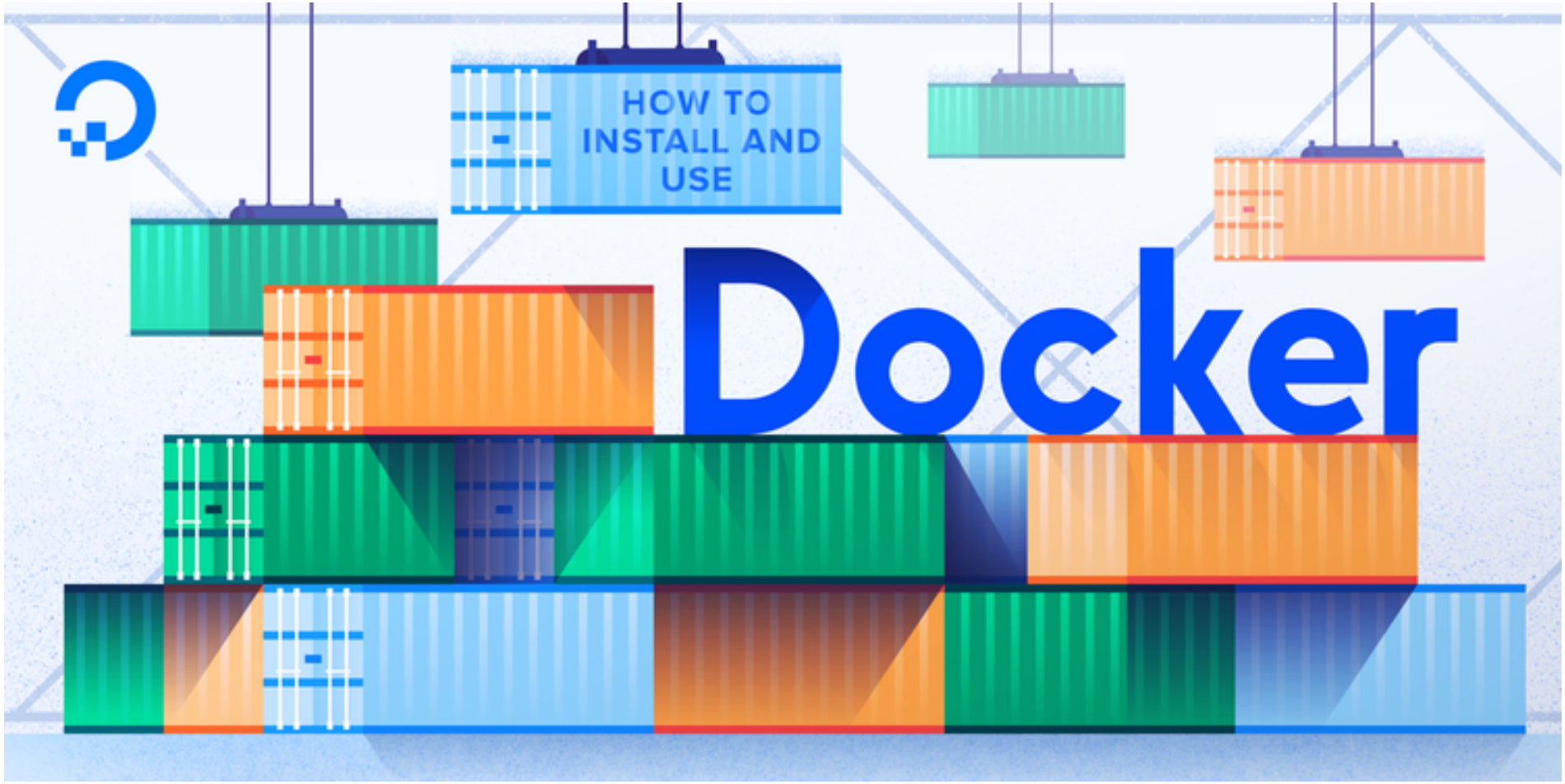
By: finid

Subscribe Share Contents



How To Install and Use Docker on Ubuntu 16.04

Updated November 3, 2016 418.2k DOCKER UBUNTU UBUNTU 16.04

SCROLL TO TOP

Introduction

Docker is an application that makes it simple and easy to run application processes in a container, which are like virtual machines, only more portable, more resource-friendly, and more dependent on the host operating system. For a detailed introduction to the different components of a Docker container, check out The Docker Ecosystem: An Introduction to Common Components.

