Table of contents

Docker Engine

- Docker Engine is an open source containerization technology for building and containerizing your applications. Docker Engine acts as a client-server application with:
  - A server with a long-running daemon process dockerd.
  - APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon.
  - A command line interface (CLI) client docker.

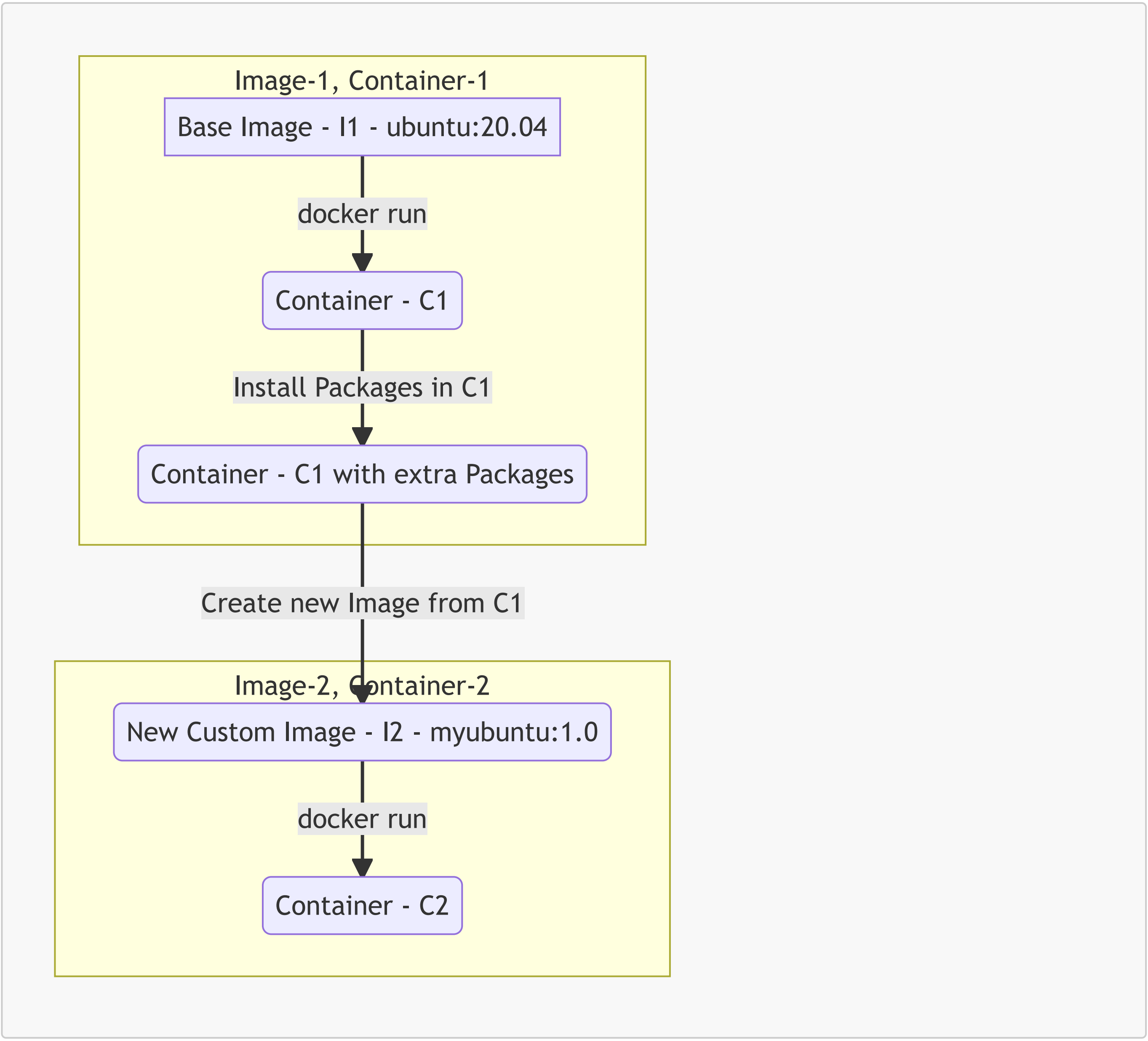
Creating Docker Images

1. Using container

- **I1 - ubuntu 20.04 > C1 > install wget or any package in C1 > I2-Custom image > C2 from I2**
- Check existing images and run a container using image.
  - Use ubuntu image as base image , install wget inside the container , then create an image from runnning container.

    - This setup is not preferred in a deployment environment.

