Docker Explained: How To Create Docker Containers Running Memcached

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: *http://faq.asphosthelpdesk.com/article.php?id=259*

Docker in Brief

The **docker project** offers higher-level tools, working together, which are built on top of some Linux kernel features. The goal is to help developers and system administrators port applications - with all of their dependencies conjointly - and get them running across systems and machines -*headache free*.

Docker achieves this by creating safe, LXC (i.e. Linux Containers) based environments for applications called docker containers. These containers are created using docker images, which can be built either by executing commands manually or automatically through Dockerfiles.

Memcached in Brief

Memcached is a distributed, open-source data storage engine. It was designed to store certain types of data in RAM (instead of slower rate traditional disks) for very fast retrievals by applications, cutting the amount of time it takes to process requests by reducing the number of queries performed against heavier datasets or APIs such as traditional databases (e.g. MySQL).

By introducing a smart, well-planned, and optimized caching mechanism, it becomes possible to handle a seemingly larger amount of requests and perform more procedures by applications. This is the most important use case of Memcached, as it is with any other caching application or component.

Heavily relied upon and used in production for web sites and various other applications, Memcached has become one of the go-to tools for increasing performance without -necessarily - needing to utilize further hardware (e.g. more servers or server resources).

It works by storing *keys* and their matching *values* (up to 1 MB in size) onto an associative array (i.e. hash table) which can be scaled and distributed across a large number of virtual servers.

Installing Docker on Ubuntu (Latest)

To start using the Docker project on your server, you can either use DigitalOcean's docker image for Ubuntu 13.04 or install it yourself. In this section, we will quickly go over the basic installation instructions for Docker 0.7.1.

Installation Instructions for Ubuntu

Update:

```
sudo aptitude    update
sudo aptitude -y upgrade
```

Make sure aufs support is available:

```
sudo aptitude install linux-image-extra-`uname -r`
```

Add docker repository key to apt-key for package verification:

```
sudo sh -c "wget -qO- https://get.docker.io/gpg | apt-key add -"
```

Add the docker repository to aptitude sources:

```
sudo sh -c "echo deb http://get.docker.io/ubuntu docker main\
> /etc/apt/sources.list.d/docker.list"
```

Update the repository with the new addition:

```
sudo aptitude    update
```

Finally, download and install docker:

```
sudo aptitude install lxc-docker
```

Ubuntu's default firewall (**UFW**: Uncomplicated Firewall) denies all forwarding traffic by default, which is needed by docker.

Enable forwarding with UFW:

Edit UFW configuration using the nano text editor.

```
sudo nano /etc/default/ufw
```

Scroll down and find the line beginning with **DEFAULT_FORWARD_POLICY**.

Replace:

```
DEFAULT_FORWARD_POLICY="DROP"
```

With:

```
DEFAULT_FORWARD_POLICY="ACCEPT"
```

Press **CTRL+X** and approve with **Y** to save and close.

Finally, reload the UFW:

```
sudo ufw reload
```

Basic Docker Commands

Running the docker daemon and CLI Usage

Upon installation, the docker daemon should be running in the background, ready to accept commands sent by the docker CLI. For certain situations where it might be necessary to manually run docker, use the following.

Running the docker daemon:

```
sudo docker -d &
```

docker CLI Usage:

```
sudo docker [option] [command] [arguments]
```

Note: docker needs sudo privileges in order to work.

