**Learning**

**Docker and Kubernetes**

**Through the Lens**

**of**

**Python Development**

Table of Contents

[1 General Introduction 3](#_Toc30320505)

[2 Glossary 3](#_Toc30320506)

[3 Docker 5](#_Toc30320507)

[3.1 Introduction 5](#_Toc30320508)

[3.2 Use Cases that illustrate the importance of using Docker 5](#_Toc30320509)

[3.3 Does Docker help us dispense with VMs , Docker versus VMs ? 5](#_Toc30320510)

[3.4 Alternatives to Docker 6](#_Toc30320511)

[4 Kubernetes 6](#_Toc30320512)

[4.1 Introduction 6](#_Toc30320513)

[4.2 Why is Kubernetes so useful, important and in the news these days ?. 7](#_Toc30320514)

[4.2.1 Use Cases that illustrate the importance of using Kubernetes 7](#_Toc30320515)

[5 Docker and Kubernetes together 7](#_Toc30320516)

[6 Why Docker and Kubernetes with Python development 7](#_Toc30320517)

[7 A complete CI/CD example using Docker, Kubernetes and Python 7](#_Toc30320518)

[8 Using Docker to run stuff on Windows 7](#_Toc30320519)

[8.1 Postgresql 7](#_Toc30320520)

[8.2 Redis 7](#_Toc30320521)

[8.3 NGINX 7](#_Toc30320522)

[8.4 Flask apps on Gunicorn 7](#_Toc30320523)

[9 Docker Hands on Tutorial 8](#_Toc30320524)

[10 Kubernetes Hands On Tutorial 14](#_Toc30320525)

[11 Ubuntu using Docker hands on Tutorial 20](#_Toc30320526)

# General Introduction

# Glossary

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Term** | **Description** | **Additional Reading** |
|  | Docker image | * A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. | <https://www.docker.com/resources/what-container> |
|  | Docker container | * A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. * A container is always derived from an image | <https://www.docker.com/resources/what-container>  (A container, in a way, is like a SandBox) |
|  | Virtualization |  |  |
|  | Hyper-V |  |  |
|  | Layer |  |  |
|  | Linux Containers |  |  |
|  | Container Orchestration |  |  |
|  | Virtual Machines |  |  |
|  | APT | Advanced Packaging Tool |  |
|  | Multi container Applications |  |  |
|  | Volumes | Volumes are directories (or files) that are outside of the default Union File System *[Docker’s own internal filesystem ]* and exist as normal directories and files on the host filesystem.  Another description –  A Volume is a a file system that lives on a host machine outside of any container.  Volumes are created and managed by Docker.  Volumes are:   * persistent * free-floating filesystems, separate from any one container * sharable with other containers * efficient for input and output * able to be hosted on remote cloud providers * encryptable * nameable * able to have their content pre-populated by a container * handy for testing | <https://docs.docker.com/storage/volumes/> |
|  | Docker Compose |  |  |
|  | Running Container |  |  |
|  | Image Developer |  |  |
|  | Dockerfile |  |  |
|  | Union File System |  |  |
|  | Mountpoint |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Docker

## Introduction

This document aims to introduce the reader to the use of Docker and Kubernetes (“DANK”) in modern software development. Generally speaking, people discuss DANK in the context of CI/CD, but the use is not just restricted to that, project teams can leverage the power of DANK for a variety of purposes.

Important Point to always remember - Docker is built to deploy applications, not machines

## Use Cases that illustrate the importance of using Docker

|  |  |  |
| --- | --- | --- |
| **#** | **Use Case** |  |
| 1 | A set of development machines need to access Postgresql database, but can’t install it |  |
| 2 | Windows environment – Need to use a software that runs only on LINUX |  |
| 3 | Need to provide a testing environment to multiple testers |  |
| 4 | Your application runs on Python 3.7 but the production environment uses Python 2 and this can’t be changed |  |
|  |  |  |
|  |  |  |

## Does Docker help us dispense with VMs , Docker versus VMs ?

A more appropriate response is “Can Docker work with VMs ?”. Don’t debate on which is a better choice. Focus on co-existence.

An interesting article to read is <https://www.docker.com/blog/containers-and-vms-together/>

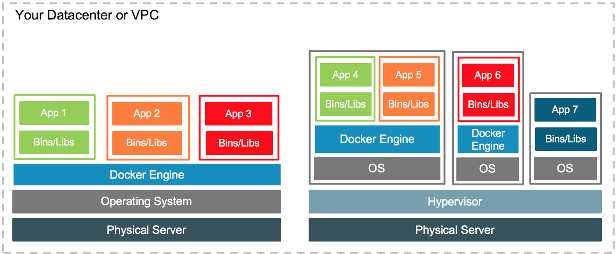


Figure – Source - <https://www.docker.com/blog/containers-and-vms-together/>

## Alternatives to Docker

# Kubernetes

## Introduction

Kubernetes is all about Container Orchestration. Well, “Orchestration” is a word that is extensively used / over-used in the industry to explain certain tasks/activities in a posh way.