--



--

```
sudo docker images
sudo docker run --name=base-image-container -i -t ubuntu:20.04 /bin/bash
```

- Run below commands inside container

```
whereis wget
apt-get update && apt-get install -y wget
whereis wget
```

- From another linux shell, use below `docker diff` command to check changes made in the above container

```
sudo docker diff <CONTAINER_NAME>
```

--

- To create a new image from exising running container using **docker commit**

```
docker images
[root@ip-172-31-25-98]# docker images
REPOSITORY                 TAG       IMAGE ID       CREATED         SIZE
ubuntu                     20.04     d5447fc01ae6   2 weeks ago     72.8MB

sudo docker commit base-image-container ubuntudocker/ubuntu_wget
sudo docker images
[root@ip-172-31-25-98]# docker images
REPOSITORY                 TAG       IMAGE ID       CREATED         SIZE
ubuntudocker/ubuntu_wget   latest    273e6395f1f0   7 seconds ago   120MB
ubuntu                     20.04     d5447fc01ae6   2 weeks ago     72.8MB
sudo docker run --name=C2-new-image-container -i -t
ubuntudocker/ubuntu_wget:latest /bin/bash
whereis wget
[root@ip-172-31-25-98 ec2-user]# docker run --name=C2-new-image-container -i -t
ubuntudocker/ubuntu_wget:latest /bin/bash
root@a0a92813e3aa:/# whereis wget
wget: /usr/bin/wget /usr/share/info/wget.info.gz
```

--

- Below is the information for the details related to `docker images` command
  - `REPOSITORY`: This is the name of the repository or image.
  - `TAG`: This is the tag associated with the image
  - `IMAGE ID`: Every image is associated with a unique ID.
  - `CREATED`: Indicates the time when the image was created.
  - `SIZE`: Highlights the virtual size of the image.

```
[root@ip-172-31-20-206 ~]# docker images
REPOSITORY                 TAG       IMAGE ID       CREATED              SIZE
ubuntudocker/ubuntu_wget   latest    519cc9e65e5a   About a minute ago   120MB
ubuntu                     20.04     d5447fc01ae6   2 weeks ago          72.8MB
```

2.Using Dockerfile

- A **Dockerfile** is a script that consists of instructions to build Docker images which can then be used to deploy a Docker container.
- A Dockerfile is a text document that contains all the commands/set of instructions a user could call on the command line to create an image.
- The commands within the **Dockerfile** can be configured to use specific versions & dependencies.
- Once a **Dockerfile** is written, you can use the **docker build** command to generate a Docker image based on the Dockerfile's instructions.

--

- Create a Docker image using DockerFile **touch Dockerfile**

- Copy the below content into `Dockerfile`

```
# Base Image will be as below
FROM ubuntu:20.04
ARG SDLC_ARG
ENV SDLC_ENV=${SDLC_ARG}
RUN echo "ARG value for SDLC_ARG is $SDLC_ARG"
RUN echo "ENV value for SDLC_ENV is $SDLC_ENV"
# To Set a default value ARG SDLC_ENV=test
ENV DEBIAN_FRONTEND=noninteractive
# Install dependencies
RUN apt-get update
RUN apt-get install -y apache2
RUN apt-get install -y apache2-utils
# Replace content of Apache Home Page
RUN echo "Docker Image created using Dockerfile for $SDLC_ENV" >
/var/www/html/index.html
# Expose Container Port
EXPOSE 80
# Execute command at container launch
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

- In yum distribution, **httpd** is webserver, in debian the httpd is ideally known as **apache2**

--

- The `DEBIAN_FRONTEND=noninteractive` instruction ensures that the subsequent RUN apt-get commands execute without requiring additional user input when building images.
- A Dockerfile typically contains of `INSTRUCTION` and `ARGUMENT`. **INSTRUCTTION** like `FROM`, `RUN` and **ARGUMENT** like `ubuntu:20.04`, `apt-get install -y apache2`.

--

Dockerfile Definitions

- ***Basic Definitions***
  - **FROM**: Define the base image, such as ubuntu or debian, used to start the build process. `Required` for each Dockerfile.

- ***Variables***
    - **ENV**: Set environment variables that persist when the container is deployed.
    - **ARG**: It is only available during the build of a Docker image (`RUN` etc), not after the image is created and containers are started from it. It is used to Pass a variable during Image build. Variable set as ARG does not persist when container is deployed from the image.

--

- ***Command Execution***
    - **RUN**: Execute commands, such as package installation commands that runs on a new intermediate container.
    - **CMD**: Execute a specific command within the container that is deployed with the image. Only one is used per Dockerfile.
    - **ENTRYPOINT**: Set a default application to be used every time a container is deployed with the image. Only one is used per Dockerfile.
    - **WORKDIR**: Set the container path where subsequent Dockerfile commands are executed.

--

- ***Data Management***
    - **ADD**: Copy files from a source to the image's filesystem at the set destination along with remote URL handling
    - **COPY**: Similar to ADD but without automatic tarball and remote URL handling.

--

- ***Networking***
    - **EXPOSE**: Expose a specific port to enable networking between the container and the Host.

--

- Use below command to build an image locally using Dockerfile.

