School of Computer Science and Engineering

J Component report

Programme : B.Tech

Course Title : Virtualization

Course Code : CSE4011

Slot : E2+TE2

Title : Creating Virtual environment using Docker and Parallel Processing

Team Members: Navur Sai Charan - 20BCE1752

Rentala Naga Sai Ganesh - 20BCE1928

Siddenki Jayanth Reddy -20BCE1793

Faculty: Dr.Anusooya G

Sign:

Date:

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr.Anusooya G,** School of Computer Science and Engineering for her consistent encouragement and valuable guidance offered to us throughout the course of the project work.

TABLE OF CONTENTS

Team Members Contribution

Navur Sai Charan, Siddenki Jayanth Reddy - creating python virtual environment using docker and git bash terminal and developing modified docker architecture to create python virtual environment

Rentala Naga Sai Ganesh - creating parallel code to run on a python virtual environment and attaching the multiple containers to the vscode terminal

Abstract

The use of virtualization has become increasingly popular for developers looking to create isolated and reproducible software environments. Docker, a containerization platform, has emerged as a leading tool for creating these virtual environments. This project aims to explore the use of Docker for creating virtual environments and to demonstrate its benefits for software development.

The project will involve setting up Docker on a host machine and using it to create isolated containers for running various applications and services. The containers will be configured with specific versions of software dependencies, allowing developers to easily reproduce and test their applications in a controlled environment.

Introduction

In modern software development, it is becoming increasingly important to be able to quickly and easily set up and manage isolated and reproducible software environments. These virtual environments allow developers to test their applications in a controlled and consistent environment, without worrying about compatibility issues or conflicts with other software installed on their machine.

Docker is an open platform for developing, shipping, and running applications. This enables you to separate your applications from your infrastructure so you can deliver software quickly. With the Docker we can manage our infrastructure in the same ways you manage our applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly we can significantly reduce the delay between writing code and running it in production.

This project aims to explore the use of Docker for creating virtual environments for software development. The project will involve setting up Docker on a host machine and using it to create isolated containers for running various applications and services. The containers will be configured with specific versions of software dependencies, allowing developers to easily reproduce and test their applications in a controlled environment

Technology Learned for the Project

1)Docker:

Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime.

Containers are lightweight, portable, and efficient virtualization technologies that allow developers to run applications consistently across different environments. Docker containers are isolated from each other and from the host system, which makes them more secure and easier to manage. Docker provides tools for managing containers, images, and networks, making it easy to deploy and manage applications at scale.

Docker works by providing a standard way to run your code. Docker is an operating system for containers. Like how a virtual machine virtualizes (removes the need to directly manage) server hardware, containers virtualize the operating system of a server. Docker is installed on each server and provides simple commands you can use to build, start, or stop containers.

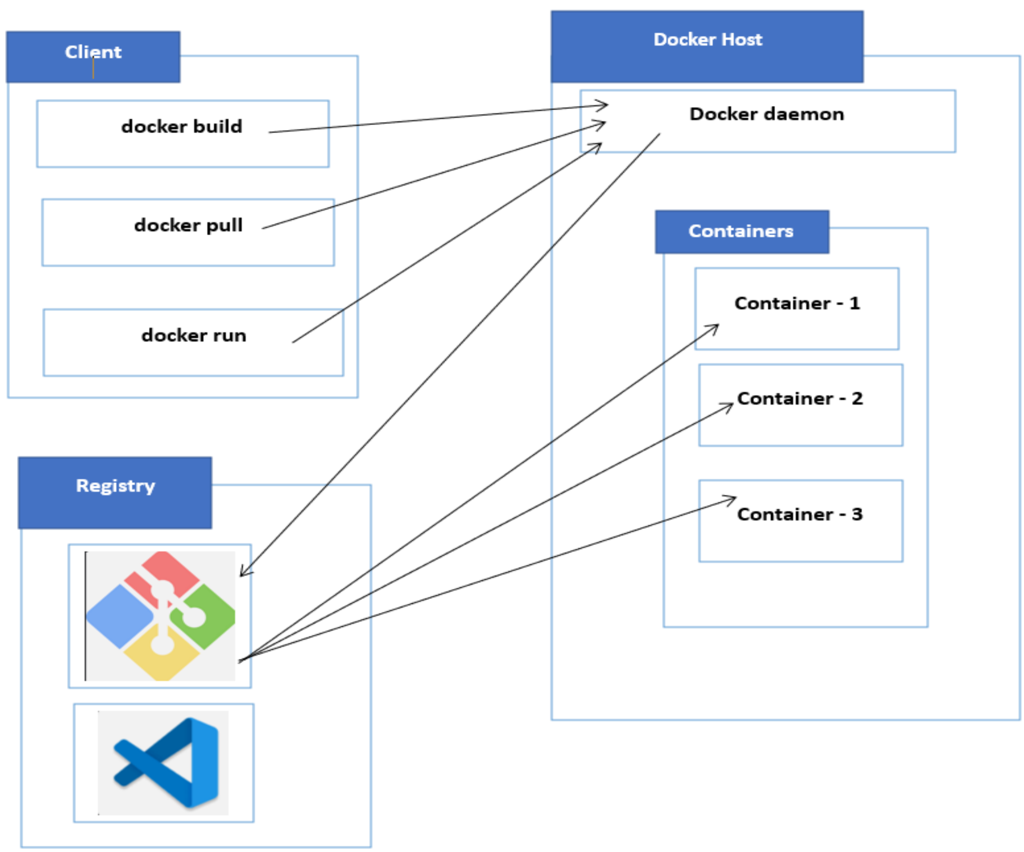
2) Python for Creating Virtual Environment

