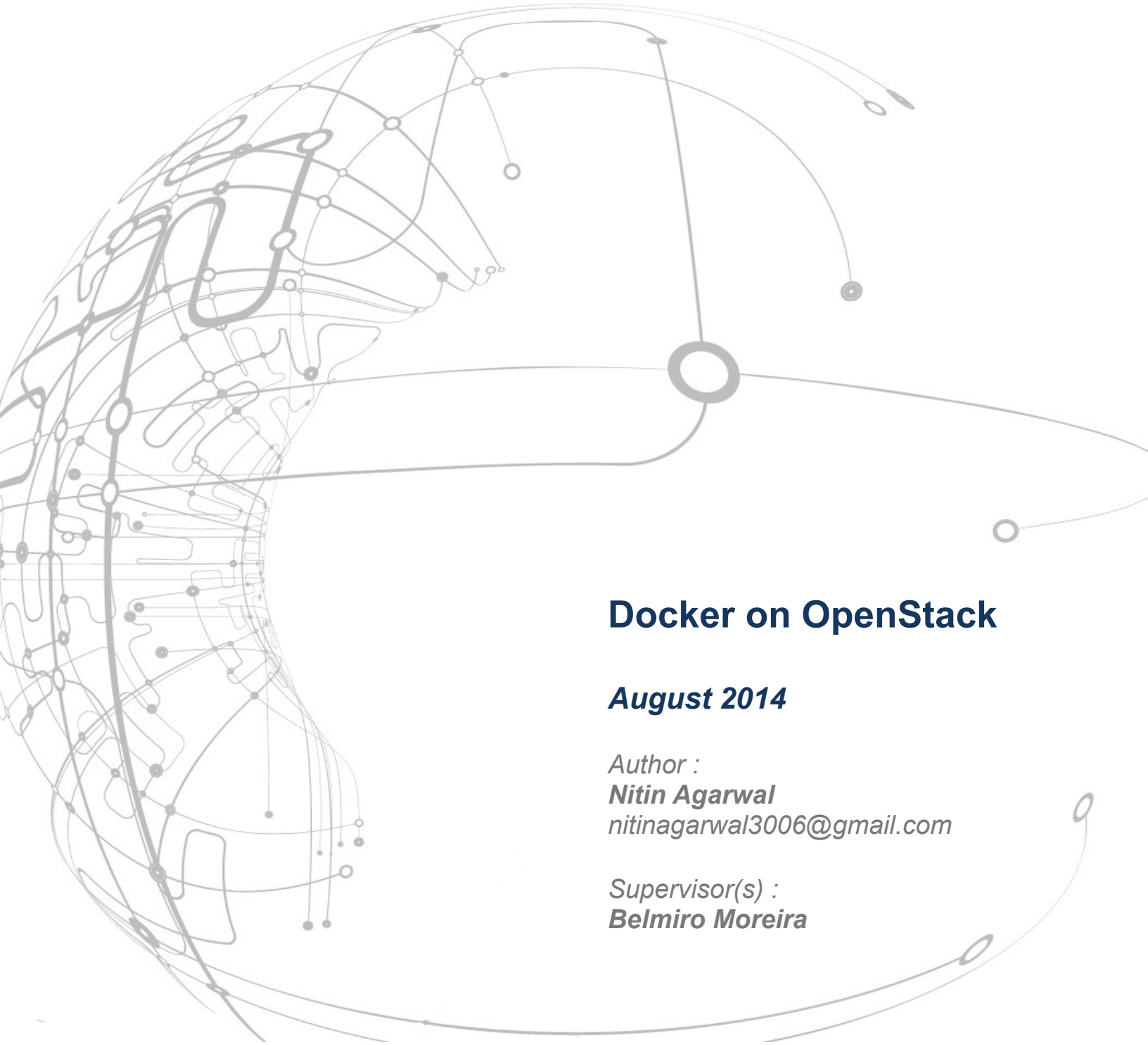


Docker on OpenStack

August 2014

Author :
Nitin Agarwal
nitinagarwal3006@gmail.com

Supervisor(s) :
Belmiro Moreira

CERN openlab Summer Student Report 2014

CERN openlab

Project Specification

CERN is establishing a large scale private cloud based on OpenStack as part of the expansion of the computing infrastructure for storing the data coming out of the Large Hadron Collider (LHC) experiments.

As the data coming out of the detectors is increasing continuously that needs to be stored in the data center, we need more physical resources (more money) and since Virtual machines takes lot of CPU and memory overhead and minutes for creating the images, booting up and for snapshotting as well. So here comes the solution to use Docker containers.

Docker is an open platform to build, ship and run distributed applications. Docker being a container based virtualisation framework makes use of LXC. Docker containers are lightweight and fast and docker makes use of Union File System which makes it unique. Docker comes with the Docker Index/ Hub where you can store and share the docker images.

This project involves the understanding of Docker and docker containers in detail, deployment of private Docker Registry as well as the integration of docker with Openstack to enable the Nova compute service to use the docker API as compute driver instead of the libvirt API.

Abstract

At CERN, with the ever increasing amount of data coming out of the detectors that needs to be stored in the data center, new ways are sought to help analyze and store this data as well as help researchers perform their own experiments. To help offer solutions to such problems, CERN has employed the use of cloud computing and in particular OpenStack; an open source and scalable platform for building public and private clouds.

OpenStack is used to view, create, and manage resources in a cloud and automate the tasks. Compute nodes form the resource core of the OpenStack Compute cloud, providing the processing, memory, network and storage resources to run instances.

As the data is increasing continuously around 50 PB/sec and about 5 PB/day of data that needs to be stored, CERN is looking for new ways to utilise the hardware resources of the data center more efficiently.

In this project we outline and document the integration of Docker with the Nova compute service of OpenStack (Devstack, Packstack), deployment of private Docker Registry at CERN for pushing and pulling the docker images. To allow the Nova compute service to use to the Docker API as compute driver instead of the Libvirt driver and to allow nova to boot the docker images, we need to store the docker images in glance that acts as an independent docker registry after configuration.

In this report, we describe about docker, its basics and importance of docker containers in comparison to virtual machines, steps for deploying and configuring the private Docker Registry at CERN and steps for configuring the Nova to use docker driver in Devstack on Ubuntu cloud image and Packstack on RHEL 7.

Table of Contents

1 Introduction

Let's consider the deployment of a relatively simple application **Wordpress**. A typical Wordpress installation requires Apache 2, PHP 5, MySQL, Wordpress source code, MySQL database with Wordpress configured to use this database, apache configuration to load the PHP module, enable the support for URL rewriting and .htaccess files, DocumentRoot pointing to the Wordpress sources.

While deploying and running a system like this on our server, we may run into some problems and challenges namely **Isolation, Security, Upgrades, downgrades**, **Snapshotting, backing up, Reproducibility, Constrain resources, Ease of installation and Ease of removal.**

At CERN, we have around 50 PB/sec of data coming out of the detectors and about 5 PB/day to be stored in the servers deployed at the data center, we make use of **OpenStack** to view, create, and manage resources in a cloud and automate the tasks. Compute nodes form the resource core of the OpenStack Compute cloud, providing the processing, memory, network and storage resources to run instances.

When we decide to run each individual application on a separate virtual machine, most of our problems go away but we come across other issues :

- **Money :** can we actually really afford booting up an instance for every application we need? Also can we predict the instance size we will need, because if we need more resources later, we need to stop the VM to upgrade it or over-pay for resources we don't end up using.

- **Time :** many operations related to virtual machines are typically slow. Booting takes minutes, snapshotting too takes minutes, creating an image takes minutes. The world keeps turning and we don't have so much of time!

So using **Docker**, Container based virtualisation framework and an open platform to build, ship and run distributed applications is the solution. Docker containers are lightweight and fast.