There are two methods for installing Docker on Ubuntu 16.04. One method involves installing it on an existing installation of the operating system. The other involves spinning up a server with a tool called Docker Machine that auto-installs Docker on it.

In this tutorial, you'll learn how to install and use it on an existing installation of Ubuntu 16.04.

Prerequisites

To follow this tutorial, you will need the following:

- 64-bit Ubuntu 16.04 Droplet
- Non-root user with sudo privileges Initial Setup Guide for Ubuntu 16.04 explains how to set this up.)

> **Note:** Docker requires a 64-bit version of Ubuntu as well as a kernel version equal to or greater than 3.10. The default 64-bit Ubuntu 16.04 Droplet meets these requirements.

All the commands in this tutorial should be run as a non-root user. If root access is required for the command, it will be preceded by `sudo`. Initial Setup Guide for Ubuntu 16.04 explains how to add users and give them sudo access.

Step 1 — Installing Docker

The Docker installation package available in the official Ubuntu 16.04 repository may not be the latest version. To get the latest and greatest version, install Docker from the official Docker repository. This section shows you how to do just that.

But first, let's update the package database:

```
$ sudo apt-get update
```

Now let's install Docker. Add the GPG key for the official Docker repository to the system:

```
$ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
```

Add the Docker repository to APT sources:

```
$ sudo apt-add-repository 'deb https://apt.dockerproject.org/repo ubuntu-xenial main'
```

Update the package database with the Docker packages from the newly added repo:

```
$ sudo apt-get update
```

Make sure you are about to install from the Docker repo instead of the default Ubuntu 16.04 repo:

```
$ apt-cache policy docker-engine
```

You should see output similar to the follow:

Output of apt-cache policy docker-engine

```
docker-engine:
  Installed: (none)
  Candidate: 1.11.1-0~xenial
  Version table:
     1.11.1-0~xenial 500
        500 https://apt.dockerproject.org/repo ubuntu-xenial/main amd64 Packages
     1.11.0-0~xenial 500
```

```
        500 https://apt.dockerproject.org/repo ubuntu-xenial/main amd64 Packages
```

Notice that `docker-engine` is not installed, but the candidate for installation is from the Docker repository for Ubuntu 16.04. The `docker-engine` version number might be different.

Finally, install Docker:

```
$ sudo apt-get install -y docker-engine
```

Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it's running:

```
$ sudo systemctl status docker
```

The output should be similar to the following, showing that the service is active and running:

```
Output
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2016-05-01 06:53:52 CDT; 1 weeks 3 days ago
     Docs: https://docs.docker.com
 Main PID: 749 (docker)
```

Installing Docker now gives you not just the Docker service (daemon) but also the `docker` command line utility, or the Docker client. We'll explore how to use the `docker` command later in this tutorial.

Step 2 — Executing the Docker Command Without Sudo (Optional)

By default, running the `docker` command requires root privileges — that is, you have to prefix the command with `sudo`. It can also be ru
docker group, which is automatically created during the installation of Docker. If you attempt to run the `docker` command without prefixing it with `sudo` or

without being in the docker group, you'll get an output like this:

```
Output
docker: Cannot connect to the Docker daemon. Is the docker daemon running on this host?.
See 'docker run --help'.
```

If you want to avoid typing `sudo` whenever you run the `docker` command, add your username to the docker group:

```
$ sudo usermod -aG docker $(whoami)
```

You will need to log out of the Droplet and back in as the same user to enable this change.

If you need to add a user to the `docker` group that you're not logged in as, declare that username explicitly using:

```
$ sudo usermod -aG docker username
```

The rest of this article assumes you are running the `docker` command as a user in the docker user group. If you choose not to, please prepend the commands with `sudo`.