```
ls -ltr
docker images
docker build -t docker-apache2 . --build-arg SDLC_ARG="dev"
docker build -t docker-apache2 -f Dockerfile --build-arg SDLC_ENV="dev"
docker images

[root@ip-172-31-20-206 docker_images] docker images
REPOSITORY                  TAG       IMAGE ID       CREATED          SIZE
docker-apache2              latest    008724744471   3 minutes ago    226MB
ubuntudocker/ubuntu_wget    latest    519cc9e65e5a   20 minutes ago   120MB
ubuntu                      20.04     d5447fc01ae6   2 weeks ago      72.8MB
```

- `.` in the above command specifies the path of the `Dockerfile`
- This Dockerfile uses the `ubuntu:20.04` image.
- The `RUN` instructions will simply run the linux commands for that image and then write the some content to the web server's document root.

- Run a container with the newly built image and keep docker running in detached mode with `-d` parameter
- The **-p 80:80** option maps the exposed port 80 on the container to port 80 on the EC2 host system **-p <HOST_PORT>:<CONTAINER_PORT>**

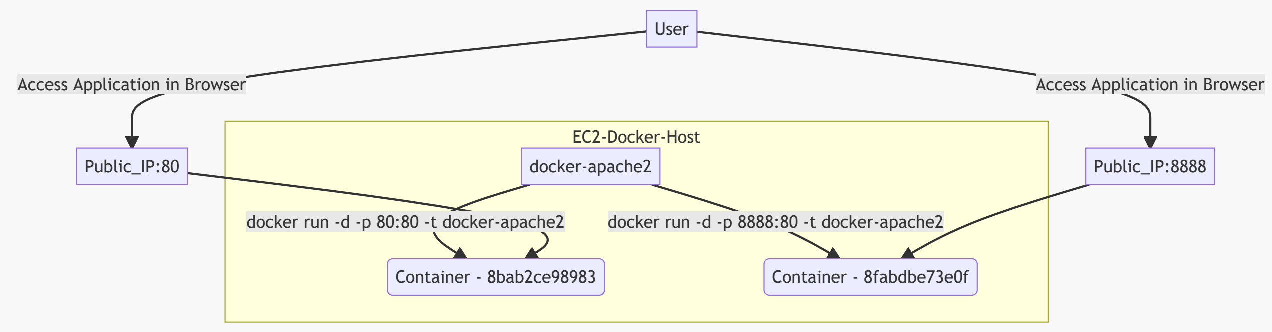
--

```
netstat -nltp
docker run -d -p 80:80 -t docker-apache2
# to get into container bash session
docker exec -it container-id bash
docker ps
# Check for Config Env values set for this container using inspect command
docker inspect <CONTAINER_NAME>

CONTAINER ID    IMAGE           COMMAND                  CREATED          STATUS
PORTS                NAMES
86c6b6c2212e    docker-test     "apache2ctl -D FOREG…"   2 seconds ago    Up 1
second      0.0.0.0:80->80/tcp   relaxed_curie

netstat -nltp
# Access the container application in browser
# Launch another container on another port with same image
docker run -d -p 8888:80 -t docker-apache2
docker ps
#
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS                  NAMES
8bab2ce98983        docker-apache2      "apache2ctl -D FOREG…"   20 seconds ago
Up 19 seconds       0.0.0.0:8888->80/tcp   amazing_nash
8fabdbe73e0f        docker-apache2      "apache2ctl -D FOREG…"   2 minutes ago
Up 2 minutes        0.0.0.0:80->80/tcp     loving_goodall
```

```
docker run --name <container_name> -p <host_port>:<container_port> -d
<container_image_label or ID>
```

- Test the content of the html file in the browser using public ip, make sure you have port 80 open in security group.

--


User
Access Application in Browser
Access Application in Browser
EC2-Docker-Host
Public_IP:80
docker-apache2
Public_IP:8888
docker run -d -p 80:80 -t docker-apache2
docker run -d -p 8888:80 -t docker-apache2
Container - 8bab2ce98983
Container - 8fabdbe73e0f

--

```
curl localhost:80
curl localhost:8888
```

- Checking Docker Image Size

```
docker image ls
docker image inspect <IMAGE_NAME>
```

- Docker images are stored at `/var/lib/docker/overlay2`

```
du -sh -m /var/lib/docker/overlay2
```

- We can use the `docker rmi` command to remove the images:

```
docker rmi <IMAGE_NAME>
du -sh -m /var/lib/docker/overlay2
```

Docker Image Registry

1.Amazon ECR

- **Registry**: An ECR registry is provided to each AWS account; we can create image repositories in our registry and store images in them. This is region specific.

- **Repository**: An ECR image repository contains our Docker images.
- **Authorization token**: The Docker client must authenticate to Amazon ECR registries as an **AWS** user before it can push and pull images. The AWS CLI **get-login-password** command provides us with authentication credentials to pass to Docker.
- **Image**: We can push and pull container images to our repositories.

--

- Attach a role to ec2 instance with ECR Permissions to create ECR Repository and push images to ECR Repository
  - `arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryFullAccess`
- Create a ECR repository using aws cli:

```
# aws ecr-public describe-repositories --region ap-south-1
aws ecr describe-repositories --region ap-south-1
# aws ecr-public create-repository --repository-name docker-testing
aws ecr create-repository --repository-name docker-testing --region ap-south-1
```

- Use the following steps to authenticate and push an image to your repository.
  - Retrieve an authentication token and authenticate your Docker client to your registry.
  - Docker login username for ECR will always be **AWS**

