Docker Quick Start

SUSE Linux Enterprise Server 12 SP1

This guide introduces Docker, a lightweight virtualization solution to run virtual units simultaneously on a single control host.

Publication Date: April 21, 2016

Contents

1 Terminology 2

2 Overview 3

3 Setting Up a Docker Host 5

4 Basic Docker Operations 7

5 Building Docker Images 7

6 SUSE Linux Enterprise Images for Docker 8

7 Hosting Docker Images On-premise 12

8 Portus 14

Docker is a lightweight virtualization solution to run multiple virtual units (containers) simultaneously on a single control host. Containers are isolated with Kernel Control Groups (*cgroups*) and *Kernel Namespaces*.

Full virtualization solutions such as Xen, KVM, or `libvirt` are based on the processor simulating a complete hardware environment and controlling the virtual machines. However, Docker only provides operating system-level virtualization where the Linux kernel controls isolated containers.

1 Terminology

The following list contains important terminology and introduces you to certain fundamental concepts of Docker.

cgroups

Control Groups is a Linux kernel feature that allows aggregating or partitioning tasks (processes) and all their children into hierarchically organized groups to isolate resources.

Image

A *virtual machine* on the host server that can run any Linux system, for example SUSE Linux Enterprise Server, SUSE Linux Enterprise Desktop, or openSUSE. A Docker image is made by a series of layers built one over the other. Each layer corresponds to a permanent change committed from a container to the image. For more details, see the official Docker documentation at http://docs.docker.com/engine/reference/glossary/#image/.

Image Name

A name that refers to an image. The name is used by the Docker commands.

Container

A running Docker image.

Container ID

An ID that refers to a particular container. The ID is used by the Docker commands.

Tag

A string associated with an image. It is commonly used to identify a specific version of an image (similar to tags in version control systems). It is also possible to refer to the same image with different tags.

Kernel Namespaces

Namespaces are a kernel feature to isolate some resources like network, users, and others for a group of processes.

Docker Host Server

The system that runs the Docker daemon, and provides images and management control capabilities through *cgroups*.

Registry

A remote storage for Docker images.

Repository

Place where all the versions of a Docker image are kept.

2 Overview

Docker is a platform that allows developers and system administrators to manage the complete life cycle of images. Docker makes it easy to build, ship and run images containing applications.

BENEFITS OF DOCKER

- Isolation of applications and operating systems through containers.
- Near native performance, as Docker manages allocation of resources in real time.
- Controls network interfaces and resources available inside containers through cgroups.
- Versioning of images.
- Allows building new images based on existing ones.
- Docker Hub allows sharing and storing of images on public [http://docs.docker.com/docker-hub/] or private [http://docs.docker.com/userguide/dockerrepos/#private-repositories] repositories.

LIMITATIONS OF DOCKER

- Containers run inside the host system's kernel and cannot use a different kernel.
- Only allows Linux *guest* operating systems.
- Docker is not a full virtualization stack like Xen, KVM, or `libvirt`.
- Security depends on the host system. Refer to the official security documentation [http://docs.docker.com/articles/security/] for more details.

2.1 Container Drivers

Docker uses libcontainer [https://github.com/docker/libcontainer] as the back-end driver to handle containers.

2.2 Storage Drivers

Docker supports different storage drivers:

- `vfs`: this driver is automatically used when the Docker host file system does not support copy-on-write. This is a simple driver which does not offer some advantages of Docker (like sharing layers, more on that in the next sections). It is highly reliable but also slow.
- `devicemapper`: this driver relies on the device-mapper thin provisioning module. It supports copy-on-write, hence it offers all the advantages of Docker.
- `btrfs`: this driver relies on Btrfs to provide all the features required by Docker. To use this driver the `/var/lib/docker` directory must be on a Btrfs file system.
- `AUFS`: this driver relies on the AUFS union file system. Neither the upstream kernel nor the SUSE one supports this file system. Hence the AUFS driver is not built into the SUSE Docker package.

SLE 12 uses the Btrfs file system by default, which leads Docker to use the `btrfs` driver.

It is possible to specify which driver to use by changing the value of the `DOCKER_OPTS` variable defined inside of the `/etc/sysconfig/docker` file. This can be done either manually or using YaST by browsing to *System* › */etc/sysconfig Editor* › *System* › *Management* › *DOCKER_OPTS* menu and entering the `-s storage_driver` string.