Step 3 — Using the Docker Command

With Docker installed and working, now's the time to become familiar with the command line utility. Using `docker` consists of passing it a chain of options and commands followed by arguments. The syntax takes this form:

```
$ docker [option] [command] [arguments]
```

To view all available subcommands, type:


```
$ docker
```

As of Docker 1.11.1, the complete list of available subcommands includes:

Output

```
    attach    Attach to a running container
    build     Build an image from a Dockerfile
    commit    Create a new image from a container's changes
    cp        Copy files/folders between a container and the local filesystem
    create    Create a new container
    diff      Inspect changes on a container's filesystem
    events    Get real time events from the server
    exec      Run a command in a running container
    export    Export a container's filesystem as a tar archive
    history   Show the history of an image
    images    List images
    import    Import the contents from a tarball to create a filesystem image
    info      Display system-wide information
    inspect   Return low-level information on a container or image
    kill      Kill a running container
    load      Load an image from a tar archive or STDIN
    login     Log in to a Docker registry
    logout    Log out from a Docker registry
    logs      Fetch the logs of a container
    network   Manage Docker networks
    pause     Pause all processes within a container
    port      List port mappings or a specific mapping for the CONTAINER
    ps        List containers
    pull      Pull an image or a repository from a registry
    push      Push an image or a repository to a registry
    rename    Rename a container
    restart   Restart a container
    rm        Remove one or more containers
    rmi       Remove one or more images
    run       Run a command in a new container
    save      Save one or more images to a tar archive
```

```
    search    Search the Docker Hub for images
    start     Start one or more stopped containers
    stats     Display a live stream of container(s) resource usage statistics
    stop      Stop a running container
    tag       Tag an image into a repository
    top       Display the running processes of a container
    unpause   Unpause all processes within a container
    update    Update configuration of one or more containers
    version   Show the Docker version information
    volume    Manage Docker volumes
    wait      Block until a container stops, then print its exit code
```

To view the switches available to a specific command, type:

```
$ docker docker-subcommand --help
```

To view system-wide information about Docker, use:

```
$ docker info
```

Step 4 — Working with Docker Images

Docker containers are run from Docker images. By default, it pulls these images from Docker Hub, a Docker registry managed by Docker, the company behind the Docker project. Anybody can build and host their Docker images on Docker Hub, so most applications and Linux distributions you'll need to run Docker containers have images that are hosted on Docker Hub.

To check whether you can access and download images from Docker Hub, type:

```
$ docker run hello-world
```


The output, which should include the following, should indicate that Docker in working correctly:

```
Output
Hello from Docker.
This message shows that your installation appears to be working correctly.
...
```

You can search for images available on Docker Hub by using the `docker` command with the `search` subcommand. For example, to search for the Ubuntu image, type:

```
$ docker search ubuntu
```

The script will crawl Docker Hub and return a listing of all images whose name match the search string. In this case, the output will be similar to this:

```
Output

NAME                            DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
ubuntu                          Ubuntu is a Debian-based Linux operating s...   3808      [OK]
ubuntu-upstart                  Upstart is an event-based replacement for ...   61        [OK]
torusware/speedus-ubuntu        Always updated official Ubuntu docker imag...   25                   [OK]
rastasheep/ubuntu-sshd          Dockerized SSH service, built on top of of...   24                   [OK]
ubuntu-debootstrap              debootstrap --variant=minbase --components...   23        [OK]
nickistre/ubuntu-lamp           LAMP server on Ubuntu                           6                    [OK]
nickistre/ubuntu-lamp-wordpress LAMP on Ubuntu with wp-cli installed            5                    [OK]
nuagebec/ubuntu                 Simple always updated Ubuntu docker images...   4                    [OK]
nimmis/ubuntu                   This is a docker images different LTS vers...   4                    [OK]
maxexcloo/ubuntu                Docker base image built on Ubuntu with Sup...   2                    [OK]
admiringworm/ubuntu             Base ubuntu images based on the official u...   1                    [OK]

...
```

In the **OFFICIAL** column, **OK** indicates an image built and supported by the company behind the project. Once you've identified the ima like to use, you can download it to your computer using the `pull` subcommand, like so:

```
$ docker pull ubuntu
```

After an image has been downloaded, you may then run a container using the downloaded image with the `run` subcommand. If an image has not been downloaded when `docker` is executed with the `run` subcommand, the Docker client will first download the image, then run a container using it:

```
$ docker run ubuntu
```

To see the images that have been downloaded to your computer, type:

```
$ docker images
```

The output should look similar to the following:

```
Output
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              c5f1cf30c96b        7 days ago          120.8 MB
hello-world         latest              94df4f0ce8a4        2 weeks ago         967 B
```

As you'll see later in this tutorial, images that you use to run containers can be modified and used to generate new images, which may then be uploaded (*pushed* is the technical term) to Docker Hub or other Docker registries.