Python is a popular programming language used for developing a wide range of applications. Python is useful in creating virtual environments within Docker containers because it allows developers to create isolated environments for their applications. This is important because it ensures that the application will run consistently regardless of the underlying infrastructure. Python's built-in virtual environment module allows developers to create virtual environments that are isolated from the system's global Python installation. When creating a Docker container, developers can specify the Python version and any required dependencies in the container's configuration file. This ensures that the container has all the necessary packages and libraries to run the application.

3) **Parallel processing** allows the build process to be split into multiple threads or processes, which can be executed simultaneously on multiple CPU cores, reducing the overall build time. Parallel processing codes can also be used to optimize the deployment process of Docker containers. For example, developers can use parallel processing to deploy multiple Docker containers concurrently, which can be useful in high-performance computing or other applications where multiple instances of an application need to be deployed.

Project Architecture

The main component which is used in the project is the docker host. For Docker host it needs a host system with Docker installed to run Docker containers. Docker client uses commands and REST APIs to communicate with the Docker Daemon (Server). When a client runs any docker command on the docker client terminal, the client terminal sends these docker commands to the Docker daemon. Docker daemon receives these commands from the docker client in the form of command and REST API's request. The Docker client contains the following commands docker build, docker pull, docker run.



Working Model

Creating Virtual Environment:

• Install docker and Git bash where we will be executing docker commands in order to

create a python virtual environment.

• Commands used for creating virtual environment: -

➢ docker pull python

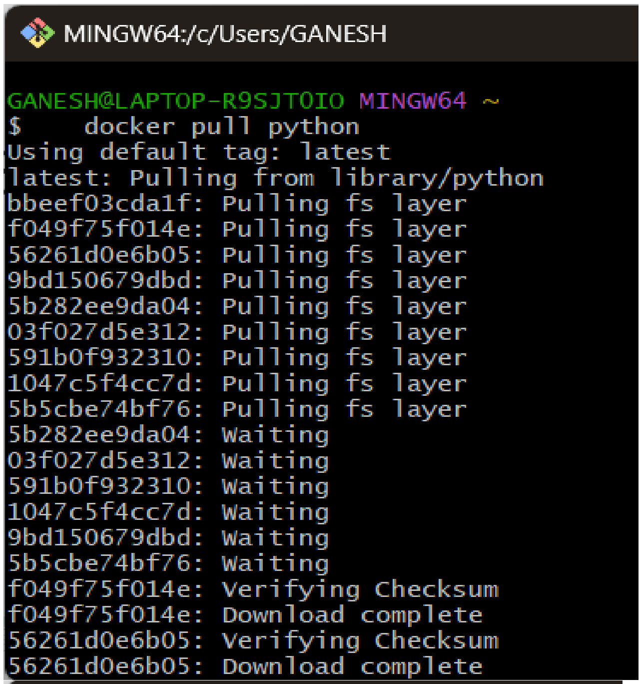
➢ docker pull ciscotestautomation/pyats

➢ docker ps

➢ docker run -d -i --name pyhtondevs python bash

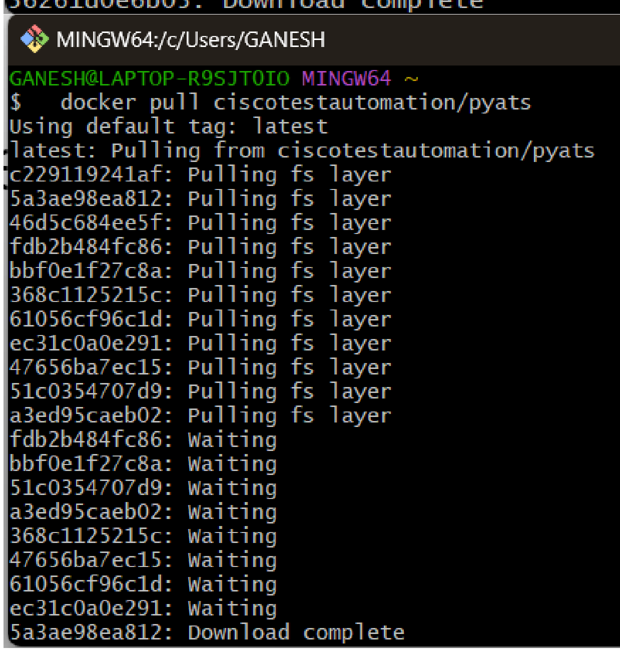
Explanation and Screenshots

Virtual Environment Creation:



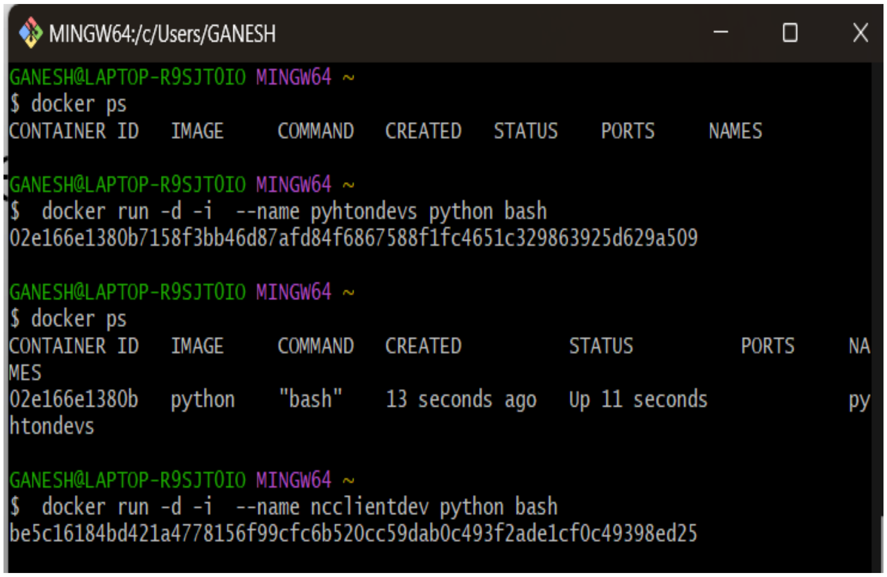
```
MINGW64:/c/Users/GANESH

GANESH@LAPTOP-R9SJTOIO MINGW64 ~
$    docker pull python
Using default tag: latest
latest: Pulling from library/python
bbeef03cda1f: Pulling fs layer
f049f75f014e: Pulling fs layer
56261d0e6b05: Pulling fs layer
9bd150679dbd: Pulling fs layer
5b282ee9da04: Pulling fs layer
03f027d5e312: Pulling fs layer
591b0f932310: Pulling fs layer
1047c5f4cc7d: Pulling fs layer
5b5cbe74bf76: Pulling fs layer
5b282ee9da04: Waiting
03f027d5e312: Waiting
591b0f932310: Waiting
1047c5f4cc7d: Waiting
9bd150679dbd: Waiting
5b5cbe74bf76: Waiting
f049f75f014e: Verifying Checksum
f049f75f014e: Download complete
56261d0e6b05: Verifying Checksum
56261d0e6b05: Download complete
```

docker pull is a Docker command that downloads a Docker image or a repository locally on the host from a public or private registry. In your case, docker pull python downloads the latest version of the Python image from Docker Hub

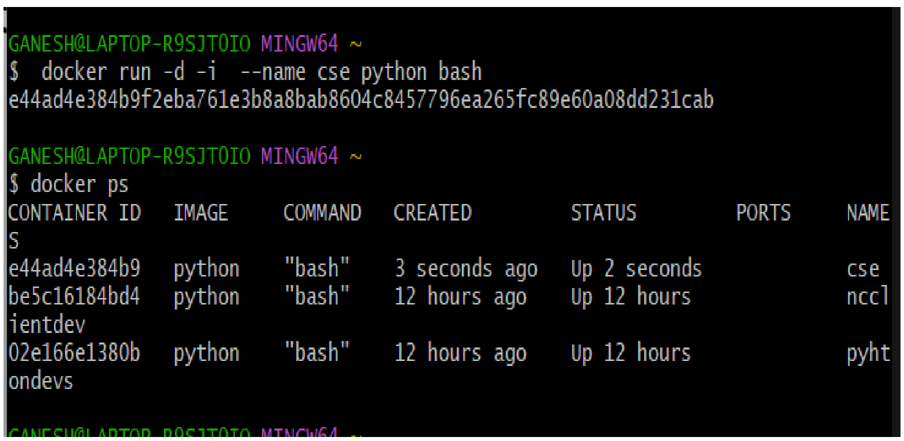


```
MINGW64:/c/Users/GANESH
GANESH@LAPTOP-R9SJTOIO MINGW64 ~
$   docker pull ciscotestautomation/pyats
Using default tag: latest
latest: Pulling from ciscotestautomation/pyats
c229119241af: Pulling fs layer
5a3ae98ea812: Pulling fs layer
46d5c684ee5f: Pulling fs layer
fdb2b484fc86: Pulling fs layer
bbf0e1f27c8a: Pulling fs layer
368c1125215c: Pulling fs layer
61056cf96c1d: Pulling fs layer
ec31c0a0e291: Pulling fs layer
47656ba7ec15: Pulling fs layer
51c0354707d9: Pulling fs layer
a3ed95caeb02: Pulling fs layer
fdb2b484fc86: Waiting
bbf0e1f27c8a: Waiting
51c0354707d9: Waiting
a3ed95caeb02: Waiting
368c1125215c: Waiting
47656ba7ec15: Waiting
61056cf96c1d: Waiting
ec31c0a0e291: Waiting
5a3ae98ea812: Download complete
```

The command docker pull ciscotestautomation/pyats downloads the latest version of this image from Docker Hub



