re-add the default Universe with this command):

dcos package repo add --index=0 Universe http://universe.mesosphere.com/repo

Configuring DC/OS Agents to trust Local Universe Docker Registry

For each of the DC/OS agent nodes in your DC/OS cluster, you must complete the following steps to configure them to trust the Local Universe Docker Registry (and allow them to run Docker images from the Local Universe Docker Registry).

1. Make an **/etc/docker/certs.d/** directory and a nested **/etc/docker/certs.d/hostname:port/** directory corresponding to the host and port of the local Universe Docker registry:

   ```
   sudo mkdir -p /etc/docker/certs.d/<hostname>:<port>
   ```

   For example, using a local Universe hosted on ec2-10-10-10-10.us-west-2.compute.amazonaws.com

   ```
   sudo mkdir -p
   /etc/docker/certs.d/ec2-10-10-10-10.us-west-2.compute.amazonaws.com:5000
   ```

2. Obtain the certificate used by the local Universe Docker registry and place it in the new directory (with the filename "ca.crt") (note that this is obtaining the certificate from the HTTP nginx server on port 8082 where it is served, but it's being used for the Docker registry on port 5000):

   ```
   curl -O http://<hostname>:8082/certs/domain.crt
   sudo cp domain.crt /etc/docker/certs.d/<hostname>:5000/ca.crt
   ```

   For example, using a local Universe hosted on ec2-10-10-10-10.us-west-2.compute.amazonaws.com

   ```
   curl -O http://ec2-10-10-10-10.us-west-2.compute.amazonaws.com:8082/domain.crt
   sudo cp domain.crt
   /etc/docker/certs.d/ec2-10-10-10-10.us-west-2.compute.amazonaws.com:5000/ca.crt
   ```

3. Restart the Docker daemon (**this will restart all Docker instances on the agent node**):

   ```
   sudo systemctl restart docker
   ```