For example, to force the usage of the `devicemapper` driver enter the following text:

```
DOCKER_OPTS="-s devicemapper"
```

Note: Mounting `/var/lib/docker`

It is recommended to have `/var/lib/docker` mounted on a different file system to not affect the Docker host operating system in case of a file system corruption.

3 Setting Up a Docker Host

3.1 General Preparation

Prepare the host as described below. Before installing any Docker-related packages, you need to enable the container module:

PROCEDURE 1: ENABLING THE CONTAINER MODULE USING YAST

1. Start YaST, and select *Software* › *Software Repositories*.
2. Click *Add* to open the add-on dialog.
3. Select *Extensions and Modules from Registration Server* and click *Next*.
4. From the list of available extensions and modules, select *Container Module 12 x86_64* and click *Next*.
   The containers module and its repositories will be added to your system.
5. If you use Subscription Management Tool, update the list of repositories on the SMT server.

PROCEDURE 2: ENABLING THE CONTAINER MODULE USING SUSECONNECT

- The Container Module can be added also with the following command:

  ```
  $ sudo SUSEConnect -p sle-module-containers/12/x86_64 -r ''
  ```

  Note: Note about the SUSEConnect syntax

  The `-r ''` flag is required to avoid a known limitation of SUSEConnect.

PROCEDURE 3: INSTALLING AND SETTING UP DOCKER

1. Install the `docker` package:

   ```
   sudo zypper install docker
   ```

2. To automatically start the Docker service at boot time:

   ```
   sudo systemctl enable docker.service
   ```

This will automatically enable docker.socket in consequence.

3. Start the Docker service:

   ```
   sudo systemctl start docker.service
   ```

   This will automatically start docker.socket in consequence.

The Docker daemon listens on a local socket which is accessible only by the `root` user and by the members of the `docker` group. The `docker` group is automatically created at package installation time. To allow a certain user to connect to the local Docker daemon, use the following command:

```
sudo /usr/sbin/usermod -aG docker USERNAME
```

The user can communicate with the local Docker daemon upon his next login.

3.2 Networking

If you want your containers to be able to access the external network, you must enable the `ipv4 ip_forward` rule. This can be done using YaST by browsing to *System* › *Network Settings* › *Routing* menu and ensuring `Enable IPv4 Forwarding` is checked.

This option cannot be changed when networking is handled by the Network Manager. In such cases the `/etc/sysconfig/SuSEfirewall2` file needs to be edited manually to ensure the `FW_ROUTE` flag is set to `yes`:

```
FW_ROUTE="yes"
```

3.2.1 Networking Limitations on Power Architecture

Currently Docker networking has two limitations on the Power architecture.

The first limitation is about iptables. SLE 12 machines cannot run Docker with the iptables support enabled. An update of the kernel is going to solve this issue. In the meantime the Docker package for Power has iptables support disabled via a dedicated directive inside of `/etc/sysconfig/docker`.

As a result of this limitation Docker containers are not going to have access to the outer network. A possible workaround is to share the same network namespace between and host and the containers. This however reduces the isolation of the containers.

The network namespace of the host can be shared on a per-container basis by supplying the `--net=host` to the `docker run` command.

Note: iptables support on SLE 12 SP1

SLE 12 SP1 hosts are not affected by this limitation but, given they use the same SLE 12 package, they are going to have iptables support disabled. This can be changed by removing the `-iptables=false` setting inside of `/etc/sysconfig/docker`.

The second limitation is about network isolation between the containers and the host. Currently it is not possible to prevent containers from probing or accessing arbitrary ports of each other.

4 Basic Docker Operations

Images can be pulled from Docker's central index at http://index.docker.io using the following command:

```
docker pull IMAGE_NAME
```

Containers can be started using the **`docker run`** command. Refer to the official documentation of Docker at http://docs.docker.com/ for more details.

5 Building Docker Images

Starting with version 5.06.8, KIWI can be used to build Docker images. Also see the official KIWI documentation, in particular https://doc.opensuse.org/projects/kiwi/doc/#chap.lxc. The official `kiwi-doc` package contains examples of Docker images.

Docker has an internal build system [http://docs.docker.com/reference/builder/] which can be used to create new images based on existing ones.

Some users might be confused about which build system to use for what task. The recommended approach is to customize the official SLE images for Docker using the Docker build system. This approach has two advantages:

- You can use Docker-specific directives (like `ENTRYPOINT`, `EXPOSE`, etc).
- You can re-use existing layers.