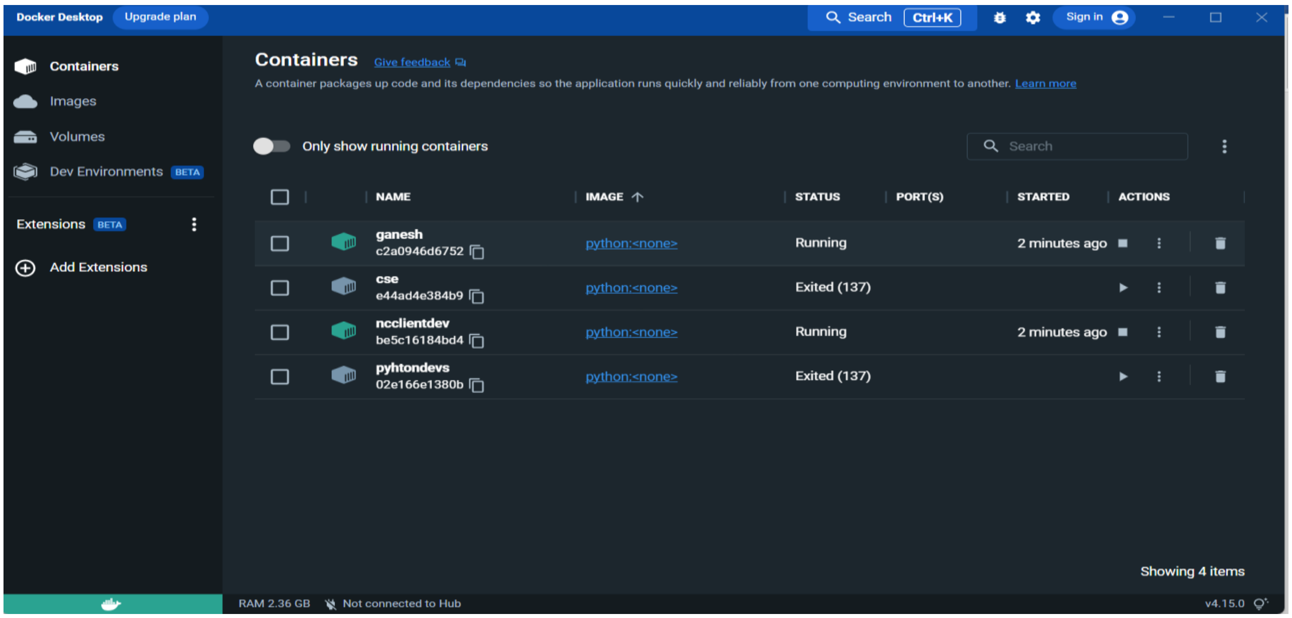
docker ps is a command used to list all running containers in Docker. By default, it only shows running containers. However, you can use different flags to get the list of other containers that are in stopped or exited status



docker run -d -i --name pyhtondevs python bash is a command used to create a new container named pyhtondevs from the python image with an interactive shell (bash) running inside it1. The -d option (shorthand for --detach) sets the container to run in the background, in detached mode, with a pseudo-TTY attached (-t). The -i option is set to keep STDIN attached (-i), which prevents the bash process from exiting immediately.