Step 5 — Running a Docker Container

The `hello-world` container you ran in the previous is an example of a container that runs and exits, after emitting a test message. Containers, however, can be much more useful than that, and they can be interactive. After all, they are similar to virtual machines, only more resource-friendly.

As an example, let's run a container using the latest image of Ubuntu. The combination of the **-i** and **-t** switches gives you interactive shell access into the container:

```
$ docker run -it ubuntu
```

Your command prompt should change to reflect the fact that you're now working inside the container and should take this form:

```
Output
root@d9b100f2f636:/#
```

Important: Note the container id in the command prompt. In the above example, it is `d9b100f2f636`.

Now you may run any command inside the container. For example, let's update the package database inside the container. No need to prefix any command with `sudo`, because you're operating inside the container with root privileges:

```
$ apt-get update
```

Then install any application in it. Let's install NodeJS, for example.

```
$ apt-get install -y nodejs
```

Step 6 — Committing Changes in a Container to a Docker Image

When you start up a Docker image, you can create, modify, and delete files just like you can with a virtual machine. The changes that you make will only apply to that container. You can start and stop it, but once you destroy it with the `docker rm` command, the changes will be lost for good.

This section shows you how to save the state of a container as a new Docker image.

After installing nodejs inside the Ubuntu container, you now have a container running off an image, but the container is different from the image you used to create it.

To save the state of the container as a new image, first exit from it:

```
$ exit
```

Then commit the changes to a new Docker image instance using the following command. The **-m** switch is for the commit message that helps you and others know what changes you made, while **-a** is used to specify the author. The container ID is the one you noted earlier in the tutorial when you started the interactive docker session. Unless you created additional repositories on Docker Hub, the repository is usually your Docker Hub username:

```
$ docker commit -m "What did you do to the image" -a "Author Name" container-id repository/new_image_name
```

For example:

```
$ docker commit -m "added node.js" -a "Sunday Ogwu-Chinuwa" d9b100f2f636 finid/ubuntu-nodejs
```

> **Note:** When you *commit* an image, the new image is saved locally, that is, on your computer. Later in this tutorial, you'll learn how to push an image to a Docker registry like Docker Hub so that it may be assessed and used by you and others.

After that operation has completed, listing the Docker images now on your computer should show the new image, as well as the old one that it was derived from:

```
$ docker images
```

The output should be similar to this:

```
Output
finid/ubuntu-nodejs       latest              62359544c9ba        50 seconds ago      206.6 MB
ubuntu              latest              c5f1cf30c96b        7 days ago          120.8 MB
hello-world         latest              94df4f0ce8a4        2 weeks ago         967 B
```

In the above example, **ubuntu-nodejs** is the new image, which was derived from the existing ubuntu image from Docker Hub. The size difference reflects the changes that were made. And in this example, the change was that NodeJS was installed. So next time you need to run a container using Ubuntu with NodeJS pre-installed, you can just use the new image. Images may also be built from what's called a Dockerfile. But that's a very involved process that's well outside the scope of this article.

Step 7 — Listing Docker Containers

After using Docker for a while, you'll have many active (running) and inactive containers on your computer. To view the **active ones**, use:

```
$ docker ps
```

You will see output similar to the following:

```
Output
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
f7c79cc556dd        ubuntu              "/bin/bash"         3 hours ago         Up 3 hours                              silly_spence
```

To view all containers — active and inactive, pass it the `-a` switch:

```
$ docker ps -a
```

To view the latest container you created, pass it the `-l` switch:

```
$ docker ps -l
```

Stopping a running or active container is as simple as typing:

```
$ docker stop container-id
```

The `container-id` can be found in the output from the `docker ps` command.

Step 8 — Pushing Docker Images to a Docker Repository

The next logical step after creating a new image from an existing image is to share it with a select few of your friends, the whole world on Docker Hub, or other Docker registry that you have access to. To push an image to Docker Hub or any other Docker registry, you must have an account there.

This section shows you how to push a Docker image to Docker Hub. To learn how to create your own private Docker registry, check out How To Set Up a Private Docker Registry on Ubuntu 14.04.