Booting up a VM is a big deal as it takes up few minutes to get started and a significant amount of memory whereas booting up a Docker container is fast and uses very little CPU and memory overhead. Almost comparable to starting a regular process. Not only running a container is fast, building an image and snapshotting the filesystem is fast as well. Docker containers are *portable* to any operating system that runs Docker whether it's Ubuntu or CentOS.

- **Isolation :** Docker isolates applications at the filesystem and networking level. It feels a lot like running "real" virtual machines in that sense.

- **Reproducibility:** We can build the system just the way we like (either by logging in and apt-get in all software, or using a Dockerfile), then commit the changes to an image. We can now instantiate as many instances of it as we want or transfer to an image to another machine to reproduce exactly the same setup.

- **Security:** Docker containers are more secure than regular process isolation. Link

- **Constrain resources:** Docker currently supports limiting CPU usage to a certain share of CPU cycles, memory usage can also be limited. Restricting disk usage is not directly supported as of yet.

- **Ease of installation:** Docker has Docker Hub / Registry, a repository with off-the-shelf docker images we can instantiate with a single command.

- **Ease of removal:** If we don't need an application anymore, just destroy the container.

- **Upgrades, downgrades:** Boot up the new version of an application first, then switch over the load balancer from the old port to the new one.

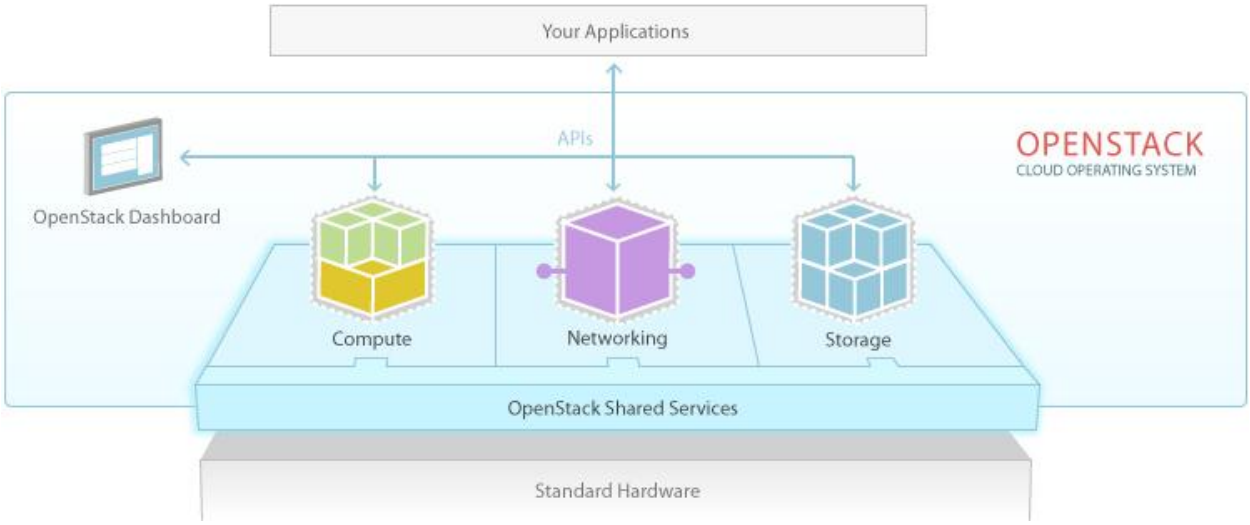
- **Snapshotting, backing up:** Docker supports committing and tagging of images, which incidentally, unlike snapshotting a VM is instant.

2 OpenStack

Openstack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a data center.

2.1 Overview

The OpenStack project contains various components that individually provide compute, storage, networking and the dashboard but together create a functioning cloud operating system (OS).



The components of OpenStack are :-

- **Compute (Nova)** - the Infrastructure as a Service (IaaS) system providing virtual machines to hosts with nova-compute installed.
- **Identity Service (Keystone)** - provides the authentication and authorization for all OpenStack components.
- **Image Service (Glance)** - an image repository for all virtual disk images. Glance can also be configured to store these images on a remote cluster, such as Ceph.
- **Dashboard (Horizon)** - the user interface to easily control most aspects of the OpenStack components. As an alternative, the OpenStack API can be used.

- **Networking (Neutron)** - provides “networking as a service” by allowing users to create their own networks and interfaces as well as manage IPs.

- **Object Storage (Swift)** - is a highly available and distributed object/blob store.

- **Block Storage (Cinder)** - provides “block storage as a service”

2.2 Installing and Running OpenStack

We have installed Devstack on the Ubuntu 14.04 image in a dedicated VM. DevStack is a set of scripts and utilities to quickly deploy an OpenStack cloud.

Clone the github repository of the devstack by executing the command :
>>> git clone https://github.com/openstack-dev/devstack

To start a dev cloud run the following NOT AS ROOT (see **DevStack Execution Environment** below for more on user accounts):
>>> ./stack.sh

When the script finishes executing, you should be able to access OpenStack endpoints, like so :
- Horizon: http://myhost/
- Keystone: http://myhost:5000/v2.0/

We also provide an environment file that you can use to interact with your cloud via CLI:
source openrc file to load your environment with OpenStack CLI creds
. openrc OR **source openrc**

list instances
nova list

2.3 Working with Nova CLI

Nova is a computing project for OpenStack. The list of all the commands that can be executed with nova can be seen here. Some of the most common nova client commands to get familiarised with and work on Docker with OpenStack are mentioned below :-

- **nova boot** - Boot a new server.
- **nova delete** - Immediately shut down and delete specified server(s).
- **nova flavor-list** - Print a list of available 'flavors' (sizes of servers).

- **nova image-create** - Create a new image by taking a snapshot of a running server.
- **nova image-delete** - Delete specified image(s).
- **nova image-list** - Print a list of available images to boot from.
- **nova list** - List active servers.
- **nova service-list** - Show a list of all running services. Filter by host & binary.
- **nova show** - Show details about the given server.

2.4 Working with Glance CLI

Glance is the image service for OpenStack. It serves as an image repository for all virtual disk images. The list of all the commands that can be executed with glance can be seen here. Some of the most common glance client commands to get familiarised with and work on Docker with OpenStack are mentioned below :-

- **glance image-list** - List images you can access.
- **glance image-show** - Describe a specific image.
- **glance image-create** - Create a new image.
- **glance image-delete** - Delete specified image(s).

3 Docker

3.1 Overview

Docker is an open platform to build, ship and run distributed applications. Docker consists of :-

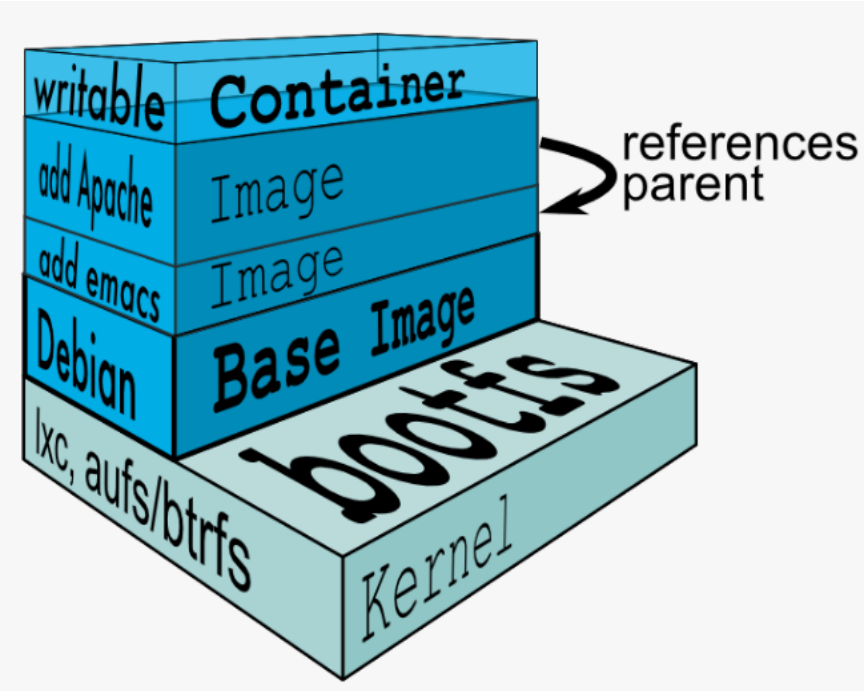
- **Docker Engine**, a lightweight and powerful open source container virtualization technology combined with a work flow for building and containerizing your applications.