```
# aws ecr-public get-login-password | docker login --username AWS --password-stdin
<ACCOUNT_ID>.dkr.ecr.<REGION_NAME>.amazonaws.com
aws ecr get-login-password --region ap-south-1

aws ecr get-login-password --region ap-south-1 | docker login --username AWS --
password-stdin <ACCOUNT_ID>.dkr.ecr.<REGION_NAME>.amazonaws.com

# Login Succeeded
```

--

- After the **docker build** is completed, check for local docker images present and tag your image so you can push the image to this repository:

> **docker tag** is what we use to define which repository an image will be pushed to, and `docker push` is the command that does the upload itself.

```
docker images

[root@ip-172-31-20-206]# docker images
REPOSITORY                 TAG       IMAGE ID       CREATED             SIZE
docker-apache2             latest    008724744471   58 minutes ago      226MB
ubuntudocker/ubuntu_wget   latest    519cc9e65e5a   About an hour ago   120MB
ubuntu                     20.04     d5447fc01ae6   2 weeks ago         72.8MB
```

```
docker tag docker-apache2:latest <ACCOUNT_ID>.dkr.ecr.
<REGION_NAME>.amazonaws.com/docker-testing:latest

docker images
[root@ip-172-31-20-206]# docker images
REPOSITORY                                                 TAG       IMAGE ID
CREATED              SIZE
082923708139.dkr.ecr.ap-south-1.amazonaws.com/docker-testing   latest
008724744471   59 minutes ago      226MB
docker-apache2                                             latest
008724744471   59 minutes ago      226MB
ubuntudocker/ubuntu_wget                                   latest
519cc9e65e5a   About an hour ago   120MB
ubuntu                                                     20.04
d5447fc01ae6   2 weeks ago         72.8MB
```

> Note that the image id remains the same between the 2 versions of the image. This is ideally the same image, just with 2 references/tags.

--

- Run the following command to push this image to your newly created AWS repository:

```
docker push 01234567890.dkr.ecr.ap-south-1.amazonaws.com/docker-testing:latest
```

- Check whether docker image is available in ECR using AWS Console OR CLI

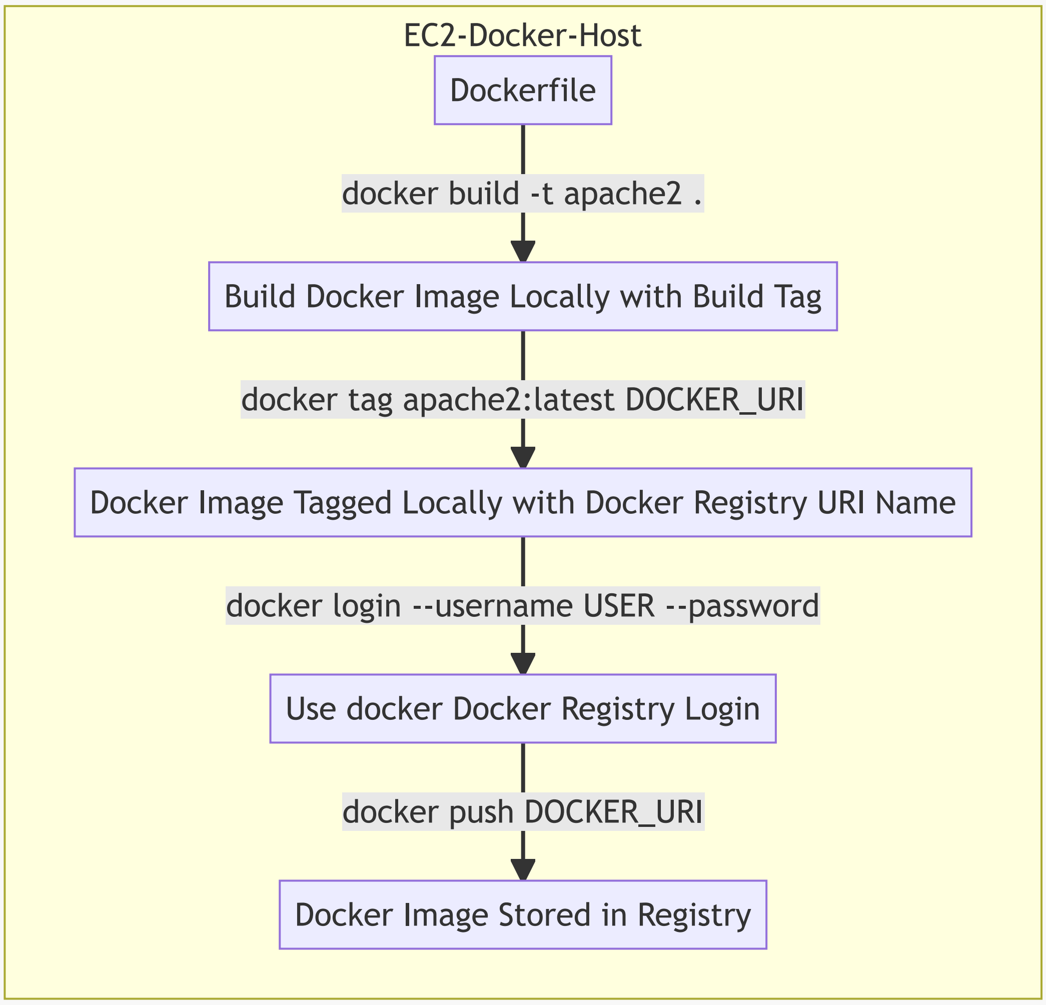
```
# aws ecr-public describe-repositories --region ap-south-1
aws ecr describe-images --repository-name docker-testing --region ap-south-1
```

--

- ECR Pricing : https://aws.amazon.com/ecr/pricing/
- To create ECR public repositories:
  - Update the aws cli version : https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html
  - Assign `arn:aws:iam::aws:policy/AmazonElasticContainerRegistryPublicFullAccess` to EC2 IAM Role.
  - Execute below command to create a Public ECR Repo.