Sharing the common layers between different images makes it possible to:

- Use less disk space on the Docker hosts and on the remote registry service.
- Deploy faster: only the requested layers are sent over the network (this is similar to upgrading installed packages using delta RPMs).
- Take full advantage of caching while building Docker images: this will result in faster executions of the **`docker build`** command.

SUSE creates the official base images [http://docs.docker.com/reference/glossary/#base-image] using KIWI and publishes them to allow customers to use those as foundation blocks inside of the Docker build system. Only the Docker images built on top of our official ones via the Docker build system are going to be supported.

6 SUSE Linux Enterprise Images for Docker

Currently we cannot distribute SLE images for Docker because there is no way to associate an End-User License Agreement (EULA) to a Docker image. However, we provide official pre-built Docker images for SLE and a convenience tool called sle2docker [https://github.com/SUSE/sle2docker] that can be used to activate these images locally.

Pre-built images do not have repositories configured. But when the Docker host has an SLE subscription that provides access to the product used in the image, Zypper will automatically have access to the right repositories. For more details see *Section 6.3, "Customizing the Images"*.

6.1 Installing sle2docker

The **`sle2docker`** tool is part of the official container module. It can be installed using the following command:

```
sudo zypper install sle2docker
```

Note: The Docker Daemon Needs to Be Running

`sle2docker` requires the Docker daemon to be running on the system.

6.2 Activating the Pre-built Docker Images

The official pre-built Docker images for SLE are shipped as RPM packages inside of the `containers` module. They can be installed using Zypper:

```
sudo zypper in sles11sp3-docker-image sles12-docker-image
```

`sle2docker` can list the available pre-built images with the following command:

```
sle2docker list
```

To activate a pre-built image, use the following command:

```
sle2docker activate PRE-BUILT_IMAGE_NAME
```

At the end of the activation, **`sle2docker`** will print the name of the Docker image that has been created.

6.3 Customizing the Images

To create custom Docker images based on the official ones use Docker's integrated build system [http://docs.docker.com/reference/builder/].

The pre-built images do not have any repository configured. They contain a zypper service [https://github.com/SUSE/container-suseconnect] that contacts either the SUSE Customer Center (SCC) or your Subscription Management Tool (SMT) server, according to the configuration of the SLE host that runs the Docker container. The service obtains the list of repositories available for the product used by the Docker image.

You do not need to add any credentials to the Docker image because the machine credentials are automatically injected into the container by the docker daemon. They are injected inside of the `/run/secrets` directory. The same applies to the `/etc/SUSEConnect` file of the host system, which is automatically injected into the `/run/secrets` file.

Note: Credentials and Security

The contents of the `/run/secrets` directory are never committed to a Docker image, hence there is no risk of your credentials leaking.

To obtain the list of repositories use the following command:

```
zypper ref -s
```

It will automatically add all the repositories to your container. For each repository added to the system a new file is going to be created under **`/etc/zypp/repos.d`**. The URLs of these repositories include an access token that automatically expires after 12 hours. To renew the token call the **`zypper ref -s`** command. It is secure to commit these files to a Docker image.

If you want to use a different set of credentials, place a custom `/etc/zypp/credentials.d/SCCcredentials` file inside of the Docker image. It contains the machine credentials that have the subscription you want to use. The same applies to the `SUSEConnect` file: to override the file available on the host system that is running the Docker container, add a custom `/etc/SUSEConnect` file inside of the Docker image.

6.3.1 Creating a Custom SLE 12 Image

The following Docker file creates a simple Docker image based on SLE 12:

```
FROM suse/sles12:latest


RUN zypper --gpg-auto-import-keys ref -s
RUN zypper -n in vim
```

When the Docker host machine is registered against an internal SMT server, the Docker image requires the SSL certificate used by SMT:

```
FROM suse/sles12:latest

# Import the crt file of our private SMT server
ADD http://smt.test.lan/smt.crt /etc/pki/trust/anchors/smt.crt
RUN update-ca-certificates

RUN zypper --gpg-auto-import-keys ref -s
RUN zypper -n in vim
```

6.3.2 Creating a Custom SLE 11 SP3 Image

The following Docker file creates a simple Docker image based on SLE 11 SP3:

```
FROM suse/sles11sp3:latest

RUN zypper --gpg-auto-import-keys ref -s
RUN zypper -n in vim
```

When the Docker host machine is registered against an internal SMT server, the Docker image requires the SSL certificate used by SMT:

```
FROM suse/sles11sp3:latest

# Import the crt file of our private SMT server
ADD http://smt.test.lan/smt.crt /etc/ssl/certs/smt.pem
RUN c_rehash /etc/ssl/certs

RUN zypper --gpg-auto-import-keys ref -s
RUN zypper -n in vim
```

7 Hosting Docker Images On-premise

This section addresses some common questions like: what can be done with the custom Docker images I created? How can they be shared within my organization?