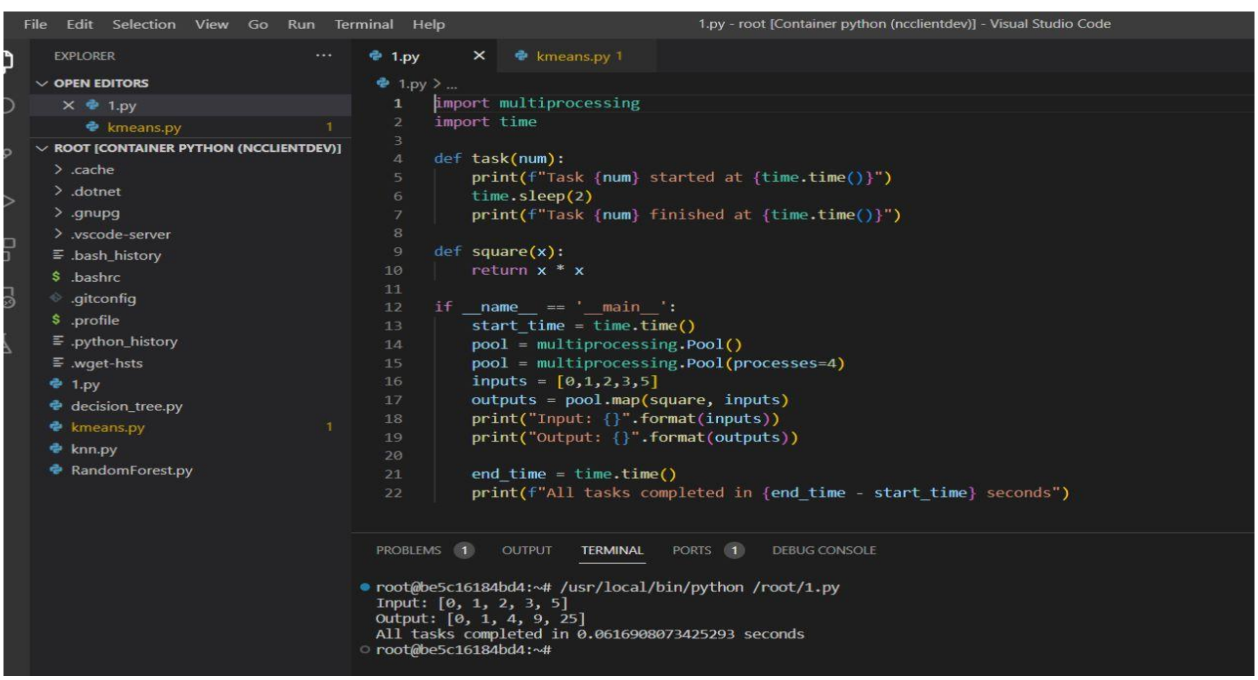
Output/Results

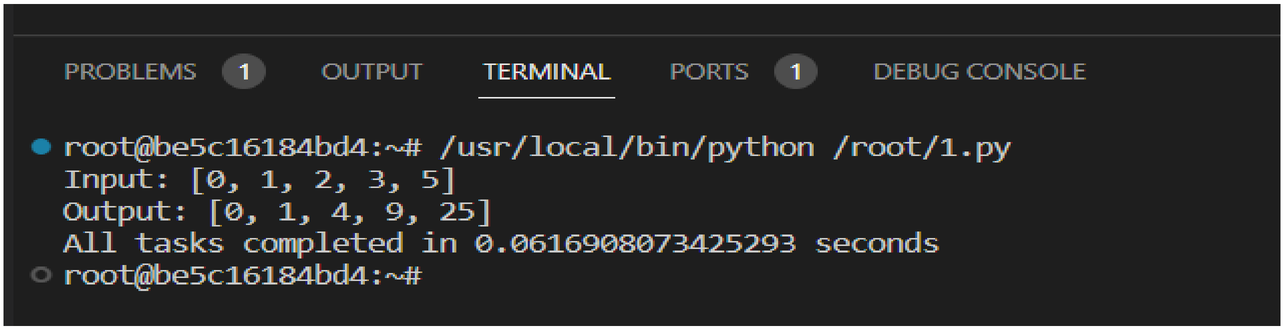
Created Virtual Environment in Docker:



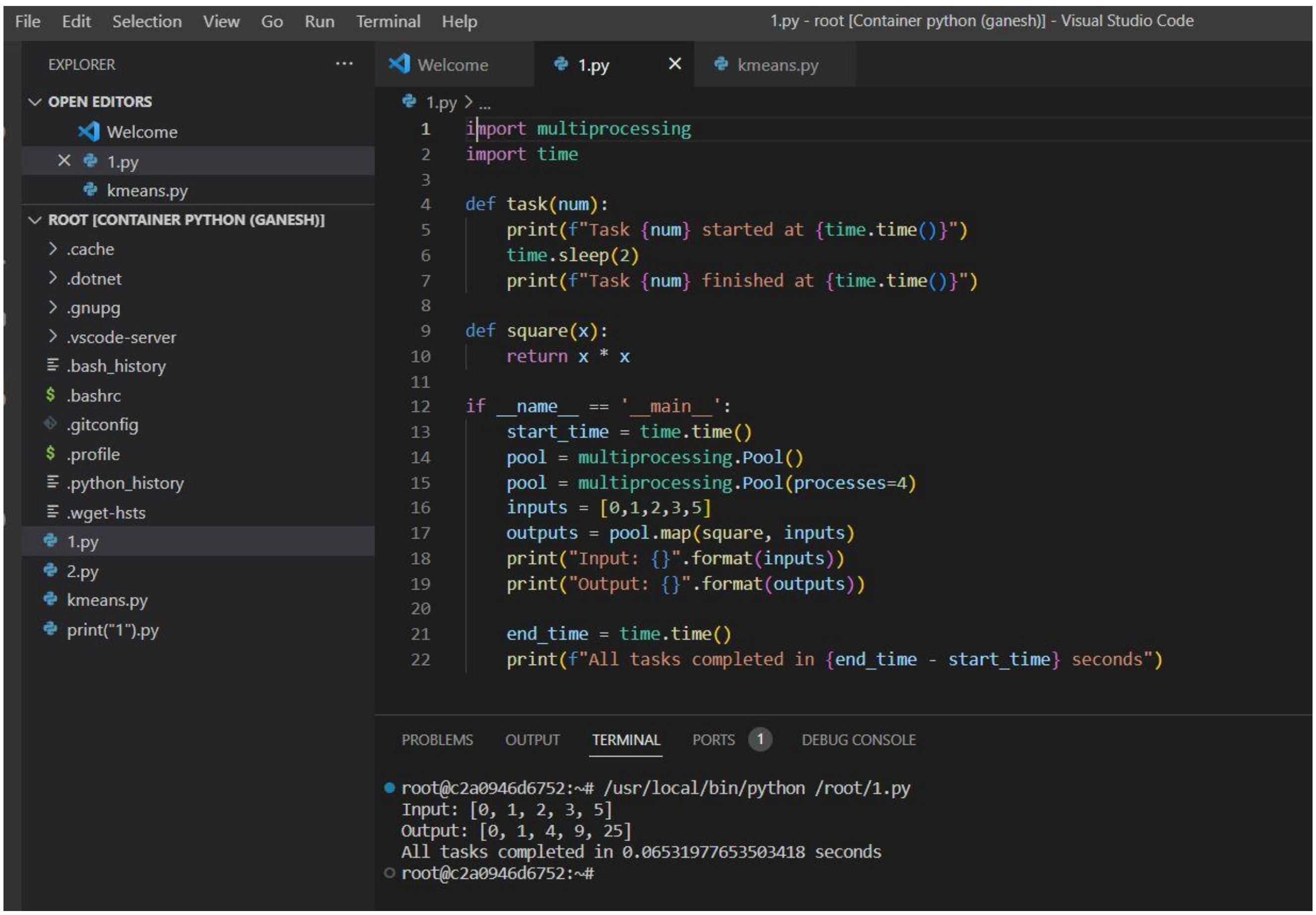
Parallelized Code for Square of an array of numbers:

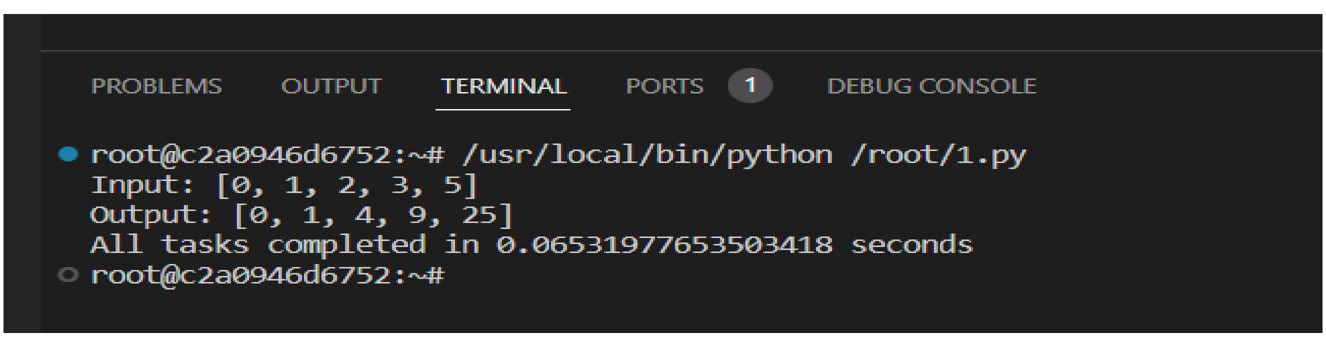
Container – 1(ncclient container) output:



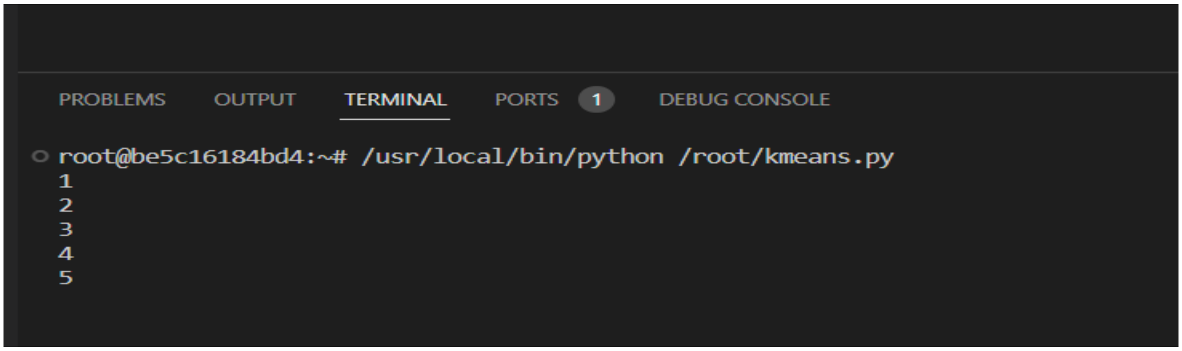


Container – 2(ganesh container) output:





Parallelized Code for k-means algorithm:



Conclusion

Creating a virtual environment with Docker and parallel processing can significantly enhance the efficiency and reliability of software development projects. Docker provides a lightweight and portable virtual environment that allows developers to easily package and distribute their software across different platforms and environments. Parallel processing enables developers to speed up their computations by utilizing multiple CPU cores or even multiple machines.

By combining these two technologies, developers can create a scalable and flexible software development environment that can handle complex workflows and large datasets with ease. Docker allows developers to easily manage dependencies and reproduce environments, while parallel processing enables them to process data faster and more efficiently.

Overall, using Docker and parallel processing can improve the speed, reliability, and reproducibility of software development projects, making it a valuable tool for any developer looking to optimize their workflow.