```
[ec2-user@ip-172-31-24-162 ~]$ /usr/local/bin/aws ecr-public create-
repository --repository-name docker-testing-public --region us-east-1
{
    "repository": {
        "repositoryArn": "arn:aws:ecr-
public::082923708139:repository/docker-testing-public",
        "registryId": "082923708139",
        "repositoryName": "docker-testing-public",
        "repositoryUri": "public.ecr.aws/c2j4o3n4/docker-testing-public",
        "createdAt": "2022-09-17T02:51:52.296000+00:00"
    },
    "catalogData": {}
}
```

--


EC2-Docker-Host
Dockerfile
docker build -t apache2 .
Build Docker Image Locally with Build Tag
docker tag apache2:latest DOCKER_URI
Docker Image Tagged Locally with Docker Registry URI Name
docker login --username USER --password
Use docker Docker Registry Login
docker push DOCKER_URI
Docker Image Stored in Registry

--

- To Delete the ECR Repository use below command

```
# aws ecr-public delete-repository --repository-name docker-testing
aws ecr delete-repository --repository-name docker-testing --region ap-south-1 --
force
```

> As part of the AWS Free Tier, new Amazon ECR customers get 500 MB-month of storage for one year for your private repositories. As a new or existing customer, Amazon ECR offers you 50 GB-month of

> always-free storage for your public repositories.

--

Amazon ECR Public Gallery

- Navigate to Amazon ECR Public Gallery to view ECR Images.

2.Docker Hub

- `Docker Hub` is a service provided by Docker for hosting, finding, and sharing Docker Repositories.
- A Docker repository can be **public** or **private**.
- The Docker Hub and other third party repository hosting services are called **registries**.
- A **registry** has many **repositories**, while a **repository** has many **different versions** of the same image.

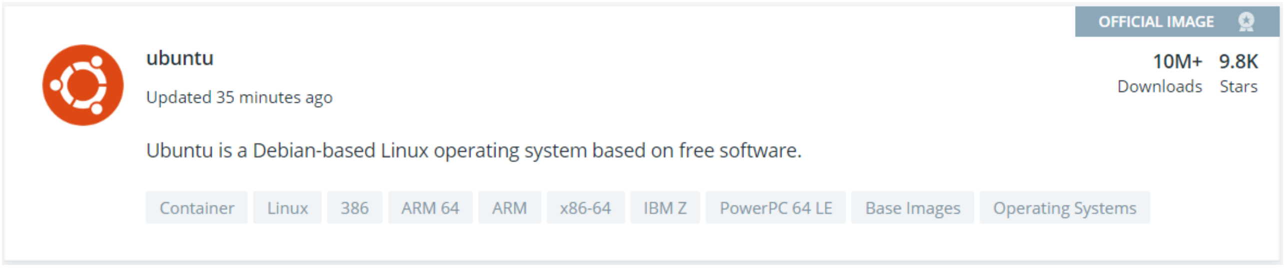
--

Official and Unofficial Docker Images

- On **Docker Hub**, there are two kinds of images - **official** and **unofficial**.
- Official images are trusted and optimized.
    - They have clear documentation, promote best practices, and are designed for the most common use cases.
- An **unofficial image** is any image that is created by a user.
- Docker Hub follows some standards so that both can easily be identified.
- Official images contain only the **<image_name>** as its image name but unofficial images have the syntax as **username/image_name**.

--

- Also, the official image has official written in the listing as per below screenshot.



OFFICIAL IMAGE

ubuntu

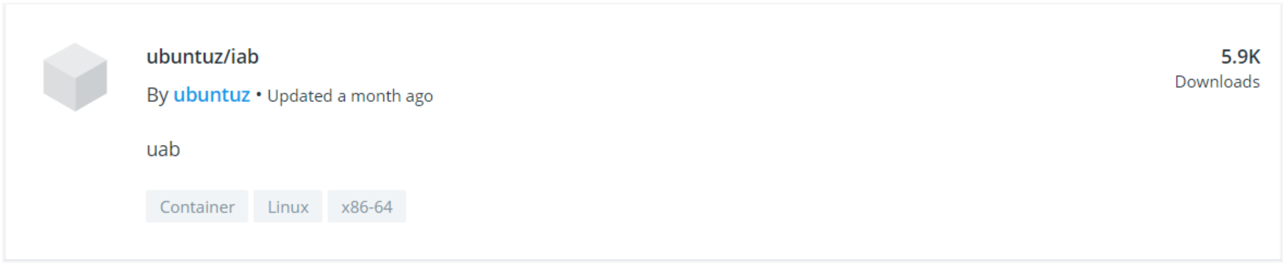
Updated 35 minutes ago

10M+ Downloads 9.8K Stars

Ubuntu is a Debian-based Linux operating system based on free software.

Container Linux 386 ARM 64 ARM x86-64 IBM Z PowerPC 64 LE Base Images Operating Systems

- Unofficial Image by User `ubuntuz`



ubuntuz/iab

By ubuntuz • Updated a month ago

5.9K Downloads

uab

Container Linux x86-64

--

- Sign Up to create a Docker Hub Acocunt on https://hub.docker.com/

- Login to docker from CLI using **docker login** command.
    - Execute this command from a Shell or Terminal where docker is installed.