- **Docker Hub / Registry**, a SaaS service for sharing and managing your application stacks and automating workflows.

Docker lets you quickly assemble applications from components and eliminates the friction that can come when shipping code. As a result, IT can ship faster and run the same app, unchanged, on laptops, data center VMs and on any cloud infrastructure. Docker lets you get your code tested and deployed into production as fast as possible.

The next wave of virtualization, and one that has the potential to displace hypervisor-based virtualization on Linux platforms, is upon us now that Docker, the software container and application packaging system.

3.2 Basics of the Docker System



Docker makes packaging all of the parts of an application—the tools, configuration files, libraries, and more—into a much simpler task. It's a bit like a virtual machine, conceptually, allowing for the segmentation of a single powerful machine to be shared with many applications with their own individual specific configuration requirements, and without allowing those applications to interfere with one another. Except unlike a virtual machine, applications run natively on the underlying Linux kernel, and each is just very carefully segmented from one another and from the underlying operating system.

Docker currently uses Linux Containers, which runs in the same operating system as its host and provides system level virtualisation. This allows it to share a lot of the host operating system resources. It uses aufs as the file system.

Docker uses Linux *cgroups* and *name spaces* to isolate processes from each other so they appear to run on their own system. Docker consists of three parts: Docker Daemon, Docker Images, the Docker Repositories which together make Linux Container easy and fun to use.

Docker Daemon runs as root and orchestrates all running containers. Just as virtual machines are based on images, Docker Containers are based on Docker images. These images are tiny compared to virtual machine images and are stackable.

AuFS is a layered file system, so you can have a read only part, and a write part, and merge those together. So you could have the common parts of the operating system as read only, which are shared amongst all of your containers, and then give each container its own mount for writing.

Namespaces : Docker uses namespaces to provide the isolated workspace for containers. When we run a container, Docker creates a set of *namespaces* for that container. This provides a layer of isolation: each aspect of a container runs in its own namespace and does not have access outside it.

Some of the namespaces that Docker uses are:

- **The pid namespace:** Used for process isolation (PID: Process ID).
- **The net namespace:** Used for managing network interfaces (NET: Networking).
- **The ipc namespace:** Used for managing access to IPC resources (IPC: InterProcess Communication).
- **The mnt namespace:** Used for managing mount-points (MNT: Mount).
- **The uts namespace:** Used for isolating kernel and version identifiers. (UTS: Unix Timesharing System).

Control groups : Docker also makes use of another technology called cgroups or control groups. A key to running applications in isolation is to have them only use the resources you want. This ensures containers are good multi-tenant citizens on a host. Control groups allow Docker to share available hardware resources to containers and, if required, set up limits and constraints. For example, limiting the memory available to a specific container.

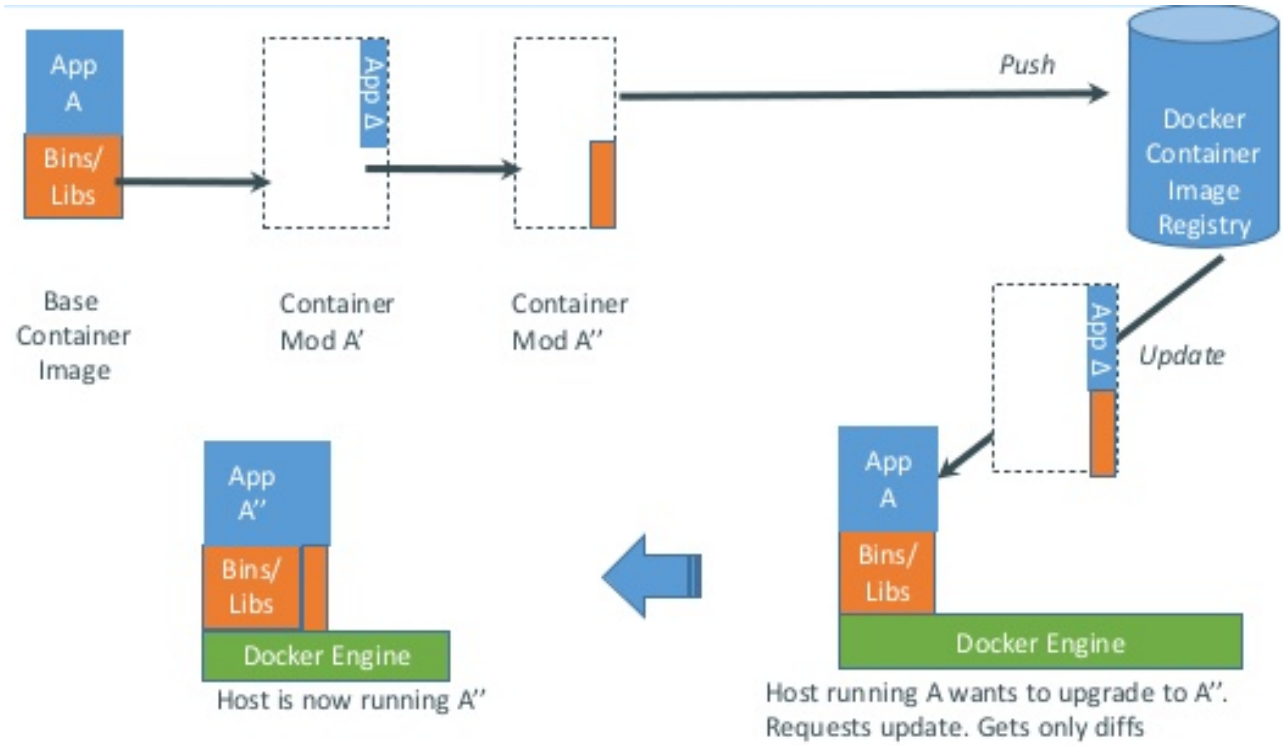
Union file systems, or **UnionFS**, are file systems that operate by creating layers, making them very lightweight and fast. Docker uses union file systems to provide the building blocks for containers. Docker can make use of several union file system variants including: AUFS, btrfs, vfs, and DeviceMapper.

Container Format : Docker combines these components into a wrapper we call a container format. The default container format is called libcontainer. Docker also supports traditional Linux containers using LXC.

So let's say you have a container image that is 1GB in size. If you wanted to use a Full VM, you would need to have 1GB times x number of VMs you want. With LXC and AuFS you can share the bulk of the 1GB and if you have 1000 containers you still might only have a little over 1GB of space for the containers OS, assuming they are all running the same OS image.

A full virtualized system gets its own set of resources allocated to it, and does minimal sharing. You get more isolation, but it is much heavier (requires more resources).
With LXC you get less isolation, but they are more lightweight and require less resources. So you could easily run 1000's on a host, and it doesn't even blink. Try doing that with Xen or KVM, and unless you have a really big host, I don't think it is possible.