To create an account on Docker Hub, register at Docker Hub. Afterwards, to push your image, first log into Docker Hub. You'll be prompted to authenticate:

```
$ docker login -u docker-registry-username
```

If you specified the correct password, authentication should succeed. Then you may push your own image using:

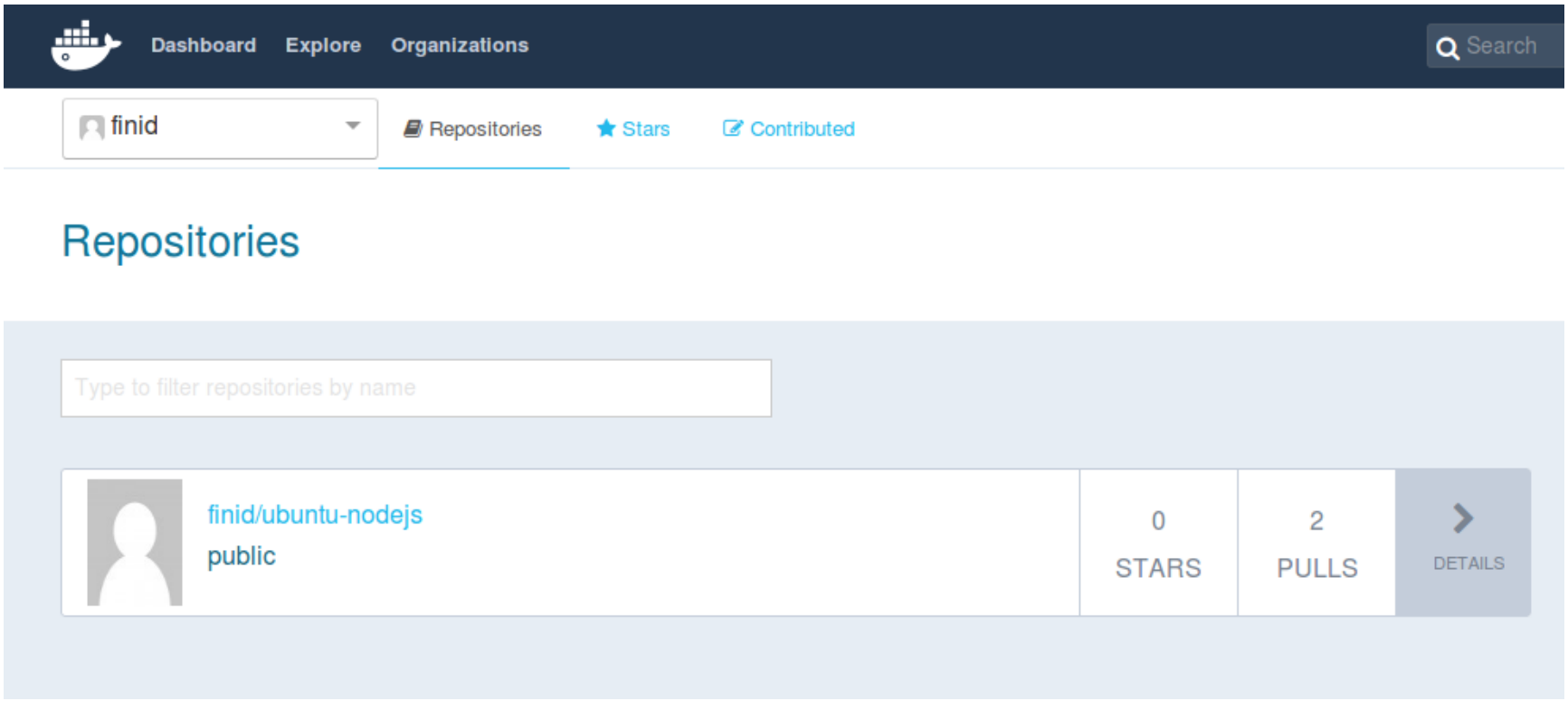
```
$ docker push docker-registry-username/docker-image-name
```

It will take sometime to complete, and when completed, the output will similar to the following:

```
Output
The push refers to a repository [docker.io/finid/ubuntu-nodejs]
e3fbbfb44187: Pushed
5f70bf18a086: Pushed
a3b5c80a4eba: Pushed
7f18b442972b: Pushed
3ce512daaf78: Pushed
7aae4540b42d: Pushed

...
```

After pushing an image to a registry, it should be listed on your account's dashboard, like that show in the image below.