References

- https://stephen-odaibo.medium.com/docker-containers-python-virtual-environments-virtual-machines-d00aa9b8475
- https://repository.stcloudstate.edu/cgi/viewcontent.cgi?article=1091&context=msia_etds
- https://www.researchgate.net/publication/354834983_Replacing_Virtual_Machine_with_Docker_to_Build_or_Deploy_Application
- https://www.geeksforgeeks.org/virtualisation-with-docker-containers/
- https://pythonspeed.com/articles/activate-virtualenv-dockerfile/
- https://www.sitepoint.com/python-multiprocessing-parallel-programming/

Annexure

Parallelized Code for Square of an array of numbers:

```
import multiprocessing
import time

def task(num):
    print(f"Task {num} started at {time.time()}")
    time.sleep(2)
    print(f"Task {num} finished at {time.time()}")

def square(x):
    return x * x

if __name__ == '__main__':
    start_time = time.time()
    pool = multiprocessing.Pool()
    pool = multiprocessing.Pool(processes=4)
    inputs = [0,1,2,3,5]
    outputs = pool.map(square, inputs)
    print("Input: {}".format(inputs))
    print("Output: {}".format(outputs))

    end_time = time.time()
    print(f"All tasks completed in {end_time - start_time} seconds")
```

k-means parallel code:

```
# Import Python modules
from __future__ import division
import numpy as np
from multiprocessing import Pool
import timeit

import time

def task(num):
    print(f"Task {num} started at {time.time()}")
    time.sleep(2)
    print(f"Task {num} finished at {time.time()}")
start_time = time.time()

class K_Means_parallel(object):
    # Initialize input values n_clusters and max_iter
```

```
    def __init__(self, n_clusters, max_iter, num_cores):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.num_cores = num_cores

    # Function that assigns points to a cluster
    def assign_points_to_cluster(self, X):
        # Label points according to the minimum euclidean distance
        self.labels_ = [self._nearest(self.cluster_centers_, x) for x in X]
        # Map labels to data points
        indices=[]
        for j in range(self.n_clusters):
            cluster=[]
            for i, l in enumerate(self.labels_):
                if l==j: cluster.append(i)
            indices.append(cluster)
        X_by_cluster = [X[i] for i in indices]
        return X_by_cluster

    # Function that randomly selects initial centroids
    def initial_centroid(self, X):
        initial = np.random.permutation(X.shape[0])[:self.n_clusters]
        return  X[initial]

    # Function that updates centroids and repeats
    # assign_points_to_cluster until convergence or max_iter is reached
    def fit(self, X):
        self.cluster_centers_ = self.initial_centroid(X)
        for i in range(self.max_iter):
            # split data to self.num_cores chunks
            splitted_X=self._partition(X,self.num_cores)
            # Parallel Process for assigning points to clusters
            p=Pool()
            result=p.map(self.assign_points_to_cluster, splitted_X )
            p.close()
            # Merge results
            p.join()
            X_by_cluster=[]
            for c in range(0,self.n_clusters):
                r=[]
                for p in range(0,self.num_cores):
                    tmp=result[p][c].tolist()
                    r=sum([r, tmp ], [])
                X_by_cluster.append(np.array(r))
            # calculate the new centers
```

```
            new_centers=[c.sum(axis=0)/len(c) for c in X_by_cluster]
            new_centers = [np.array(arr) for arr in new_centers]
            old_centers=self.cluster_centers_
            old_centers = [np.array(arr) for arr in old_centers]
            # if the new centroids are the same as the old centroids then
            # the algorithm has converged
            if all([np.allclose(x, y) for x, y in zip(old_centers, new_centers)])
:
                self.number_of_iter=i
                break
            else :
                self.cluster_centers_ = new_centers
        self.number_of_iter=i
        return self

    # Function that randomly shuffles and partitions the dataset
    def _partition ( self,list_in, n):
        temp = np.random.permutation(list_in)
        result = [temp[i::n] for i in range(n)]
        return result

    # Function that calculates the minimum euclidean distance
    def _nearest(self, clusters, x):
        return np.argmin([self._distance(x, c) for c in clusters])

    # Function to calculate euclidean distance between two points
    def _distance(self, a, b):
        return np.sqrt(((a - b)**2).sum())

    # Function that returns predicted clusters for each point
    def predict(self, X):
        return self.labels_



sim2 = []
sim3 = []
sim4 = []


TEST_CODE2 = """
parakmeans = K_Means_parallel(n_clusters = 3, max_iter = 500, num_cores = 2)