A full virtualized system usually takes minutes to start, LXC containers take seconds, and sometimes even less than a second.



3.3 Advantages of using Docker

- Docker allows the faster delivery of your applications. Docker containers are lightweight and fast. Containers have sub-second launch times, which helps in reducing the cycle time of development, testing and deployment.

- Docker containers run almost everywhere. You can deploy containers on desktops, physical servers, virtual machines, into data centers, and up to public and private clouds.You can easily move an application from a testing environment into the cloud and back whenever you need.
- Docker's lightweight containers also make scaling up and down fast and easy. You can quickly launch more containers when needed and then shut them down easily when they're no longer needed.

- Docker containers don't need a hypervisor, so you can pack more of them onto your hosts to get higher density and run more workloads. This means you get more value out of every server and can potentially reduce what you spend on equipment and licenses.

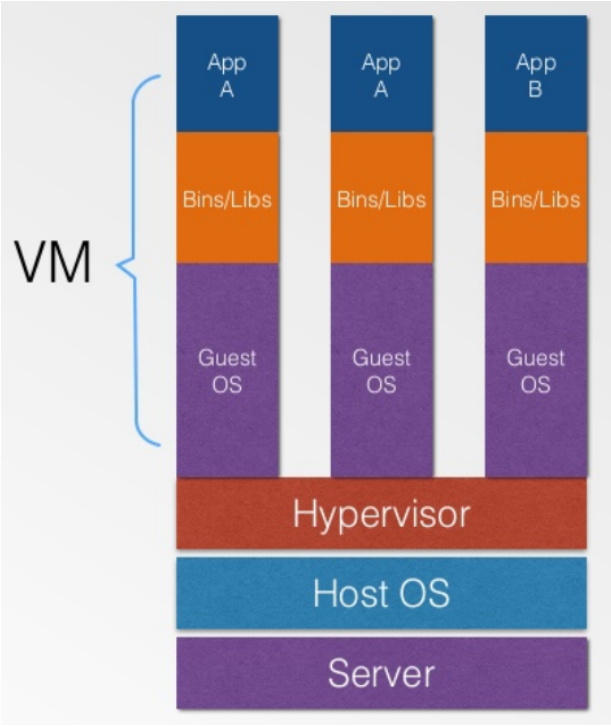
- As Docker speeds up with your work flow, it is easy for the sysadmins to make lots of small changes instead of huge, big bang updates. Smaller changes mean reduced risk and more uptime.

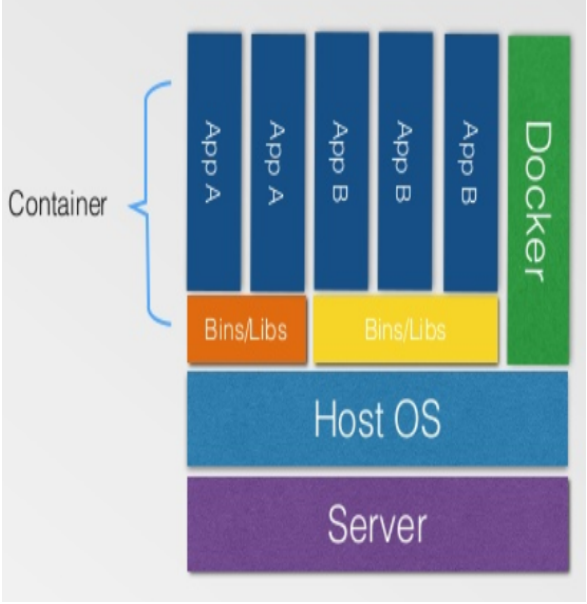
3.4 How are Docker Containers lightweight ?

In VMs, every app, every copy of the app and every slight modification of the app requires a new virtual server whereas in Containers,

- **Original App :** No OS to take up space, resources or require restart.
- **Copy of the App :** No OS, can share binaries / libraries.
- **Modified App :** Union file system allows us to only save the diffs among the containers.

Containers are Isolated, but share OS and, where appropriate, bins / libraries.





3.5 Docker Containers vs VMs

If you want full isolation with guaranteed resources, a full VM is the way to go, whereas if you just want to isolate processes from each other and want to run a ton of them on a reasonably sized host, then LXC is the way to go.

Deploying a consistent production environment is easier as even if you use tools like chef and puppet, there are always OS updates and other things that change between hosts and environments.

What docker does is it gives you the ability to snapshot the OS into a common image, and makes it easy to deploy on other docker hosts.

This is great for unit testing, lets say you have 1000 tests and they need to connect to a database, and in order to not break anything you need to run serially so that the tests don't step on each other (run each test in a transaction and roll back). With Docker you could create an image of your database, and then run all the tests in parallel since you know they will all be running against the same snapshot of the database. Since they are running in parallel and in LXC containers they could run all on the same box at the same time, and your tests will finish much faster.

Docker allows to bundle artifacts and configurations in an image. These images run as light weight system-level virtual machines.

3.6 Installing Docker on RHEL 7

Firstly, you need to put the repos file in the **/etc/yum.repos.d/** directory after creating the new file named **rhel7.repo**.
The repos file is available at http://linux.web.cern.ch/linux/rhel/rhel7/rhel7.repo then update the repos file by running the command :
>>> sudo yum update all

Please note that all the docker commands are executed as **root user**.

You can now install docker using the below command :
>>> sudo yum install docker

The default version of docker v0.11.1-dev gets installed.

You need to start the docker service by running the below command as root user

>>> systemctl start docker.service

then you can also check the status of the docker service by running the command :
>>> systemctl status docker.service

You can also stop or restart the docker service.

You can also check the version of the docker installed by running the command :
>>> sudo docker -v command.

4 Docker Containers and Images

4.1 Running Docker Containers

Docker containers are similar to a directory. A Docker container holds everything that is needed for an application to run. Each container is created from a Docker image. Docker containers can be run, started, stopped, moved, and deleted. Each container is an isolated and secure application platform. Docker containers are the **run** component of Docker.

A container consists of an operating system, user-added files, and meta-data. Each container is built from an image and that image tells Docker what the container holds, what process to run when the container is launched, and a variety of other configuration data. The Docker image is read-only. When Docker runs a container from an image, it adds a read-write layer on top of the image (using a union file system) in which your application can then run.

The Docker daemon is the persistent process that manages containers. Docker uses the same binary for both the daemon and client. To run the daemon you provide the -d flag.
>>> sudo docker -d

By using either the docker binary or via the API, the Docker client tells the Docker daemon to run a container.
>>> docker run -i -t ubuntu /bin/bash

The Docker client is launched using the docker binary with the run option telling it to launch a new container. The bare minimum the Docker client needs to tell the Docker daemon to run the container is :

- What Docker image to build the container from, here ubuntu, a base Ubuntu image;
- The command you want to run inside the container when it is launched, here /bin/bash, to start the Bash shell inside the new container.

In more detail, Docker does the following :

- **Pulls the ubuntu image:** Docker checks for the presence of the ubuntu image and, if it doesn't exist locally on the host, then Docker downloads it from the Docker Hub. If the image already exists, then Docker uses it for the new container.
- **Creates a new container:** Once Docker has the image, it uses it to create a container.
- **Allocates a filesystem and mounts a read-write *layer*:** The container is created in the file system and a read-write layer is added to the image.
- **Allocates a network / bridge interface:** Creates a network interface that allows the Docker container to talk to the local host.
- **Sets up an IP address:** Finds and attaches an available IP address from a pool.
- **Executes a process that you specify:** Runs the application.
- **Captures and provides application output:** Connects and logs standard input, outputs and errors for you to see how your application is running.

We now have a running container. From here, we can manage the container, interact with the application application and then, when finished, stop and delete the container.