If a push attempt results in an error of this sort, then you likely did not log in:

```
Output
The push refers to a repository [docker.io/finid/ubuntu-nodejs]
e3fbbfb44187: Preparing
5f70bf18a086: Preparing
a3b5c80a4eba: Preparing
7f18b442972b: Preparing
3ce512daaf78: Preparing
7aae4540b42d: Waiting
unauthorized: authentication required
```


Log in, then repeat the push attempt.

Conclusion

There's a whole lot more to Docker than has been given in this article, but this should be enough to getting you started working with it on Ubuntu 16.04. Like most open source projects, Docker is built from a fast-developing codebase, so make a habit of visiting the project's blog page for the latest information.

Also check out the other Docker tutorials in the DO Community.

By: finid

Editor:
Tammy Fox

Related Tutorials

How To Remove Docker Images, Containers, and Volumes

Working with Docker Containers

Naming Docker Containers: 3 Tips for Beginners

The Docker Ecosystem: Scheduling and Orchestration

The Docker Ecosystem: An Overview of Containerization

20 Comments

Leave a comment...

Log In to Comment

zetadisseny *May 23, 2016*

0 I only can say that: I get at your feet. Excellent tutorial

panousis *May 29, 2016*

0 Thank you very much for your article. It's really excellent!. Please keep going

SCROLL TO TOP

oded *June 2, 2016*

3 Its probably better to use the Ubuntu APT tools to update the APT configuration. Instead of `echo`, use `apt-add-repository` to manipulate the apt sources:

```
sudo apt-add-repository 'deb https://apt.dockerproject.org/repo ubuntu-xenial main'
```

MelissaAnderson MOD *November 3, 2016*

0 Thanks for taking time to point that out. I've made the update.

iambobby *January 13, 2017*

0 If you choose to use apt-add-repository you need to make sure software-properties-common is installed...

davemanginelli *June 17, 2016*

1 I had some trouble with docker failing to run after some updates to a 16.04 droplet. The solution was to install a dependency not mentioned above but recommended in Docker's instructions for 16.04:

sudo apt-get install linux-image-extra-$(uname -r)

After which I was able to get Docker running again.

dekkermichelle1 *June 22, 2016*

0 Hi,

I have followed the tutorial, but I get this message

_*docker*machine_ps1: command not found

tapeason *July 22, 2016*

0 Great tutorial, easy to follow, and no problems along the way. Thank you!

florianfalk *August 1, 2016*

0

Well, it's not working for me. After I try running a ubuntu container, I get the following Error:

docker: failed to register layer: devmapper: Error mounting '/dev/mapper/docker-8:2-526216-d9cc121a2c92ab980cbf1218ba4f467d6cc27248993b1a79165e7602113d88b0' on '/var/lib/docker/devicemapper/mnt/d9cc121a2c92ab980cbf1218ba4f467d6cc27248993b1a79165e7602113d88b0': invalid argument.

Have anybody a solution?

florianfalk *August 1, 2016*

0

Well it doesn't work for me. It get this error when I try to run a container:

docker: failed to register layer: devmapper: Error mounting '/dev/mapper/docker-8:2-526216-d9cc121a2c92ab980cbf1218ba4f467d6cc27248993b1a79165e7602113d88b0' on '/var/lib/docker/devicemapper/mnt/d9cc121a2c92ab980cbf1218ba4f467d6cc27248993b1a79165e7602113d88b0': invalid argument.

Does anybody has any solution?

zx1986 *August 24, 2016*

0

I failed on Linode ...

root@ubuntu:/var/log# service docker status

● docker.service - Docker Application Container Engine
Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
Active: failed (Result: signal) since Wed 2016-08-24 15:09:17 UTC; 2min 11s ago
Docs: https://docs.docker.com
Main PID: 3881 (code=killed, signal=KILL)

Aug 24 13:07:30 ubuntu systemd[1]: Starting Docker Application Container Engine...
Aug 24 13:07:30 ubuntu dockerd[3881]: time="2016-08-24T13:07:30.731767656Z" level=info msg="libcontainerd: new containerd process, pid: 3984"
Aug 24 15:07:47 ubuntu dockerd[3881]: time="2016-08-24T15:07:47.334383528Z" level=info msg="Processing signal 'terminated'"
Aug 24 15:09:17 ubuntu systemd[1]: docker.service: State 'stop-sigterm' timed out. Killing.
Aug 24 15:09:17 ubuntu systemd[1]: docker.service: Main process exited, code=killed, status=9/KILL
Aug 24 15:09:17 ubuntu systemd[1]: Stopped Docker Application Container Engine.

