Docker Images

Agenda

In this session, you will learn about:

- Understand Container Image
- Understand Dockerfile
- Image vs Container
- Creating custom Image
- Tag an Image

Docker Image

- An image is a read-only template with instructions for creating a Docker container.
- Often, an image is based on another image, with some additional customization.
- For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

Docker Image…

- We can create custom images using **Dockerfile** with a simple syntax for defining the steps needed to create the image and run it.
- Each instruction in a Dockerfile creates a layer in the image.

Docker Images

- Images can share layers to optimize:
  - Disk usage
  - Transfer times
  - Memory use

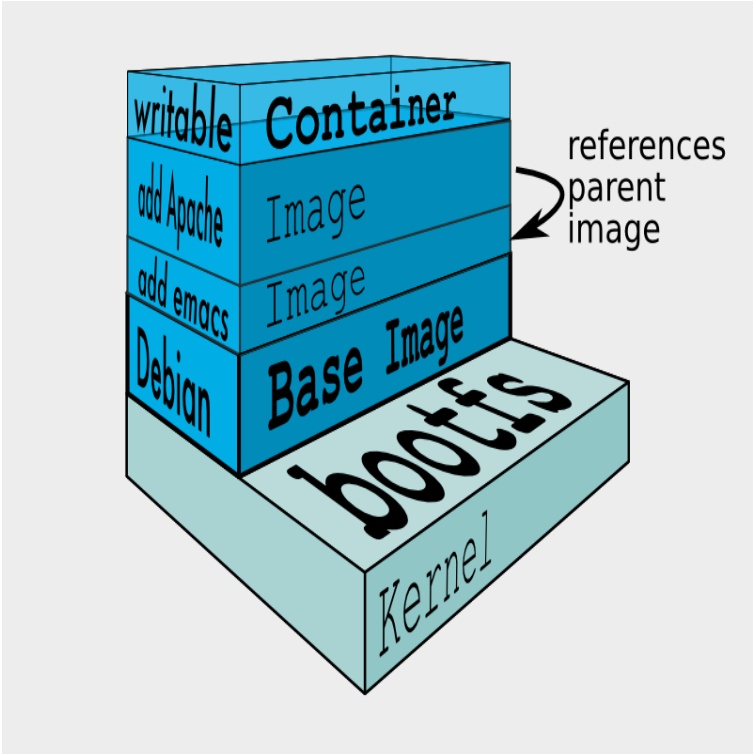


Image vs Container

- An image is a read-only filesystem
- A container is an encapsulated set of processes running in a read-write copy of that filesystem
- To optimize container boot time, copy-on-write is used instead of regular copy
- “docker container run” starts a container from a given image