The various commands available with the Docker Client are :-

- **docker ps** - Lists containers.
- **docker logs** - Shows us the standard output of a container.
- **docker stop** - Stops running containers.
- **docker build** - Build Docker images from a Dockerfile and a “context”.
- **docker commit** - commit an exisiting container.
- **docker cp** - Copy files/folders from a container's filesystem to the host path. Paths are relative to the root of the filesystem.
- **docker diff** - List the changed files and directories in a container’s filesystem.

To view all the commands that can be run using the docker client, execute the command :-
>>> sudo docker

4.2 Working with Docker Images

A Docker image is a read-only template. For example, an image could contain an Ubuntu operating system with Apache and your web application installed. Images are used to create Docker containers. Docker provides a simple way to build new images or update existing images, or you can download Docker images that other people have already created. Docker images are the **build** component of Docker.

Docker images are read-only templates from which Docker containers are launched. Each image consists of a series of layers. Docker makes use of union file systems to combine these layers into a single image. Union file systems allow files and directories of separate file systems, known as branches, to be transparently overlaid, forming a single coherent file system.

One of the reasons Docker is so lightweight is because of these layers. When you change a Docker image—for example, update an application to a new version— a new layer gets built. Thus, rather than replacing the whole image or entirely rebuilding, as you may do with a virtual machine, only that layer is added or updated. Now you don't need to distribute a whole new image, just the update, making distributing Docker images faster and simpler.

Every image starts from a base image, for example ubuntu, a base Ubuntu image, or fedora, a base Fedora image. You can also use images of your own as the basis for a new image, for example if you have a base Apache image you could use this as the base of all your web application images.

Docker images are then built from these base images using a simple, descriptive set of steps we call *instructions*. Each instruction creates a new layer in our image.

Instructions include actions like:

- Run a command.
- Add a file or directory.
- Create an environment variable.
- What process to run when launching a container from this image.

These instructions are stored in a file called a Dockerfile. Docker reads this Dockerfile when you request a build of an image, executes the instructions, and returns a final image.

To list all the images on the Host, execute the command :-
>>> sudo docker images

Here, we can see three crucial pieces of information about our images in the listing.

- What repository they came from.
- The tags for each image.
- The image ID of each image.

Docker will automatically download any image we use that isn't already present on the Docker host. But this can potentially add some time to the launch of a container.
If we want to pre-load an image we can download it using the docker pull command. Let's say we'd like to download the centos image.

>>> sudo docker pull centos

We can also search for images on the command line using the docker search command.
We can search for a suitable image by using the docker search command to find all the images that contain the term ubuntu.

>>> sudo docker search ubuntu

4.3 Unique Advantages of Docker over other container technologies

Docker takes advantage of containers and filesystem technologies in a high-level which are not generic enough to be managed by libvirt.

- **Process-level API :** Docker can collect the standard outputs and inputs of the process running in each container for logging or direct interaction, it allows blocking on a container until it exits, setting its environment, and other process-oriented primitives which don't fit well in libvirt's abstraction.

- **Advanced change control at the filesystem level :** Every change made on the filesystem is managed through a set of layers which can be snapshotted, rolled back, diff-ed etc.

- **Image portability :** The state of any docker container can be optionally committed as an image and shared through a central image Registry. Docker images are designed to be portable across infrastructures, so they are a great building block for hybrid cloud scenarios.

- **Build facility :** Docker can automate the assembly of a container from an application's source code. This gives developers an easy way to deploy payloads to an OpenStack cluster as part of their development workflow.

5 Docker Registry

5.1 Overview

A Docker Registry is a SaaS service for sharing and managing your application stacks.
It allows you to push the Docker Images for storage and sharing and pull the images whenever needed.

In general, a repository is a collection of images which are hosted in a registry. When you use the public Docker.io repository, also known as Docker Index, your repository is *<username>/<repo_name>.*

At CenterDevice, private Docker registries allows to safely share Docker images in an organization. One can maintain all the backend services as well as the app images in a private registry. In this way, a developer only needs to pull changed images to update his development environment.

Docker registries hold images. These are public or private stores from which you upload or download images. The public Docker registry is called Docker Hub. It provides a huge collection of existing images for your use. These can be images you create yourself or you can use images that others have previously created.

Once you build a Docker image, you can *push* it to a public registry Docker Hub or to your own deployed private registry. Using the Docker client, you can search for already published images and then pull them down to your Docker host to build containers from them.

Docker Hub provides both public and private storage for images. Public storage is searchable and can be downloaded by anyone. Private storage is excluded from search results and only you and your users can pull down images and use them to build containers.

5.2 Deploying your own private Docker Registry

For deploying the private docker image repository, also termed as docker registry, you need to follow the below mentioned steps in chronological order.

1. Install git and wget on RHEL 7 by running the command :
>>> sudo yum install git
>>> sudo yum install wget

2. Install the required dependencies by running the command :
>>> sudo yum install python-devel libevent-devel python-pip gcc xz-devel

Note : python-pip does not gets installed while the other packages gets successfully installed.

3. To install python-pip, firstly install the EPEL repositories by running the below commands :
>>> cd /tmp
>>> wget http://dl.fedoraproject.org/pub/epel/beta/7/x86_64/epel-release-7-0.2.noarch.rpm
>>> ls *.rpm
>>> sudo yum install epel-release-7-0.2.noarch.rpm

4. Now you can install python-pip by running the command :
>> sudo yum install python-pip

5. There exists two ways to install docker registry :

Install docker registry from the github repository by running the commands :
>>> sudo git clone https://github.com/dotcloud/docker-registry
>>> cd docker-registry
>>> pip install .

Assuming we are in the /var directly while cloning the docker-registry code from github.

OR
Install docker registry directly by running the command :
>>> sudo python-pip install docker-registry
The package which gets installed is **https://pypi.python.org/pypi/docker-registry/0.7.0**

6. The Docker Registry comes with a sample configuration file, config_sample.yml. Copy this to config.yml to provide a basic configuration :

If you have installed docker registry using the first method, run the below commands :
>>> cd /var/docker-registry
>>> cp config/config_sample.yml config/config.yml

Else run the below commands :
>>> cd /usr/lib/python2.7/site-packages/
>>> cp config/config_sample.yml config/config.yml

7. Make a directory where you wish to store your images and repositories for the docker registry by running the commands :

Here, I'm making a directory named docker_registry in /var directory.
>>> cd /var
>>> mkdir docker_registry

8. Docker Registry can run in several flavors. This enables you to run it in development mode, production mode or your own predefined mode.

In the config_sample.yml file, you'll see several sample flavors:

- common: used by all other flavors as base settings
- local: stores data on the local filesystem
- s3: stores data in an AWS S3 bucket
- dev: basic configuration using the local flavor
- test: used by unit tests
- prod: production configuration (basically a synonym for the s3 flavor)
- gcs: stores data in Google cloud storage
- swift: stores data in OpenStack Swift
- glance: stores data in OpenStack Glance, with a fallback to local storage
- glance-swift: stores data in OpenStack Glance, with a fallback to Swift
- elliptics: stores data in Elliptics key/value storage

You can define your own flavors by adding a new top-level yaml key.

You can specify which flavor to run by setting SETTINGS_FLAVOR in your environment:
>>> export SETTINGS_FLAVOR=dev

The default flavor is **dev**.

NOTE - The configuration flavour used is local to store data on the file system.