In the context of Kubernetes, the platform delivers true orchestration.

Please note that Kubernetes is not a Docker replacement.

Docker is designed to create all kinds of applications that can be deployed as containers. The focus is on the creation of containerized apps.

Kubernetes, on the other hand, has been created for the following

* Enable the deployments of containers on clusters
* Scale the deployments
* Update the containers
* Troubleshoot issues

To know more - <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

## Why is Kubernetes so useful, important and in the news these days ?.

### Use Cases that illustrate the importance of using Kubernetes

# Docker and Kubernetes together

Docker and Kubernetes are frequently used together. In fact, these two platforms dominate the landscape in today’s Container Era.

# Why Docker and Kubernetes with Python development

# A complete CI/CD example using Docker, Kubernetes and Python

# Using Docker to run stuff on Windows

## Postgresql

## Redis

## NGINX

## Flask apps on Gunicorn

# Docker Hands on Tutorial

| **MAIN TOPIC** | **SUB TOPIC** | **DETAILS**  **/**  **(LINKS FOR FURTHER STUDY)**  **/**  **(FEEDBACK)**  **/**  **(SAMPLE PROGRAMS)** | **(CLASSROOM EXERCISES)**  **/**  **(ASSIGNMENTS)** | **TRACKING DATA** |
| --- | --- | --- | --- | --- |
| **OVERALL CONTEXT** | WHAT ARE YOU EXPECTING ? | <Update after feedback from the students> | **N/A** | DAY 1  (<=15 mins) |
|  | MY EXPECTATIONS FROM THE STUDENTS/YOU | * Be aware of the course content (*Have all of you gone through the course details [separate doc] ?)* * Do the class room exercises * Complete your assignments * Make notes *(I do it and it helps me)* * Don’t just nod your head to what I say. Digest it slowly. Stop me if I am going too fast | **N/A** | DAY 1  (<= 15 mins) |
|  | An overview of commonly used Docker commands | *#*  *# Docker Images, Containers, docker files,* layers , docker build  #  *# Use this command to search the repository*  docker search <>  *##*  *## Launch a container*  docker **run** -t -i ubuntu /bin/bash  *## get the container IDs*  docker **ps -all**  *##*  docker ps  *##*  1. docker ps -all (use this to get the container id)  2. docker **start** <container ID>  3. docker **attach** <container id>  *## Listing all containers running or otherwise*  docker container ls -a  *## Removing a container*  docker container rm  ## Run a command in a running container  docker exec  *## Attach local standard input, output, and error streams to a running container*  docker attach  ## Docker inspect |  |  |
|  | Going deeper into Docker container commands | <https://docs.docker.com/engine/reference/commandline/container/>  The main ones :-  *# start an existing docker container*  docker start [container id]  *# stop an existing docker container*  docker stop [container id]  *# Use docker attach to attach your terminal’s*  *# standard input, output, and error (or any*  *# combination of the three) to a running*  *# container using the container’s ID or name.*  *# This allows you to view its ongoing output or*  *# to control it interactively, as though the*  *# commands were running directly in your*  *# terminal*  docker **attach** <container id>  *# Kill an existing container*  docker kill [container id]  *# pause an existing docker container*  docker pause [container id]  *# Run a shell command inside a container*  docker exec {-ti} [container id] {command}  *# restart an existing docker container*  docker restart [container id]  *# Inspect an existing docker container*  docker container inspect [container id]  Note – There are many more commands. Please refer to the docker documentation |  |  |
|  |  |  |  |  |
|  | Persisting data – setting the stage for Volumes | There are two ways to persist data beyond the life of the container. One way is to *bind mount* a file system to the container.  With a bind mount, processes outside Docker also can modify the data.  <https://docs.docker.com/storage/bind-mounts/>  <https://docs.docker.com/storage/volumes/>  Volumes rather than bind mounts should be used for persisting data. |  |  |
|  |  |  |  |  |
|  | Understanding Docker Volumes | * Preferred way to save data over restarts of a docker container * Volumes can be visualized as directories (or files) that are outside of the default UFS and exist as normal directories and files on the host system * There are different types of Docker Volumes * Volume drivers allow volumes to be stored on remote hosts or cloud providers |  |  |
|  |  |  |  |  |
|  | Volume related commands | Key commands   * docker volume create * docker volume ls * docker volume inspect * docker volume rm * docker volume prune |  |  |
|  |  |  |  |  |
|  | Run multiple containers of the same image | This is an exercise that you can try out. The Hello World image can be used | **Why would you launch multiple containers of the same image ?**  **Create a volume that is used by the different containers of the same image** |  |
|  |  |  |  |  |
|  | For a single container, persisting data | This is achieved through the use of Volumes |  |  |
|  |  |  |  |  |
|  | Sharing data across multiple containers | This is achieved through the use of Volumes |  |  |
|  |  |  |  |  |
|  | Networking concepts in Docker | * Docker networking is used to establish communication between docker containers and the outside world via the host machine * This is an important topic in container architecture * This is of importance to network administrators * It is no longer a physical world, but one of VMs and Containers | **Q? Why are Docker networks important ?** |  |
|  |  |  |  |  |
|  | Docker Application Development | * <https://docs.docker.com/engine/reference/commandline/image_build/> * <https://www.scalyr.com/blog/create-docker-image/> * <https://www.youtube.com/watch?v=YFl2mCHdv24> |  |  |
|  |  |  |  |  |
|  | **Docker build**  **(Critical command)** | * A **Dockerfile** is a script that contains collections of commands and instructions that will be automatically executed in sequence in the docker environment for building a new docker image |  |  |
|  |  |  |  |  |
|  | Key DockerFile commands | **FROM**  **MAINTAINER**  **RUN**  **ADD** |  |  |
|  |  |  |  |  |
|  | Design considerations | * Slim, medium sized and fat builds * Layer caching * Multi-stage builds |  |  |
|  |  |  |  |  |
|  | Managing Application Data |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  | **Docker-compose**  **(Critical feature)** | <https://docs.docker.com/compose/>  <https://www.baeldung.com/docker-compose>  <https://www.youtube.com/watch?v=Qw9zlE3t8Ko>  Using Compose is basically a three-step process:   * Define your app’s environment with a Dockerfile so it can be reproduced anywhere. * Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment. * Run docker-compose up and Compose starts and runs your entire app. | **>> To the audience – Do you understand the Docker build process, Volumes, networking in Docker ?**  **These are pre-requisites for using docker compose which is much more difficult to understand and implement** |  |
|  |  |  |  |  |
|  | Run multiple containers (from different images) / Multi-container docker applications | This is achieved using Docker compose.  A thorough understanding of Docker compose is required to operate a multi container docker environment |  |  |
|  |  |  |  |  |
|  | Deploying a Docker image to another environment |  |  |  |