The easiest solution would be to push these images to the Docker Hub. However, there are a couple of important matters to keep in mind: By default, all the images pushed to the Docker Hub are public. To keep them private you need to pay a subscription-based fee. Moreover you do not have control over the servers where your data is stored. This may not comply with your organization's or company's policies to have full control over their intellectual property. The solution to this problem is simple: To avoid the use of Docker Hub, you can run an on-site Docker registry where to store all the Docker images used by your organization or company.

7.1 What is a Docker Registry?

The Docker registry is an open source project created by Docker Inc. It allows the storage and retrieval of Docker images. By running a local instance of the Docker registry it is possible to completely avoid usage of the Docker Hub.

The Docker registry is also used by the Docker Hub. However, the Docker Hub, as seen from the user perspective, is made of the following parts at least:

- The user interface (UI): The part that is accessed by users with their browser. The UI provides a nice and intuitive way to browse the contents of the Docker Hub either manually or by using a search feature. It also allows to create organizations made by different users. This component is closed source.
- The authentication component: This is used to protect the images stored inside of the Docker Hub. It validates all push, pull and search requests.
  This component is closed source.
- The storage back-end: This is where the Docker images are sent and downloaded from. It is provided by the Docker registry.
  This component is open source.

7.2 Installing and Setting Up Docker Registry

1. Install the `docker-distribution-registry` package:

   ```
   sudo zypper install docker-distribution-registry
   ```

2. To automatically start the Docker registry at boot time:

   ```
   sudo systemctl enable registry
   ```

3. Start the Docker registry:

   ```
   sudo systemctl start registry
   ```

The Docker registry configuration is defined inside of `/etc/registry/config.yml`.

With the default configuration the registry listens on ports `5000` and stores the Docker images under `/var/lib/docker-registry`.

Note: Incompatible Versions of Docker and Docker Registry

Docker registry version 2.3 is not compatible with Docker versions older than 1.10, because v2 manifests were only introduced with Docker 1.10. As Docker and Docker registry can be installed on different boxes, the versions might be incompatible. If you experience communication errors between Docker and Docker registry, update both to the latest versions.

For more details about Docker registry and its configuration, see the official documentation at: https://docs.docker.com/registry/.

7.3 Limitations

The Docker registry has two major limitations:

- It lacks any form of authentication. That means everybody with access to the Docker registry can push and pull images to it. That also includes the possibility to overwrite already existing images.
- There is no way to see which images have been pushed to the Docker registry. You can manually take notes of what is being stored inside of it. There is also no search functionality, which makes collaboration harder.

The next section is going to introduce Portus, the solution to all of the problems above.

8 Portus

Portus is an authentication service and user interface for the Docker registry. It is an open source project created by SUSE to address all the limitations faced by the local instances of Docker registry.

By combining Portus and Docker registry, it is possible to have a secure and enterprise ready on-premise version of the Docker Hub.

Portus is accessible for SLE 12 customers through the Containers module. To install Portus, use the following command:

```
sudo zypper in Portus
```

In order to configure Portus properly, follow these steps:

1. First of all, you should install Portus' dependencies if you haven't already. This is thoroughly documented here: http://port.us.org/docs/setups/1_rpm_packages.html#portus-dependencies. This document will help you to get through the installation process, and it will also warn you about some of the common pitfalls.

2. After installing Portus and its dependencies, you need to configure your instance. The initial setup of Portus is explained here: http://port.us.org/docs/setups/1_rpm_packages.html#initial-setup. When you are done with portusctl, you should modify some configurable values before using Portus. This is thoroughly explained in this documentation page: http://port.us.org/docs/Configuring-Portus.html.

3. To apply the configuration changes, restart Apache (this is required after each configuration change).

4. Finally, when entering Portus for the first time, you will be required to enter some information about your installed registry. For details, see: http://port.us.org/docs/setups/1_rpm_packages#the-default-installation.html.

5. The Portus setup is now complete and you can start using Portus.

Currently, Portus is part of SUSE's Docker offer as a technology preview. For more information and documentation about Portus, see: http://port.us.org/.