You need to specify the storage type and storage path under the dev configuration flavour in the config.yml file.
storage: local
storage_path: /var/docker_registry

Example configuration -

- common :
  - loglevel: info
  - search_backend: "_env:SEARCH_BACKEND:"
  - sqlalchemy_index_database:
  - "_env:SQLALCHEMY_INDEX_DATABASE:sqlite:////tmp/docker-registry.db"
- prod :
  - loglevel: warn
  - storage: s3
  - s3_access_key: _env:AWS_S3_ACCESS_KEY
  - s3_secret_key: _env:AWS_S3_SECRET_KEY
  - s3_bucket: _env:AWS_S3_BUCKET
  - boto_bucket: _env:AWS_S3_BUCKET
  - storage_path: /srv/docker
  - smtp_host: localhost
  - from_addr: docker@meSPAMNOT.com
  - to_addr: contact@meSPAMNOT.com
- dev :
  - loglevel: debug
  - storage: local
  - storage_path: /var/docker_registry
- test :
  - storage: local
  - storage_path: /tmp/tmpdockertmp

9. Specify the config file to be used by setting DOCKER_REGISTRY_CONFIG in your environment :
>>> export DOCKER_REGISTRY_CONFIG=config.yml

The default location of the config file is config.yml, located in the config subdirectory. If DOCKER_REGISTRY_CONFIG is a relative path, that path is expanded relative to the config subdirectory.

10. You can run the docker registry by running the command :
>>> gunicorn --access-logfile - --debug -k gevent -b 0.0.0.0:5000 -w 1 docker_registry.wsgi:application

The recommended setting to run the Registry in a prod environment is gunicorn behind a nginx server which supports chunked transfer-encoding (nginx >= 1.3.9).

Gunicorn is a Python WSGI HTTP Server for UNIX. It's a pre-fork worker model ported from Ruby's Unicorn project. It is broadly compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy.

5.3 Pushing and Pulling Images from Registry

To push an image to the docker registry, first you need to tag the image and then push to the docker registry:

>>> sudo docker tag <image-id> hostname.cern.ch:5000/slc6
>>> sudo docker push hostname.cern.ch:5000/slc6

The image gets pushed to your docker registry.

You can pull the image from the docker registry by running the command :
>>> sudo docker pull hostname.cern.ch:5000/slc6

6 Docker on OpenStack

6.1 Overview

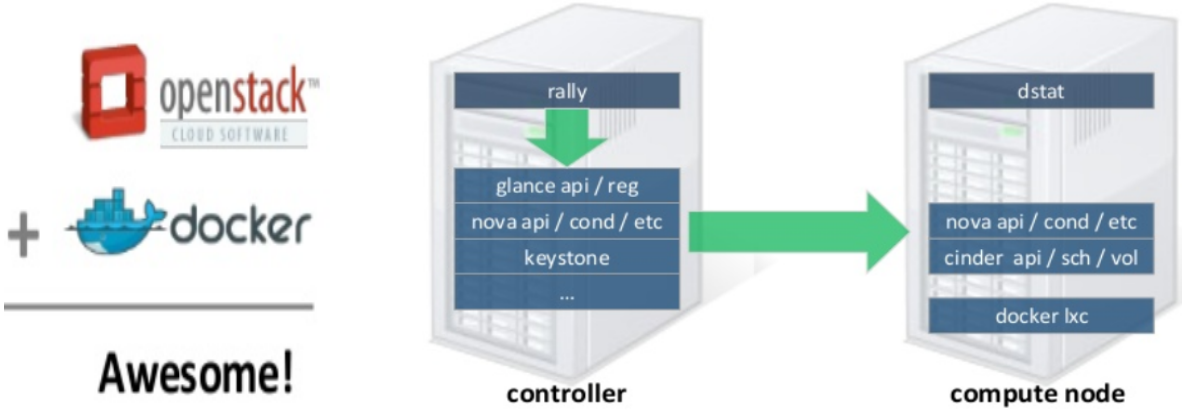
The Docker driver is a hypervisor driver for Openstack Nova Compute.

Docker is an open-source engine which automates the deployment of applications as highly portable, self-sufficient containers which are independent of hardware, language, framework, packaging system and hosting provider.

Docker provides management of Linux containers with a high level API providing a lightweight solution that runs processes in isolation. It provides a way to automate software deployment in a secure and repeatable environment. A Docker container includes a software component along with all of its dependencies - binaries, libraries, configuration files, scripts, virtualenvs, jars, gems, tarballs, etc. Docker can be run on any x64 Linux kernel supporting cgroups and aufs.

Docker is a way of managing multiple containers on a single machine. However used behind Nova makes it much more powerful since it's then possible to manage several hosts, which in turn manage hundreds of containers. The current Docker project aims for full OpenStack compatibility.

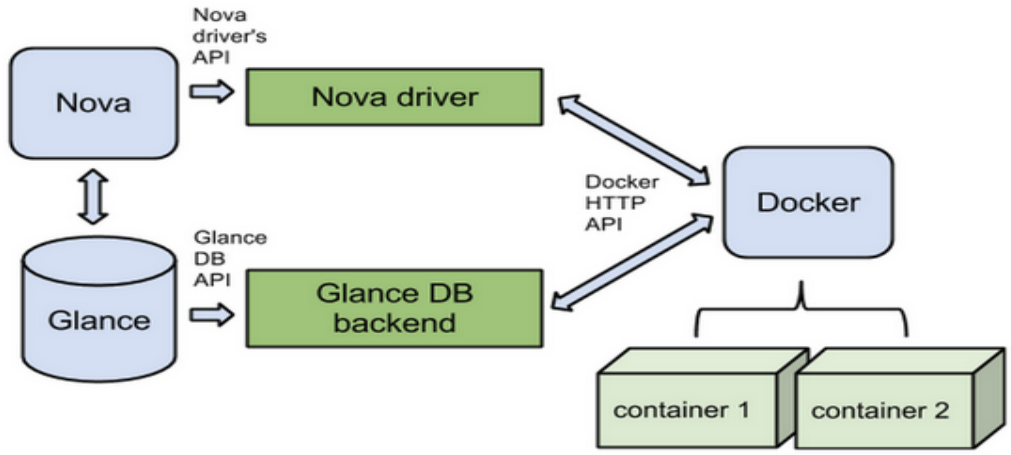
Containers don't aim to be a replacement for VMs, they are complementary in the sense that they are better for specific use cases.