# Kubernetes Hands On Tutorial

| **MAIN TOPIC** | **SUB TOPIC** | **DETAILS**  **/**  **(LINKS FOR FURTHER STUDY)**  **/**  **(FEEDBACK)**  **/**  **(SAMPLE PROGRAMS)** | **(CLASSROOM EXERCISES)**  **/**  **(ASSIGNMENTS)** | **TRACKING DATA** |
| --- | --- | --- | --- | --- |
| **OVERALL CONTEXT** | WHAT ARE YOU EXPECTING ? | <Update after feedback from the students> | **N/A** | DAY 1  (<=15 mins) |
|  | MY EXPECTATIONS FROM THE STUDENTS/YOU | * Be aware of the course content (*Have all of you gone through the course details [separate doc] ?)* * Do the class room exercises * Complete your assignments * Make notes *(I do it and it helps me)* * Don’t just nod your head to what I say. Digest it slowly. Stop me if I am going too fast | **N/A** | DAY 1  (<= 15 mins) |
|  |  |  |  |  |
| **Deployments** | Deploy a Kubernetes cluster on a local machine | * Minikube is a tool that makes it easy to run Kubernetes locally. * Minikube runs a single-node Kubernetes cluster inside a Virtual Machine (VM) on your laptop for users looking to try out Kubernetes or develop with it day-to-day. |  |  |
|  | Using Minikube on Windows | NOTE – Minikube will allow you to start a cluster     * Minikube start * Minikube status * Minikube stop * Minikube delete * minikube dashboard |  |  |
|  | Kubectl commands | * kubectl version --client |  |  |
|  | On the cloud |  |  |  |
|  | On-prem datacenter |  |  |  |
|  | Managed Kubernetes cluster |  |  |  |
|  |  |  |  |  |

# Ubuntu using Docker hands on Tutorial

|  |
| --- |
| docker container run --interactive --tty --rm ubuntu bash |
|  |
| (sudo) apt-get update |
| (sudo) apt-get install apt-utils |
| (sudo) apt-get install python3.7 |
| apt install python3-pip |
| # difference between apt-get and apt  # apt list –upgradable  #  # dpkg -l | grep systemd  #  # |
| python3/3.7 -m pip install --upgrade pip |
| # for virtual environments  apt-get install python3.7-venv |
|  |
| python3.7 -m venv /home/workspaces/my-python-sandpit/muttli  / |
| Source activate  Deactivate  From the bin folder that resides within your virtual environment |
|  |
| Apt install nginx |
|  |
| apt install docker.io  systemctl start docker  systemctl enable docker |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

## 