```
docker login --username <DOCKER_USERNAME>

#Login with your Docker ID to push and pull images from Docker Hub. If you don't
have a Docker ID, head over to https://hub.docker.com to create one.
# Username:
# Password:
# WARNING! Your password will be stored unencrypted in /root/.docker/config.json
```

- Once Docker Login is successful in Local Shell/Terminal,Tag the local image with a name matching to your profile name with command.

```
docker tag docker-apache2:latest <Dockerhub_Username>/docker-apache2:latest
```

- Check the local docker images again to verify the tag.

```
docker images
```

--

- Push the docker image to docker hub using `docker push <username>/ image:tag`

```
docker push <Dockerhub_Username>/docker-apache2:latest
```

- Validate whether Image is available in Docker Hub Account from browser.
- Modify the **Dockerfile**
- Build the new image with a new tag
- Use docker tag to add a URI Tag
- Use docker push

Docker ECR Push Shell Script

- The previously executed steps are manual executions of the commands, if the image creation has to be done each time there is change in **application code or Dockerfile**, above steps will be required to executed multiple times.
- Below is the **shell script** that will create docker image using **Dockerfile** and push the **Docker Image in ECR Repo**.

--

- Create a shell script file **docker_ecr_push.sh** with code.
- Keep `Dockerfile` in same path as above shell script.

```
#!/bin/bash
# This script shows how to build the Docker image and push it to ECR
# The argument to this script is the image name. This will be used as the image on the local machine and combined with the account and region to form the repository name for ECR.
image=$1
region=$2
image_version=$3

echo "value of image is $image"
if [ "$image" == "" ]
then
    echo "Usage: $0 <image-name> not specified"
    exit 1
fi

# Get the account number associated with the current IAM credentials
account=$(aws sts get-caller-identity --query Account --output text)

# $? will if previously executed command was successful.
if [ $? -ne 0 ]
then
    exit 255
fi

# region="ap-south-1"
ecr_repo_name=$image"-ecr-repo"
image_name=$image-$image_version
# If the repository doesn't exist in ECR, create it.
echo "Checking ECR Repo with name $ecr_repo_name"
# || means if the first command succeed the second will never be executed
aws ecr describe-repositories --repository-names ${ecr_repo_name} --region $region
|| aws ecr create-repository --repository-name ${ecr_repo_name} --region $region

# Get the login command from ECR and execute docker login
aws ecr get-login-password --region $region | docker login --username AWS --
password-stdin ${account}.dkr.ecr.${region}.amazonaws.com

# Build the docker image locally with the image name and then push it to ECR with the full name.

docker build -t ${image_name} .
fullname="${account}.dkr.ecr.${region}.amazonaws.com/${ecr_repo_name}:$image_name"
echo "fullname is $fullname"

# Tag the locally created docker image with the ECR Repo URI
echo "Tagging the Local Image ${image_name} with ${fullname}"
```

```
docker tag ${image_name} ${fullname}

# docker images
echo "Pushing the Docker Image with URI ${fullname}"
docker push ${fullname}

if [ $? -eq 0 ]
then
        echo "Docker Push Event is successfull with ${fullname}"
else
        echo "Docker Push Event failed."
fi
```

- As this shell script accepts the Command Line Arguments as: SDLC_ENVIRONMENT, image, region
- Use below command to run the shell script:

```
bash docker_ecr_push.sh testimage ap-south-1 1.1
```

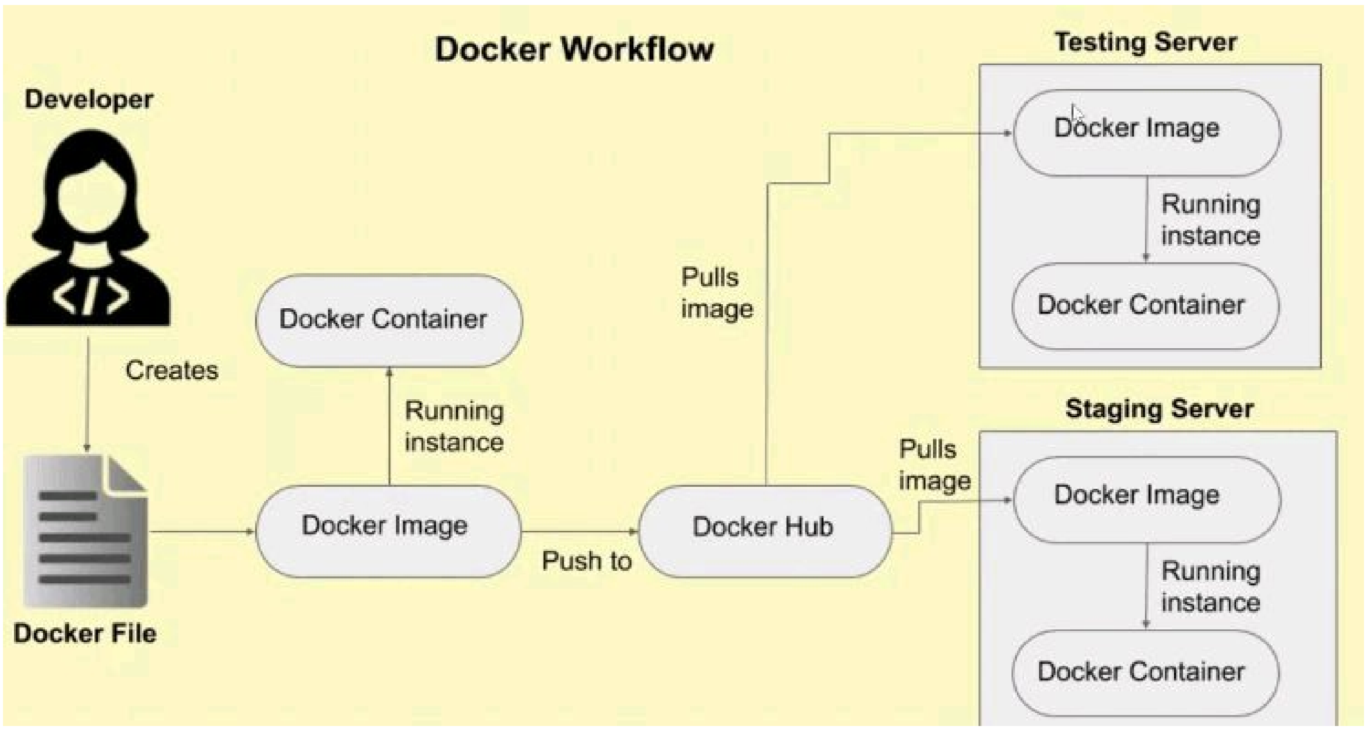
--

Docker Hub Image Push Shell Script Assignment

- Refer above shell script and convert above shell script to take users input as Docker Hub Account Name, Image Name, and build, tag and push image into Docker Hub Account.

--

Docker Image and Containers



Set environment variables for docker containers

- Use the **-e, --env, and --env-file** flags to set environment variables in the container you're running, or overwrite variables that are defined in the Dockerfile of the image you're running.
- You can define the variable and its value when running the container:

```
docker run -e MYVAR1=MYVALUE1 --env SDLC_ENV=dev -it ubuntu bash
echo $SDLC_ENV

docker run -e MYVAR1=MYVALUE1 --env SDLC_ENV=qa -it ubuntu bash
echo $SDLC_ENV
```

- Here, the SDLC_ENV as a environment variable value

A) You can define the variable and its value when running the container:

```
docker run --env VAR1=value1 --env VAR2=value2 -it ubuntu printenv | grep VAR
```

--

B) You can also load the environment variables from a file.

- This file should use the syntax **variable=value**
- Create **env.list** file locally

`env.list`

```
VARIABLE1=val1
VARIABLE2=val2
VARIABLE3=val3
USER=ec2-user
```

- Execute the below command to Run a Docker Container that will have environment variables set inside the container from the **env.list** file.