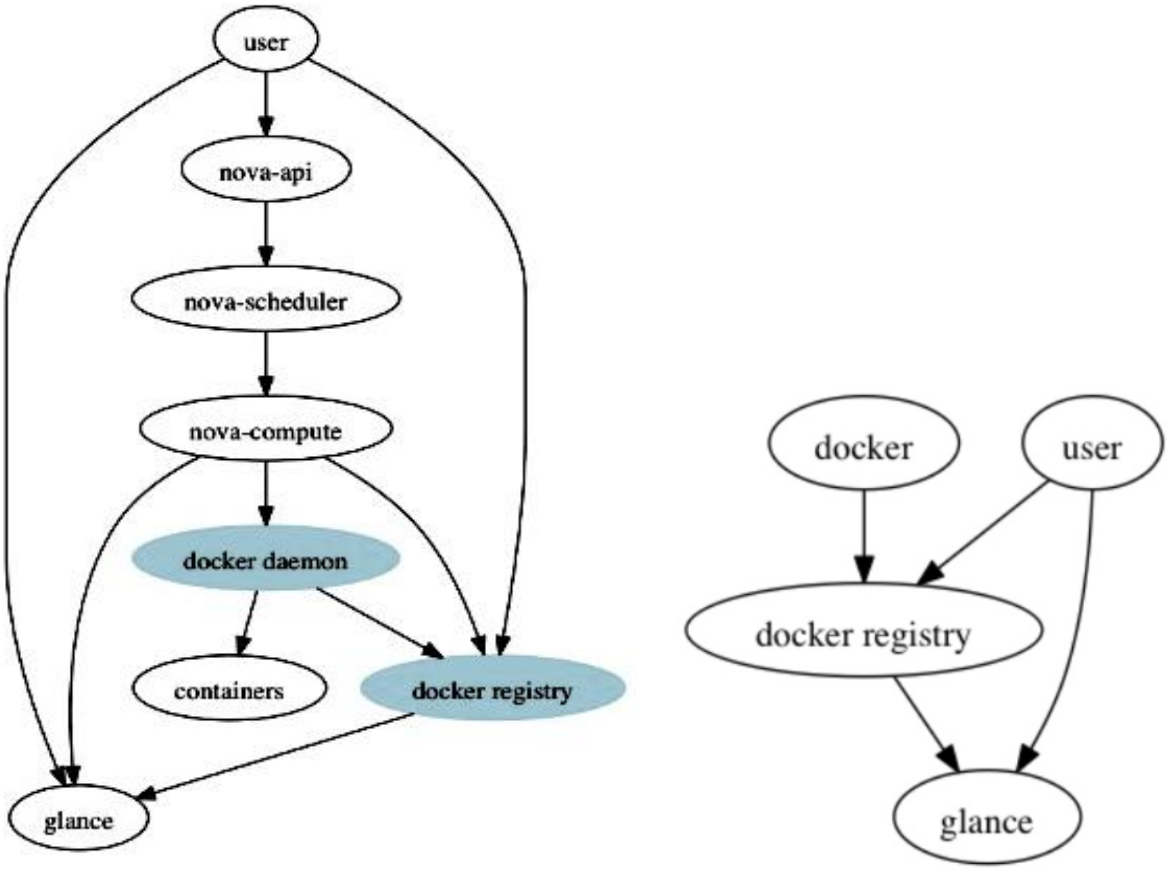


6.2 Nova Docker Architecture

The Nova driver embeds a tiny HTTP client which talks with the Docker internal Rest API through a unix socket. It uses the HTTP API to control containers and fetch information about them.

The driver will fetch images from the OpenStack Image Service (Glance) and load them into the Docker filesystem. Images may be placed in Glance by exporting them from Docker using the 'docker save' command.





- Docker registry is an image proxy.
- Users can upload through docker registry or to glance directly.
- Docker pulls images through the docker registry proxy.

6.3 Configuring OpenStack to enable Docker

6.3.1 Installing Docker for OpenStack

In order for Nova to communicate with Docker over its local socket, add *nova* to the *docker* group and restart the compute service to pick up the change:
>>> usermod -G docker nova
>>> service openstack-nova-compute restart

You will also need to install the nova-docker driver by executing the commands :
>>> git clone https://git.openstack.org/stackforge/nova-docker
>>> cd nova-docker
>>> python setup.py install

6.3.2 Nova Configuration

Nova needs to be configured to use the Docker virt driver.
Edit the configuration file /etc/nova/nova.conf according to the following options:

[DEFAULT]
compute_driver = novadocker.virt.docker.DockerDriver

Restart the Nova compute service to pick up the change :
>>> systemctl restart openstack-nova-compute.service

You can check the status of the nova compute service by executing the command :
>>> systemctl status openstack-nova-compute.service

6.3.3 Glance Configuration

Glance needs to be configured to support the **docker** container format. It's important to leave the default ones in order to not break an existing glance install. Edit the **glance-api.conf** file located in **/etc/glance** and restart the glance-api service to pick up the change.

>>> vi /etc/glance/glance-api.conf
container_formats=ami,ari,aki,bare,ovf,docker

>>> systemctl restart openstack-glance-api.service

You can check the status of the glance api service by executing the command :
>>> systemctl status openstack-glance-api.service

6.4 Deployment with Devstack

Using Docker hypervisor through DevStack replaces all manual configuration needed above. To install the nova-docker on Devstack, run the following commands :

>>> git clone https://git.openstack.org/stackforge/nova-docker /opt/stack/nova-docker
>>> git clone https://git.openstack.org/openstack-dev/devstack /opt/stack/devstack

Note : only needed until we can make use of configure_nova_hypervisor_rootwrap
>>> git clone https://git.openstack.org/openstack/nova /opt/stack/nova

>>> cd /opt/stack/nova-docker
>>> ./contrib/devstack/prepare_devstack.sh

Now, run devstack by running the commands :
>>> cd /opt/stack/devstack
>>> ./stack.sh

6.5 Uploading Docker Images to Glance

Images can now be saved directly to Glance:

>>> docker pull ubuntu
>>> docker save ubuntu | glance image-create --container-format=docker --disk-format=raw --name ubuntu

The name of the image in Glance should be explicitly set to the same name as the image as it is known to Docker. In the example above, an image has been tagged in Docker as 'ubuntu'. Matching this is the '--name ubuntu' argument to glance image-create. If these names do not align, the image will not be bootable.

NOTES :

- Earlier releases of this driver required the deployment of a private docker registry which is no longer required. Images are now saved and loaded from Glance, which serves as an independent docker registry to store images.

- Images loaded from Glance may do bad things. Only allow administrators to add images. Users may create snapshots of their containers, generating images in Glance -- these images are managed and thus safe.

6.6 Booting Instances using Nova

You can now boot instances with docker images using nova by executing the command :
>>> nova boot --flavor <flavor-type> --image <image-id> <instance-name>

The different flavor types available are **m1.tiny**, **m1.small**, **m1.medium** and **m1.large**. You can specify any of these flavor types.

You can check the instance id and the status of the instance booted by executing the command :

>>> nova list

OR
>>> nova show <instance-id>

7 Docker on Packstack

7.1 Overview

Packstack is a utility that uses Puppet modules to deploy various components of OpenStack on multiple pre-installed servers over SSH automatically. Currently only Fedora, Red Hat Enterprise Linux (RHEL) and compatible derivatives of both are supported.

Docker is an open-source engine which automates the deployment of applications as highly portable, self-sufficient containers which are independent of hardware, language, framework, packaging system and hosting provider.

Docker provides management of Linux containers with a high level API providing a lightweight solution that runs processes in isolation. It provides a way to automate software deployment in a secure and repeatable environment. A Docker container includes a software component along with all of its dependencies - binaries, libraries, configuration files, scripts, virtualenvs, jars, gems, tarballs, etc.

The Docker driver is a hypervisor driver for Openstack Nova Compute. It was introduced with the Havana release, but lives out-of-tree for Icehouse. Being out-of-tree has allowed the driver to reach maturity and feature-parity faster than would be possible should it have remained in-tree.

7.2 Installing Packstack on RHEL 7

1. Firstly, you need to put the repos file in the **/etc/yum.repos.d/** directory after creating the new file named **rhel7.repo**.

The repos file is available at **http://linux.web.cern.ch/linux/rhel/rhel7/rhel7.repo**
then update the repos file by running the command :
>>> sudo yum update all

2. You can now install docker using the below command :
>>> sudo yum install docker

The default version of docker v0.11.1-dev gets installed.
Please note that all the docker commands are executed as root user.

3. Install Icehouse RDO by executing the command :
>>> sudo yum install -y
http://rdo.fedorapeople.org/openstack-icehouse/rdo-release-icehouse.rpm

On the controller node run:
>>>sudo yum install openstack-packstack

4. Run packstack by executing the command :
>>> packstack --gen-answer-file=config.txt

Edit config.txt for your environment and then execute :
>>> packstack --answer-file=config.txt

5. Source the keystonerc_admin file and verify services are up by executing the below commands :
>>> source keystonerc_admin
>>> openstack-status

6. Download and source the **Openstack RC** file from the openstack dashboard. Login with the username and password provided in the keystone_admin file.
>>> source admin-openrc.sh

7.3 Configuring Packstack to enable Docker

7.3.1 Installing Docker for Packstack

Install nova-docker by executing the commands :
>>> git clone https://git.openstack.org/stackforge/nova-docker
>>> cd nova-docker
>>> python setup.py install

7.3.2 Nova Configuration

Enable the driver in nova configuration file **nova.conf** located in **/etc/nova**.
>>> vi /etc/nova/nova.conf

compute_driver=novadocker.virt.docker.DockerDriver

In order for Nova to communicate with Docker over its local socket, add nova to the docker group and restart the compute service to pick up the change :
>>> usermod -G docker nova
>>> systemctl restart openstack-nova-compute.service

You can check the status of the nova compute service by executing the command :
>>> systemctl status openstack-nova-compute.service

ERROR : The Nova compute service failed to restart and got inactive as the nova docker has been allowed to use only the docker client API version 1.13 which does not comes by default in RHEL 7.