Managing Images

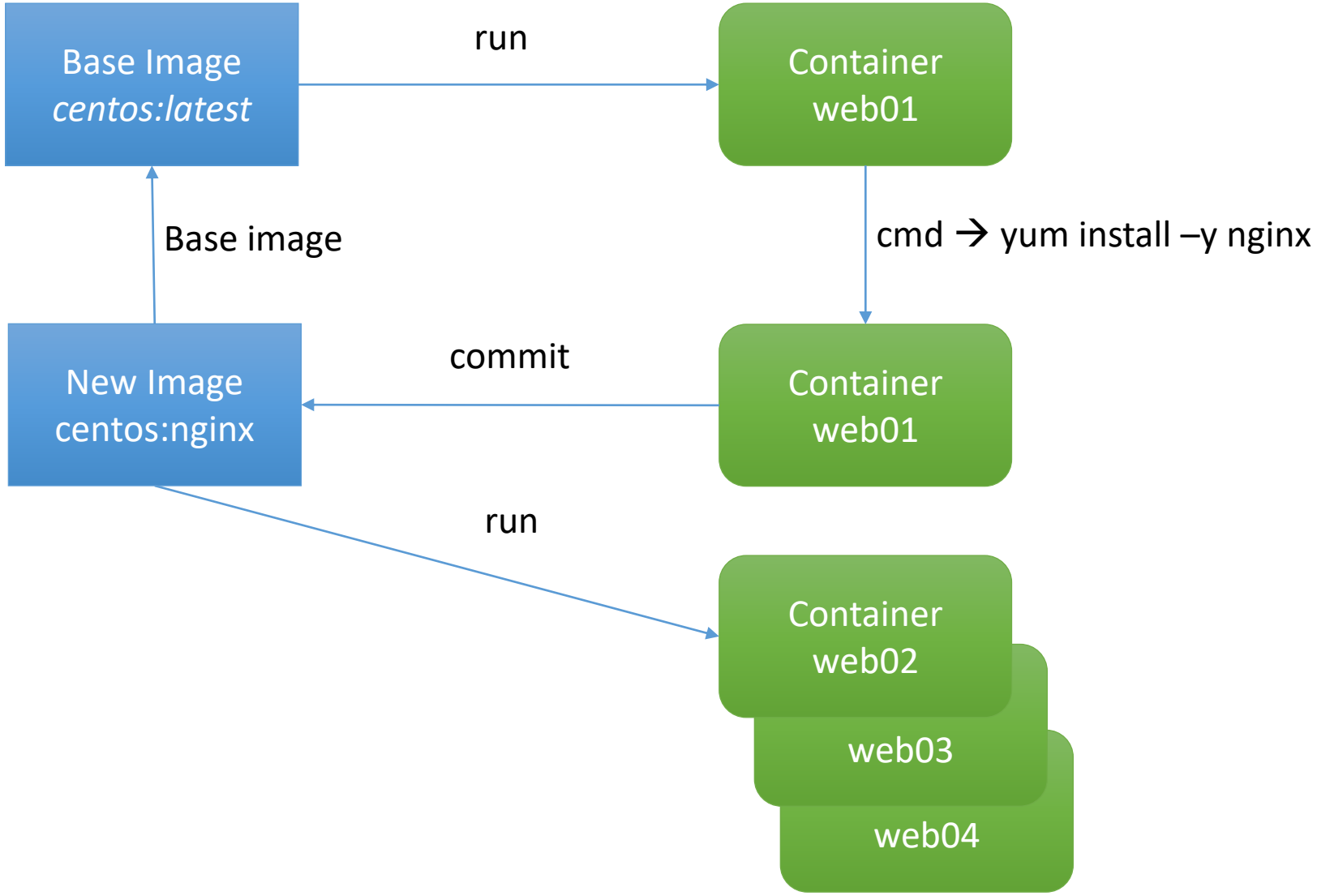
```
[root@master ~]# docker image --help

Usage:  docker image COMMAND

Manage images

Options:
      --help   Print usage

Commands:
  build       Build an image from a Dockerfile
  history     Show the history of an image
  import      Import the contents from a tarball to create a filesystem image
  inspect     Display detailed information on one or more images
  load        Load an image from a tar archive or STDIN
  ls          List images
  prune       Remove unused images
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rm          Remove one or more images
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

Run 'docker image COMMAND --help' for more information on a command.
```

Lab – Working with Images

Getting an Image

- Download from Docker Hub or Private Registry
- Commit the R/W container layer as a new R/O image layer
- Create Using **Dockerfile**
- Import a Tarball into Docker as a standalone base layer
- Load an image from the Tarball.

Image vs Container



Committing Docker Images

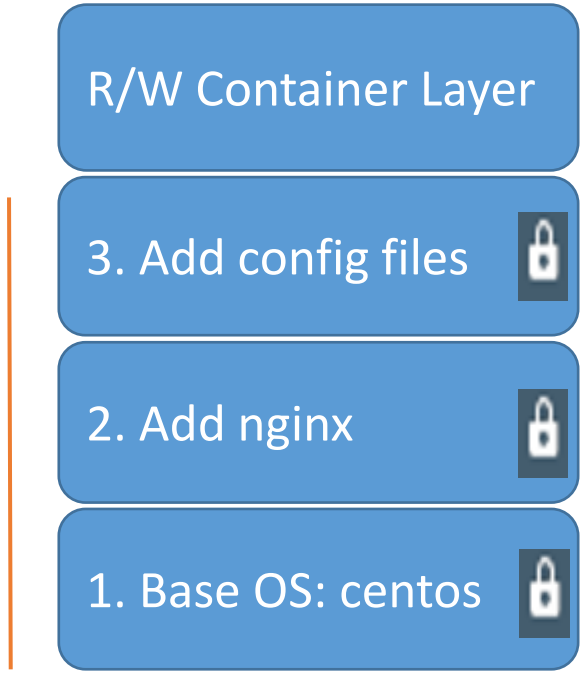
- Create a new image from a container's changes
- **`docker container commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]`**
- Options

-a, --author string	Author (e.g., "John Hannibal Smith <hannibal@a-team.com>")
-c, --change value	Apply Dockerfile instruction to the created image (default [])
--help	Print usage
-m, --message string	Commit message
-p, --pause	Pause container during commit (default true)