```
docker run --env-file env.list -it ubuntu bash

root@9851f1bb4134:/# printenv
VARIABLE1=val1
VARIABLE2=val2
VARIABLE3=val3
USER=ec2-user
```

--

C) You can also use variables that you've exported to your local environment:

```
export VAR1=value1
export VAR2=value2
docker run --env VAR1 --env VAR2 ubuntu env | grep VAR
```

Docker Stats Resources Consumption

- `docker stats` : Display a live stream of container(s) resource usage statistics

```
docker stats

CONTAINER ID   NAME               CPU %     MEM USAGE / LIMIT     MEM %     NET
I/O        BLOCK I/O      PIDS
ff70f0a5ae9f   great_bartik       0.00%     768KiB / 965.8MiB     0.08%     1kB /
0B      0B / 0B        1
c559ac5e30c9   frosty_newton      0.00%     772KiB / 965.8MiB     0.08%     1.33kB
/ 0B    0B / 0B        1
a20bf7d5fc8e   sad_jennings       0.00%     764KiB / 965.8MiB     0.08%     1.4kB
/ 0B     0B / 0B        1
03071a5fd70d   elastic_sinoussi   0.00%     780KiB / 965.8MiB     0.08%     1.4kB
/ 0B     0B / 0B        1
d5086de0b615   stupefied_dirac    0.00%     3.113MiB / 965.8MiB   0.32%     1.84kB
/ 0B    3.18MB / 0B    1

docker run -d -p 8888:80 -t docker-apache2
docker stats

docker run --memory="20m" -d -p 8080:80 -t docker-apache2
docker stats
```

- Access the container on the browser with multiple request and check for memory usage.

--

Docker-Essential Commands

- Below are the list of `essential commands` for docker

Commands	Description
`docker version`	Show the Docker version information
`docker ps`	List all running containers
`docker ps -a`	List all containers `stopped`, `running`
`docker stop CONTAINER_ID`	Stop the container which is running

Commands	Description
`docker start CONTAINER_ID`	Start the container which is stopped
`docker restart CONTAINER_ID`	Restart the container which is running
`docker port CONTAINER_ID`	List port mappings of a specific container
`docker rm CONTAINER_ID or name`	Remove the stopped container
`docker rm -f CONTAINER_ID or name`	Remove the running container forcefully
`docker pull IMAGE_NAME:TAG`	Pull the image from docker hub repository

--

Commands	Description
`docker exec -it container-name /bin/bash`	Connect to linux container and execute commands in container
`docker attach container-name`	Connect to linux container and execute commands in container
`docker rmi image-id`	Remove the docker image
`docker login -u username -p password`	Login to docker hub
`docker logout`	Logout from docker hub
`docker stats`	Display a live stream of container(s) resource usage statistics
`docker info`	Display system-wide information
`docker push IMAGE_NAME:TAG`	Push the image from docker hub repository

--

CMD vs ENTRYPOINT

- `CMD` and `ENTRYPOINT` are `Dockerfile` instructions.
- Primary usage of `CMD` and `ENTRYPOINT` is used to run the executable when instantiating the image.
- Below are the key differences between the same.
    - CMD can pass default parameters to ENTRYPOINT if both are defined.

CMD

- The `CMD` directive allows to specify the default command executed by the container.
- This command runs when the container starts and no other command is specified for docker run.
- If docker run specifies another command, the default command specified by CMD will be ignored.
- If a Dockerfile has multiple CMD instructions, it only applies the instructions from the last one.

- `CMD` has three formats:
    - Exec format: `CMD ["executable","param1","param2"]`
    - `CMD ["param1", "param2"]`, this format is used in combination of **ENTRYPOINT** , to provide extra parameters
    - Shell format: `CMD command param1 param2`

--

ENTRYPOINT

- The `ENTRYPOINT` directive allows the container to run as an application or service.

- `ENTRYPOINT` looks similar to `CMD` in that both specify the command to execute and its parameters

- You `cannot override` an `ENTRYPOINT` when starting a container unless you add the `--entrypoint` flag.

- `ENTRYPOINT` has two formats:
    - Exec format: `ENTRYPOINT [“executable”, “param1”, “param2”]` This is the recommended format for `ENTRYPOINT`.
    - Shell format: `ENTRYPOINT command param1 param2`
    - The parameters in `ENTRYPOINT` are always used, while the extra parameters of `CMD` can be dynamically replaced when the container starts.

--

```
ENTRYPOINT ["/bin/echo", "Hello"]
CMD ["world"]
# Output
Hello world
```

- Note the shell format of `ENTRYPOINT` ignores any arguments provided by `CMD` or `docker run`.

```
FROM busybox
ENTRYPOINT echo hello
CMD world
# Output
hello
```

Best practices for writing Dockerfiles

- Follow this official documentation : https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

Docker Practice

Play With Docker Labs

- Login to Docker Hub Account.
    - Your First Linux Containers: In this lab you will explore the basics of running containers: pulling images from a registry, running an containerized application, and container instances and isolation.
    - Customizing Docker Images: Move on to building your own custom Docker images and explore the Docker concept of image layers and the Dockerfile.
    - Docker Linux: In this lab, we will look at some basic Docker commands and a simple build-ship-run workflow. We'll start by running some simple containers, then we'll use a Dockerfile to build a custom app. Finally, we'll look at how to use bind mounts to modify a running container as you might if you were actively developing using Docker.
- https://training.play-with-docker.com/ops-stage2/

IBM Cognitive Class

- Sign Up/Register here in **https://cognitiveclass.ai/**
- Complete the below Essentials and achieve a Badge.
    - Docker Essentials - Course + Badge

Reference

- everyday-hacks-docker
- play-with-docker