Check this :
https://github.com/stackforge/nova-docker/blob/master/novadocker/virt/docker/client.py#L98

In rhel 7, the default docker client API version 1.12 comes.

STEPS to resolve the above ERROR :

1. Remove the default version of docker by executing the command :
>>> sudo yum remove docker

2. Get the RPM to install the docker version 1.1.2 that has the client API version 1.13.
>>> wget
https://kojipkgs.fedoraproject.org//packages/docker-io/1.1.2/2.fc22/x86_64/docker-io-1.1.2-2.fc22.x86_64.rpm

3. Install the RPM by executing the command :
>>> sudo yum install docker-io-1.1.2-2.fc19.x86_64.rpm

4. Check the version of docker installed by executing the command :
>>> sudo docker version

Output :
Client version: 1.1.2
Client API version: 1.13
Go version (client): go1.2.2
Git commit (client): d84a070/1.1.2

Server version: 1.1.2
Server API version: 1.13
Go version (server): go1.2.2
Git commit (server): d84a070/1.1.2

5. Make sure docker daemon is running by executing the command :
>>> ps auwx | grep docker

If docker is not running currently, execute the command :
>>> sudo docker -d

6. In order for Nova to communicate with Docker over its local socket, add nova to the docker group and restart the compute service to pick up the change :
>>> usermod -G docker nova
>>> systemctl restart openstack-nova-compute.service

7. Check the status of the nova compute service by executing the command :
>>> systemctl status openstack-nova-compute.service

OR
To check the status only of the nova services, execute the command :
>>> nova-manage service list

7.3.3 Glance Configuration

Glance needs to be configured to support the **docker** container format. It's important to leave the default ones in order to not break an existing glance install. Edit the **glance-api.conf** file located in **/etc/glance** and restart the glance-api service to pick up the change.
>>> vi /etc/glance/glance-api.conf

container_formats=ami,ari,aki,bare,ovf,docker

>>> systemctl restart openstack-glance-api.service

You can check the status of the glance api service by executing the command :
>>> systemctl status openstack-glance-api.service

7.4 Uploading Docker Images to Glance

Images can now be saved directly to glance by executing the commands :

>>> docker pull ubuntu
>>> docker save ubuntu | glance image-create --container-format=docker --disk-format=raw --name ubuntu

The name of the image in Glance should be explicitly set to the same name as the image as it is known to Docker. In the example above, an image has been tagged in Docker as 'ubuntu'. Matching this is the '--name ubuntu' argument to glance image-create. If these names do not align, the image will not be bootable.

7.5 Booting Instances using Nova

You can now boot instances with docker images using nova by executing the command :
>>> nova boot --flavor <flavor-type> --image <image-id> <instance-name>

The different flavor types available are **m1.tiny**, **m1.small**, **m1.medium** and **m1.large**. You can specify any of these flavor types.

You can check the instance id and the status of the instance booted by executing the command ;
>>> nova list

OR
>>> nova show <instance-id>

ERROR : The instance booted resulted in error. The fault message displayed was "No valid host was found".

For more information, you can check the nova-compute logs in the **/var/log/nova/nova-compute.log** file and nova-scheduler logs in the **/var/log/nova/nova-scheduler.log** file.

STEPS to resolve the above ERROR :

1. Edit the **/etc/nova/nova.conf** file to change the scheduler default filters to return all hosts so the scheduler will return some instance.

scheduler_default_filters = ComputeFilter

Reference : https://answers.launchpad.net/nova/+question/192511

Restart the scheduler service to pick up the change by executing the command :
>>> systemctl restart openstack-nova-scheduler.service

Check the status of the nova scheduler service by executing the command :
>>> systemctl status openstack-nova-scheduler.service

The same **ERROR** still persists while booting an instance with docker image.
NOTE : The same error also persists while booting any image other than docker images.

2. Relaunched the Instance with the RHEL 7 image. Now, installed Packstack following the same steps but disabled the nova neutron service for networking and configured the nova network on a single host **eth0**.

CONFIG_NEUTRON_INSTALL=n
CONFIG_NOVA_COMPUTE_PRIVIF=eth0
CONFIG_NOVA_NETWORK_PRIVIF=eth0

You can disable the Cinder, Ceilometer and Cinder service for quick installation of Packstack as they are not needed.

CONFIG_CINDER_INSTALL=n
CONFIG_SWIFT_INSTALL=n
CONFIG_CEILOMETER_INSTALL=n

NOTE : The same ERROR [No Valid Host was found] still persists while booting an instance with docker image.

8 Containers on Google Cloud Platform

8.1 Overview

Everything at Google, from Search to Gmail, is packaged and run in a Linux container. Every week, Google launches more than 2 billion container instances across their global data centers, and the power of containers has enabled both more reliable services and higher, more-efficient scalability.

The Kubernetes and container-optimized VM releases makes it simple to run Docker containers on Google Cloud Platform. The container-optimized VM provides a way to statically and declaratively run multiple Docker containers on a Google Compute Engine instance. The Kubernetes enables dynamic container scheduling across multiple VM instances, including over container-optimized VMs.

8.2 Container VMs

Container-optimized Google Compute Engine images are Debian images with a few additions :

- The Docker runtime is pre-installed, so you are ready to create containers as soon as your instance is up.
- The image includes an agent that handles container manifest files, to create and monitor containers automatically.

8.3 Kubernetes

Kubernetes is an open source container cluster manager. It schedules any number of container replicas across a group of node instances. A master instance exposes the Kubernetes API, through which tasks are defined. Kubernetes spawns containers on nodes to handle the defined tasks.

The number and type of containers can be dynamically modified according to need. An agent (a *kubelet*) on each node instance monitors containers and restarts them if necessary.

Kubernetes is optimized for Google Cloud Platform, but can run on any physical or virtual machine.

9 Conclusion

In conclusion, Docker is inarguably a huge plus, must be released in Production at CERN to solve the issues of money and time and to use the hardware resources more efficiently.

- **Fast Boot :** Docker provides ability to create / delete a clean environment quickly which is great for Unit Testing. Using Docker makes a big difference when developing Chef recipes in which a clean environment must be started for every failure.

- **Small Container Sizes and ability to diff containers :** While it's nearly impossible to share a VM with a remote teammate, it's entirely possible to share a container due to the Docker registry because of its ability to pull only incremental changes. The ability to diff two containers is indeed a big point of observation.

- **Ability to run a container in AWS as well as on Cross Cloud :** We can not run VMware in AWS, but Docker offers the ability to use the same container in AWS. Furthermore, Docker can use the same container cross cloud, which is indeed a win.

12 Bibliography

- Docker - https://www.docker.com/
- Docker Hub - https://registry.hub.docker.com/
- Docker Registry - https://github.com/docker/docker-registry
- OpenStack - http://www.openstack.org/
- Devstack - https://github.com/openstack-dev/devstack
- Nova Docker - https://github.com/stackforge/nova-docker
- Packstack - https://github.com/stackforge/packstack