parakmeans.fit(X)
"""
TEST_CODE3 = """
```

```
parakmeans = K_Means_parallel(n_clusters = 3, max_iter = 500, num_cores = 3)
parakmeans.fit(X)
"""
TEST_CODE4 = """
parakmeans = K_Means_parallel(n_clusters = 3, max_iter = 500, num_cores = 4)
parakmeans.fit(X)
"""


SETUP_CODE = """
import sklearn.datasets as skl
X, y = skl.make_blobs(n_samples=500, centers=3, cluster_std=0.60, random_state=0)
from __main__ import K_Means_parallel
"""
sim2.append(timeit.timeit(stmt=TEST_CODE2,setup=SETUP_CODE,number=50)/50)
sim3.append(timeit.timeit(stmt=TEST_CODE3,setup=SETUP_CODE,number=50)/50)
sim4.append(timeit.timeit(stmt=TEST_CODE4,setup=SETUP_CODE,number=50)/50)

print ("1")
SETUP_CODE = """
import sklearn.datasets as skl
X, y = skl.make_blobs(n_samples=1000, centers=3, cluster_std=0.60,
random_state=0)
from __main__ import K_Means_parallel
"""
sim2.append(timeit.timeit(stmt=TEST_CODE2,setup=SETUP_CODE,number=50)/50)
sim3.append(timeit.timeit(stmt=TEST_CODE3,setup=SETUP_CODE,number=50)/50)
sim4.append(timeit.timeit(stmt=TEST_CODE4,setup=SETUP_CODE,number=50)/50)

print ("2")

SETUP_CODE = """
import sklearn.datasets as skl
X, y = skl.make_blobs(n_samples=5000, centers=3, cluster_std=0.60,
random_state=0)
from __main__ import K_Means_parallel
"""

sim2.append(timeit.timeit(stmt=TEST_CODE2,setup=SETUP_CODE,number=50)/50)
sim3.append(timeit.timeit(stmt=TEST_CODE3,setup=SETUP_CODE,number=50)/50)
sim4.append(timeit.timeit(stmt=TEST_CODE4,setup=SETUP_CODE,number=50)/50)

print ("3")

SETUP_CODE = """
```

```
import sklearn.datasets as skl
X, y = skl.make_blobs(n_samples=10000, centers=3, cluster_std=0.60,
random_state=0)
from __main__ import K_Means_parallel
"""

sim2.append(timeit.timeit(stmt=TEST_CODE2,setup=SETUP_CODE,number=50)/50)
sim3.append(timeit.timeit(stmt=TEST_CODE3,setup=SETUP_CODE,number=50)/50)
sim4.append(timeit.timeit(stmt=TEST_CODE4,setup=SETUP_CODE,number=50)/50)
print ("4")

SETUP_CODE = """
import sklearn.datasets as skl
X, y = skl.make_blobs(n_samples=50000, centers=3, cluster_std=0.60,
random_state=0)
from __main__ import K_Means_parallel
"""
sim2.append(timeit.timeit(stmt=TEST_CODE2,setup=SETUP_CODE,number=50)/50)
sim3.append(timeit.timeit(stmt=TEST_CODE3,setup=SETUP_CODE,number=50)/50)
sim4.append(timeit.timeit(stmt=TEST_CODE4,setup=SETUP_CODE,number=50)/50)

print ("5")

SETUP_CODE = """
import sklearn.datasets as skl
X, y = skl.make_blobs(n_samples=100000, centers=3, cluster_std=0.60,
random_state=0)
from __main__ import K_Means_parallel
"""
sim2.append(timeit.timeit(stmt=TEST_CODE2,setup=SETUP_CODE,number=50)/50)
sim3.append(timeit.timeit(stmt=TEST_CODE3,setup=SETUP_CODE,number=50)/50)
sim4.append(timeit.timeit(stmt=TEST_CODE4,setup=SETUP_CODE,number=50)/50)

print ("6")

SETUP_CODE = """
import sklearn.datasets as skl
X, y = skl.make_blobs(n_samples=200000, centers=3, cluster_std=0.60,
random_state=0)
from __main__ import K_Means_parallel
"""

sim2.append(timeit.timeit(stmt=TEST_CODE2,setup=SETUP_CODE,number=50)/50)
sim3.append(timeit.timeit(stmt=TEST_CODE3,setup=SETUP_CODE,number=50)/50)
sim4.append(timeit.timeit(stmt=TEST_CODE4,setup=SETUP_CODE,number=50)/50)
```

```
print ("7")

SETUP_CODE = """
import sklearn.datasets as skl
X, y = skl.make_blobs(n_samples=300000, centers=3, cluster_std=0.60,
random_state=0)
from __main__ import K_Means_parallel
"""

sim2.append(timeit.timeit(stmt=TEST_CODE2,setup=SETUP_CODE,number=50)/50)
sim3.append(timeit.timeit(stmt=TEST_CODE3,setup=SETUP_CODE,number=50)/50)
sim4.append(timeit.timeit(stmt=TEST_CODE4,setup=SETUP_CODE,number=50)/50)

print ("8")

SETUP_CODE = """
import sklearn.datasets as skl
X, y = skl.make_blobs(n_samples=400000, centers=3, cluster_std=0.60,
random_state=0)
from __main__ import K_Means_parallel
"""

sim2.append(timeit.timeit(stmt=TEST_CODE2,setup=SETUP_CODE,number=50)/50)
sim3.append(timeit.timeit(stmt=TEST_CODE3,setup=SETUP_CODE,number=50)/50)
sim4.append(timeit.timeit(stmt=TEST_CODE4,setup=SETUP_CODE,number=50)/50)

print ("9")

SETUP_CODE = """
import sklearn.datasets as skl
X, y = skl.make_blobs(n_samples=500000, centers=3, cluster_std=0.60,
random_state=0)
from __main__ import K_Means_parallel
"""

sim2.append(timeit.timeit(stmt=TEST_CODE2,setup=SETUP_CODE,number=50)/50)
sim3.append(timeit.timeit(stmt=TEST_CODE3,setup=SETUP_CODE,number=50)/50)
sim4.append(timeit.timeit(stmt=TEST_CODE4,setup=SETUP_CODE,number=50)/50)

print ("10")


import pandas as pd
results2 = pd.DataFrame(sim2)
results2.to_csv('./sim_k3_p2.csv', sep='\t')
```

```
results3 = pd.DataFrame(sim3)
results3.to_csv('./sim_k3_p3.csv', sep='\t')
results4 = pd.DataFrame(sim4)
results4.to_csv('./sim_k3_p4.csv', sep='\t')
end_time = time.time()
print(f"All tasks completed in {end_time - start_time} seconds")
```