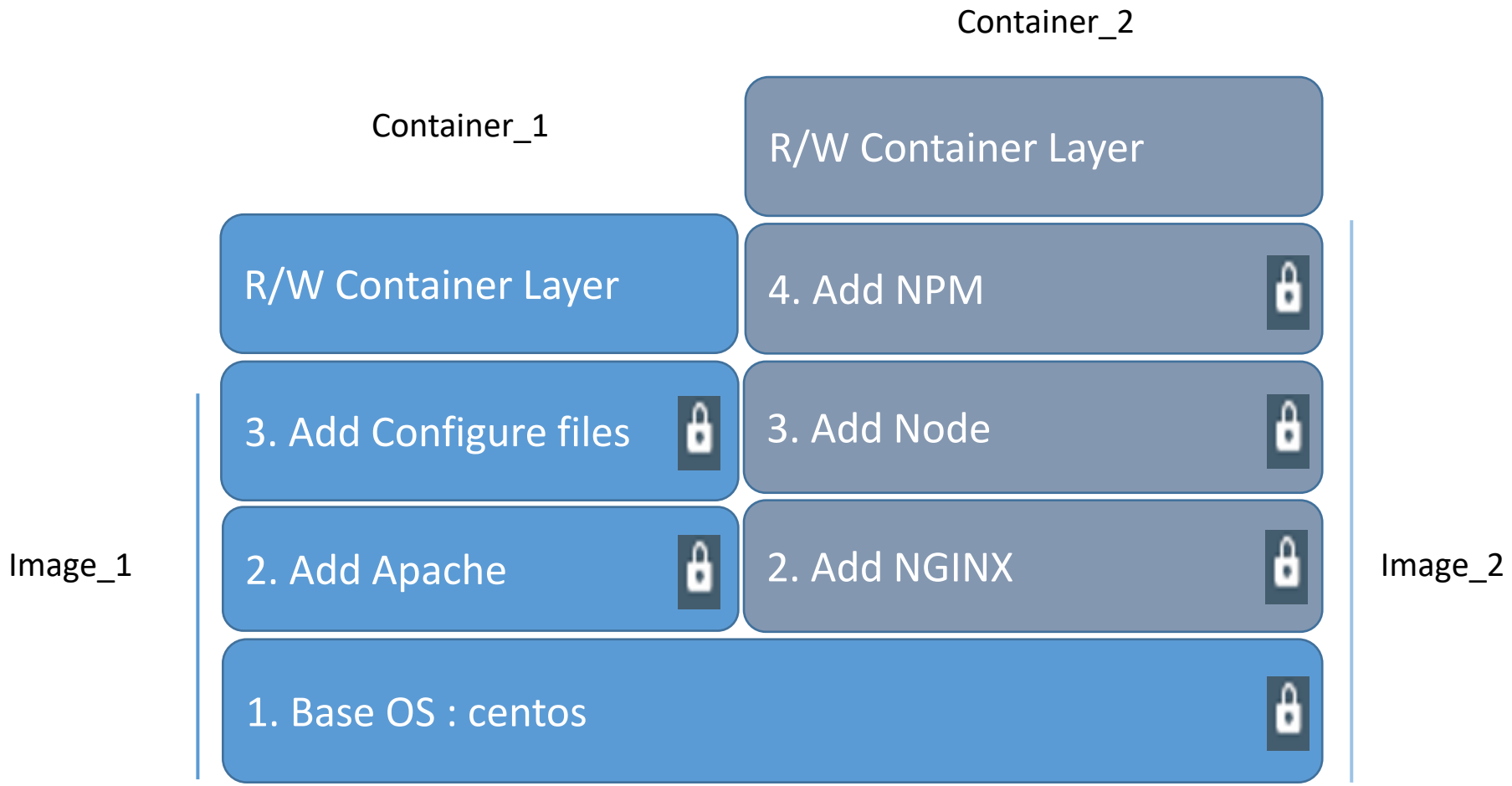
Image Layers

- Docker image is a filesystem for container process
- Made of stack of immutable layers
- Start with a Base image
- New layer for each change

Image = layered FS



Sharing Layers



What is Dockerfile?

- Docker can build images automatically by reading the instructions from a **Dockerfile**.
- A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.
- Using **docker image build** users can create an automated build that executes several command-line instructions in succession.

Dockerfile - Basic Commands

- **FROM** - The base image to use in the build. This is mandatory and must be the first command in the file
- **LABEL** - The LABEL instruction adds metadata to an image. A LABEL is a key-value pair
- **RUN** - Executes a command and save the result as a new layer
- **CMD** - The command that runs when the container starts
- **EXPOSE** - Opens a port for linked containers
- **ENV** - Sets an environment variable in the new container
- **ADD** or **COPY** - Copies a file from the host system onto the container
- **ENTRYPOINT** - Allows to configure container that will run as a executable
- **VOLUME** - Creates a shared volume that can be shared among containers or by the host machine
- **USER** - Sets the default user within the container
- **WORKDIR** - Set the default working directory for the container
- **ONBUILD** - A command that is triggered when the image in the Dockerfile is used as a base for another image

Dockerfile example

```
FROM centos
RUN yum -y install epel-release
RUN yum -y update
RUN yum -y install nginx
ADD index.html /usr/share/nginx/html/index.html
EXPOSE 80/tcp
CMD ["nginx", "-g daemon off;"]
```

What is Dockerfile?..