Aug 24 15:09:17 ubuntu systemd[1]: docker.service: Unit entered failed state.
Aug 24 15:09:17 ubuntu systemd[1]: docker.service: Failed with result 'signal'.

---

gurukumara *August 29, 2016*

0

I like the Clear and Simple explanation. Excellent!

---

waryak *September 16, 2016*

0

in September 2016, problems with 1st step - docker installation

---

ganeshb *October 12, 2016*

0

The push refers to a repository [docker.io/ganesh08/grasp/ubuntu-python]
An image does not exist locally with the tag: ganesh08/grasp/ubuntu-python

--- this was an error while iam pushing an image into the docker.
is there any solution?

---

couchpotatoe *October 23, 2016*

0

I run into this issue, complaining about libsystemd-journal0 being required. when I run the "sudo apt-get install -y docker-engine" step. I haven't been able to install libsystemd-journal0 with apt-get. Is there another recommended way to install this? Or did I miss a step?

amato@amato-VirtualBox:~$ sudo apt-get install -y docker-engine
Reading package lists... Done
Building dependency tree

Reading state information... Done
Some packages could not be installed. This may mean that you have
requested an impossible situation or if you are using the unstable
distribution that some required packages have not yet been created
or been moved out of Incoming.
The following information may help to resolve the situation:

The following packages have unmet dependencies:
docker-engine : Depends: libsystemd-journal0 (>= 201) but it is not installable
Recommends: aufs-tools but it is not going to be installed
Recommends: cgroupfs-mount but it is not going to be installed or
cgroup-lite but it is not going to be installed
Recommends: git
E: Unable to correct problems, you have held broken packages.

---

mleewise *November 18, 2016*

0

I created a user specifically for docker. Would someone concerned about security frown upon adding that user to the docker group to avoid having to use sudo? I already did the standard stuff for restricting ssh access (no login, ssh required). Thanks !

Squonk42 *December 9, 2016*

0

Extremely useful tutorial, thank you!

---

billyallen *January 6, 2017*

0

excellent

Allwo *March 5, 2017*

0

Thank you very much for this tutorial!

However, I get an error that is, after of a couple of attemps, is really frustrating me. I have been installing docker a lot of times already, also on this OS, but this never happened.

I am stuck at the end of Step 1, when docker-engine cannot be installed:

```
~# systemctl status docker.service
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: failed (Result: exit-code) since So 2017-03-05 17:47:20 CET; 32s ago
     Docs: https://docs.docker.com
 Main PID: 18194 (code=exited, status=1/FAILURE)
```

```
dockerd[18194]: time="2017-03-05T17:47:20.567753592+01:00" level=error msg="'overlay' not found as a supported filesystem on this host
dockerd[18194]: time="2017-03-05T17:47:20.569299675+01:00" level=error msg="'overlay' not found as a supported filesystem on this host
dockerd[18194]: time="2017-03-05T17:47:20.591796895+01:00" level=info msg="Graph migration to content-addressability took 0.00 seconds
dockerd[18194]: time="2017-03-05T17:47:20.592394882+01:00" level=warning msg="Your kernel does not support oom control"
dockerd[18194]: time="2017-03-05T17:47:20.592410368+01:00" level=warning msg="Your kernel does not support memory swappiness"
dockerd[18194]: time="2017-03-05T17:47:20.592421460+01:00" level=warning msg="Your kernel does not support kernel memory limit"
dockerd[18194]: time="2017-03-05T17:47:20.592427398+01:00" level=warning msg="Unable to find cpu cgroup in mounts"
dockerd[18194]: time="2017-03-05T17:47:20.592458649+01:00" level=warning msg="Unable to find cpuset cgroup in mounts"
dockerd[18194]: time="2017-03-05T17:47:20.592490516+01:00" level=warning msg="mountpoint for pids not found"
dockerd[18194]: Error starting daemon: Devices cgroup isn't mounted
```

I added root to the group, also I found the advice to add `GRUB_CMDLINE_LINUX="cgroup_enable=memory swapaccount=1"` to the file /etc/default/grub, but that file does not exist!

Thank you for your help!

jayb7c00ad853d0b1e62f6bfce *April 14, 2017*

0 p80.pool.sks-keyservers.net does not resolve, so you can't get keys this way anymore.