Commands List

Here is a summary of currently available (version 0.7.1) docker commands:

attach

Attach to a running container

build

Build a container from a Dockerfile

commit

Create a new image from a container's changes

cp

Copy files/folders from the containers filesystem to the host path

diff

Inspect changes on a container's filesystem

events

Get real time events from the server

export

Stream the contents of a container as a tar archive

history

Show the history of an image

images

List images

import

Create a new filesystem image from the contents of a tarball

info

Display system-wide information

insert

Insert a file in an image

inspect

Return low-level information on a container

kill

Kill a running container

load

Load an image from a tar archive

login

Register or Login to the docker registry server

logs

Fetch the logs of a container

port

Lookup the public-facing port which is NAT-ed to PRIVATE_PORT

ps

List containers

pull

Pull an image or a repository from the docker registry server

push

Push an image or a repository to the docker registry server

restart

Restart a running container

rm

Remove one or more containers

rmi

Remove one or more images

run

Run a command in a new container

save

Save an image to a tar archive

search

Search for an image in the docker index

start

Start a stopped container

stop

Stop a running container

tag

Tag an image into a repository

top

Lookup the running processes of a container

version

Show the docker version information

Getting Started with Creating Memcached Images

Building on our knowledge gained from the previous articles in the docker series, let's dive straight into building a Dockerfile to have docker automatically build Memcached installed images (which will be used to run sandboxed Memcached instances).

Quick Recap: What Are Dockerfiles?

Dockerfiles are scripts containing commands declared successively which are to be executed, in the order given, by docker to automatically create a new docker image. They help greatly with deployments.

These files always begin with the definition of a base image by using the FROM command. From there on, the*build process*starts and each following action taken forms the final with commits (saving the image state) on the host.

Usage:

```
# Build an image using the Dockerfile at current location
# Tag the final image with [name] (e.g. *nginx*)
# Example: sudo docker build -t [name] .
sudo docker build -t memcached_img .
```

Dockerfile Commands Overview

Add

Copy a file from the host into the container

CMD

Set default commands to be executed, or passed to the ENTRYPOINT

ENTRYPOINT

Set the default entrypoint application inside the container

ENV

Set environment variable (e.g. "key = value")

EXPOSE

Expose a port to outside

FROM

Set the base image to use

MAINTAINER

Set the author / owner data of the Dockerfile

RUN

Run a command and commit the ending result (container) image

USER

Set the user to run the containers from the image

VOLUME

Mount a directory from the host to the container

WORKDIR

Set the directory for the directives of CMD to be executed

Creating a Dockerfile

Since Dockerfiles constitute of plain-text documents, creating one translates to launching your favourite text editor and writing the commands you want docker to execute in order to build an image. After you start working on the file, continue with adding all the content below (one after the other) before saving the final result.

Note: You can find what the final Dockerfile will look like at the end of this section.

Let's create an empty Dockerfile using nano text editor:

```
nano Dockerfile
```

We need to have all instructions (commands) and directives listed successively. However, everything starts with building on a base image (set with the FROM command).

Let's define the purpose of our Dockerfile and declare the base image to use:

```
############################################################
# Dockerfile to run Memcached Containers
# Based on Ubuntu Image
############################################################
# Set the base image to use to Ubuntu
FROM ubuntu
# Set the file maintainer (your name - the file's author)
MAINTAINER Maintaner Name
```

After this initial block of commands and declarations, we can begin with listing the instructions for Memcached installation.

```
# Update the default application repository sources list
RUN apt-get update
```

```
# Install Memcached
RUN apt-get install -y memcached
```

Set the default port to be exposed to outside the container:

```
# Port to expose (default: 11211)
EXPOSE 11211
```

Set the default execution command and entrpoint (i.e. Memcached daemon):

```
# Default Memcached run command arguments
CMD ["-u", "root", "-m", "128"]
# Set the user to run Memcached daemon
USER daemon
# Set the entrypoint to memcached binary
ENTRYPOINT memcached
```

Final Dockerfile

```
###############################################################
# Dockerfile to run Memcached Containers
# Based on Ubuntu Image
###############################################################
# Set the base image to use to Ubuntu
FROM ubuntu
# Set the file maintainer (your name - the file's author)
MAINTAINER Maintaner Name
# Update the default application repository sources list
RUN apt-get update
# Install Memcached
RUN apt-get install -y memcached
# Port to expose (default: 11211)
EXPOSE 11211
# Default Memcached run command arguments
CMD ["-m", "128"]
# Set the user to run Memcached daemon
USER daemon
# Set the entrypoint to memcached binary
ENTRYPOINT memcached
```

After having everything written inside the Dockerfile, save it and exit by pressing CTRL+X followed by Y.