- **From** command defines base image
- Each subsequent command adds a layer
- **`docker image build .`** Builds image from Dockerfile

Build Context

- Create a separate Directory – Directory Archive
- Must contain all local files necessary for image
- Will omit anything listed in **.dockerignore**

```
$ docker image build -t demo .
Sending build context to Docker daemon  2.048kB
```

Examining the Build process

- Launch a new container
- Execute command in that container

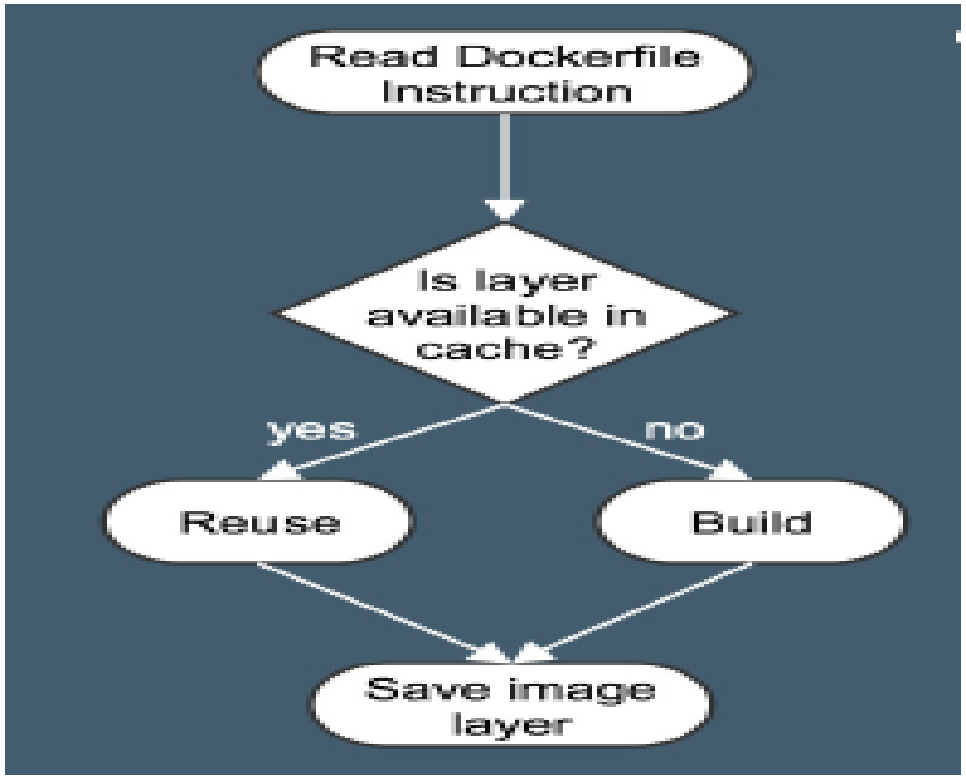
```
Step 2/3 : RUN apt-get update
 ---> Running in 2e8333703768
```

- Commit R/W layer to image
- Delete intermediate container

```
 ---> 0fac7902d4d4
Removing intermediate container 2e8333703768
```

Build Cache

- After completion, the resulting image layer is labeled with a hash of the content of all current image layers in the stack.



RUN vs CMD

- **RUN** is an image build step, the state of the container after a RUN command will be committed to the docker image.
- A Dockerfile can have many RUN steps that layer on top of one another to build the image.
- **CMD** is the command the container executes by default when you launch the built image.
- A Dockerfile can only have one CMD.
- The CMD can be overridden when starting a container with

`docker container run image other_command`.

COPY and ADD commands

- **COPY** copies files from build context to image:

```
COPY <src> <dest>
```

- **ADD** can also **untar** and **Fetch** URLs.

What are Docker Tags?

- Docker tags convey useful information about a specific image version/variant.
- They are aliases to the ID of your image which often look like this: f1477ec11d12.
- It’s just a way of referring to your image.
- A good analogy is how Git tags refer to a particular commit in your history.

Tag an Image

- Tag the image so that it points to your registry
  - **`docker image tag ubuntu localhost:5000/myfirstimage`**
- Push an image
  - **`docker image push localhost:5000/myfirstimage`**
- Pull an image
  - **`docker image pull localhost:5000/myfirstimage`**

Lab – Building an Image