Using this Dockerfile, we are ready to get started with dockerised Memcached containers!

Creating the Docker Image for Memcached Containers

We can now create our first Memcached image by following the usage instructions explained in the Dockerfile Basics section.

Run the following command to create an image, tagged as "memcached_img":

```
sudo docker build -t memcached_img .
```

Note: Do not forget the trailing`.`for docker to find the`Dockerfile`.

Running dockerised Memcached Containers

It is very simple to create any number of perfectly isolated and self-contained memcached instances -**now**- thanks to the image we have obtained in the previous section. All we have to do is to create a new container with `docker run`.

Creating a Memcached Installed Container

To create a new container, use the following command, modifying it to suit your requirements following this example:

```
# Example: sudo docker run -name [container name] -p [port to access:port
exposed] -i -t [memcached image name]
sudo docker run -name memcached_ins -d -p 45001:11211 memcached_img
```

Now we will have a docker container named "memcached_ins", accessible from port **45001**,`run`using our image tagged "memcached_img", which we built previously.

Limiting the Memory for a Memcached Container

In order to limit the amount of memory a docker container process can use, simply set the`-m [memory amount]`flag with the limit.

To run a container with memory limited to 256 MBs:

```
# Example: sudo docker run -name [name] -m [Memory (int)][memory unit (b, k, m
or g)] -d (to run not to attach) -p (to set access and expose ports) [image ID]
sudo docker run -name memcached_ins -m 256m -d -p 45001:11211 memcached_img
```

To confirm the memory limit, you can inspect the container:

```
# Example: docker inspect [container ID] | grep Memory
sudo docker inspect memcached_ins | grep Memory
```

Note: The command above will grab the memory related information from the inspection output. To see all the relevant information regarding your container, opt for`sudo docker inspect [container ID]`.

Testing the Memcached Container

There are various ways to try your newly created Memcached running container(s). We will use a simple

Python CLI application for this. However, you can just get to production with your application using caching add-ons, frameworks, or libraries.

Make sure that your host has the necessary libraries for Python / Memcached:

```
sudo apt-get update && sudo apt-get -y upgrade
sudo apt-get install -y python-pip
pip install python-memcached
```

Let's create a simple Python script called "mc.py" using nano:

```
nano cache.py
```

Copy-and-paste the below (self-explanatory) content inside:

```
# Import python-memcache and sys for arguments
import memcache
import sys
# Set address to access the Memcached instance
addr = 'localhost'
# Get number of arguments
# Expected format: python cache.py [memcached port] [key] [value]
len_argv = len(sys.argv)
# At least the port number and a key must be supplied
if len_argv < 3:
    sys.exit("Not enough arguments.")
# Port is supplied and a key is supplied - let's connect!
port  = sys.argv[1]
cache = memcache.Client(["{0}:{1}".format(addr, port)])
# Get the key
key   = str(sys.argv[2])
# If a value is also supplied, set the key-value pair
if len_argv == 4:
    value = str(sys.argv[3])
    cache.set(key, value)
    print "Value for {0} set!".format(key)
# If a value is not supplied, return the value for the key
else:
    value = cache.get(key)
    print "Value for {0} is {1}.".format(key, value)
```

Press CTRL+X and approve with Y to save and close.

Testing a docker memcached instance using the script above from your host:

```
# Example: python cache.py [port] [key] [value]
python cache.py 45001 my_test_key test_value
# Return: Value for my_test_key set
# See if the key is set:
python cache.py 45001 my_test_key
# Return: Value for my_test_key is test_value.
```