Design and Implementation of Docker Architecture for Parking Management Systems

***MSc. Eliana GJEDIKU*[1]**

***Ph.D.(c) Roland PLAKA*[2]**

Abstract

Docker Container is a virtualization method becoming an alternative to today's virtual machines. Containers isolate processes from each other and not the entire operating system. This study presents the usage of docker containers as an optional solution for

[1] MSc. Eliana Gjediku obtained her Bachelor's degree in Telecommunications Engineering at the Polytechnic University of Tirana and her Master of Science in Informatics Engineering at the European University of Tirana. During her master's, she also got into an Erasmus Exchange program for six months at Mälardalen's University in Sweden. She has over five years of experience in technical support and maintaining large cluster environments. Currently working as a system administrator for the National Academic Infrastructure for Supercomputing in Sweden (NAISS).

[2] Ph.D. Candidate Roland Plaka. Mr. Roland Plaka is a cybersecurity researcher at Linköping University, Sweden. He received his Bachelor's in Computer Engineering at the National Defence University of Turkey. He finished his Master's in Information System Security at the European University of Tirana. During his master's studies at the European University of Tirana, he got a double degree via Erasmus + program in software engineering at Mälardalen University, Sweden. Roland Plaka has over five years of experience in the IT / Cyber Security domain, spending most of his career in Cybersecurity Security Hands-on, Research, and Training. He has delivered several official training workshops at Epoka University and Linköping University. He maintains an exceptional track record of excellent feedback throughout his course delivery at academic / training entities or military institutions in PenTest, Digital Forensics, Risk Assessment, and Management. Roland has consulted defence, financial, and energy industries nationally and internationally. In a strive to continually raise cyber security awareness, Roland has dedicated hundreds of hours Ministry of Defence of Albania, BNP Paribas Cardif Nordics, and Linköping University providing this sector with his experience and knowledge in the field. Currently, he is conducting his Ph.D. studies in the cybersecurity of energy systems at Linköping University.

virtualization, which can replace virtual machines. As proof of concept, we design and implement a parking lot management system using containers. Our work shows that adapting docker containers instead of virtual machines makes the system use the resources more efficiently and access them shorter in time. An overview of the topic is presented by investigating and identifying the following features: challenges, issues, solutions, techniques, factors, and effects regarding the adoption of docker technology. Finally, this study highlights the benefits of containers and the possibilities of their application in particular processes of current systems.

Keywords: *docker container, virtual machine, parking system, optimization, design*

Introduction

The world of IT is constantly evolving, and there is no doubt that part of this development has happened thanks to virtualization. Virtualization is one of the most preferred techniques that has dramatically benefited businesses. Virtualization creates a virtual version of something instead of having it in a physical environment. For example, to create a virtual network, application, or server version. This makes it possible to split a physical server into several virtual servers sharing physical resources. Each server can then run its operating system. This technology has become so popular due to hardware consolidation, as virtualization reduces the need for physical hardware by increasing its availability and simplifying administration. As the use of the Internet has increased, so has the need for servers and data centers because companies want their websites to be available to their customers to expand their business.

The bigger the company, the more equipment is needed, meaning large companies can have hundreds of servers. This increases the need for fencing, which leads to an increase in the cost of maintenance, energy, and time; an exceptional staff is needed to perform these tasks. Virtualization is an alternative technique that solves scalability problems and reduces costs simultaneously. With virtualization, some physical machines can be replaced with one that runs these machines virtually. Virtualization brings significant benefits to businesses because it significantly reduces IT costs and enables time savings by increasing the efficiency and flexibility of resources.

These advantages have brought great interest in the market, where businesses want to stay constantly updated on the most efficient and cost-effective virtualization solutions. Virtualization has been around for several years and is commonly known as hypervisor-based virtualization. Virtualization is a technology that enables the creation of several virtual machines on a single physical machine. Virtualization reduces the number of servers, reduces the size of data centres, and reduces hardware costs and the need for maintenance. A virtual machine is a software that emulates a physical computer system and its functionalities using the resources such as processor, memory, and bandwidth of the physical machine on which it is installed. The physical host is called "host," and the virtual machine built on the "host" is called "guest." Hypervisors enable resources for virtual machines and make it possible to allocate resources so that virtualization works appropriately. A general approach to virtualization has been the use of virtual machines. This was the first method on the market that enabled virtualization.

Hypervisor-based virtualization uses the hypervisor to segment a system into several systems. This segmentation is achieved by creating several virtual machines that can run on the "host." There are two types of hypervisor-based virtualization, type 1 and type 2. Type 1, a "bare-metal hypervisor," runs directly on the host's hardware (see Figure 1). The host operating system and the hypervisor are combined into a single hypervisor layer. This creates a direct hardware connection where the hypervisor can communicate directly with the host's hardware components. A system can leverage resources such as performance and scalability through this integration. Type 2, called "hosted hypervisor," separates the host operating system and the hypervisor into different layers (see Figure 2). All virtual machines will run through the host's operating system and not directly on the hardware, as in type 1. In type 2, an operating system is built into the host's hardware, and its top layer, a hypervisor, is installed and behaves like software used for virtualization. It is the most widely used in virtualization; however, it is no longer the only method that offers virtualization options.

Container-based virtualization is another type that has created a lot of curiosity and interest in the IT market. Container-based virtualization provides operating system virtualization and does not use a hypervisor. This is the most significant difference between containers and virtual machines; they are visualized on the operating system, not the hardware. Each container created on the same physical host can share the same host operating system with another. This makes it possible to run several applications simultaneously without using several operating systems, unlike the traditional virtual machine method. Containers package software, making running several containers on the same operating system possible. These containers can run concurrently and separately on the same operating system. It does not require access to the entire operating system to run the software, as they have its resources and libraries. Different applications, servers, and databases can run on the same physical machine by placing them in isolated containers.

FIGURE 1. Type 1 Virtualization

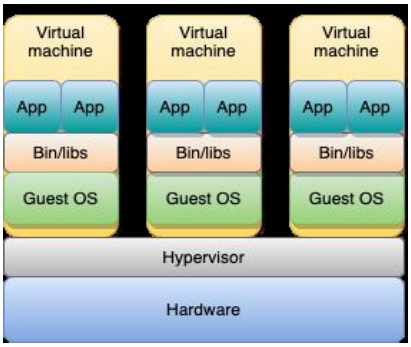


FIGURE 2. Type 2 Virtualization

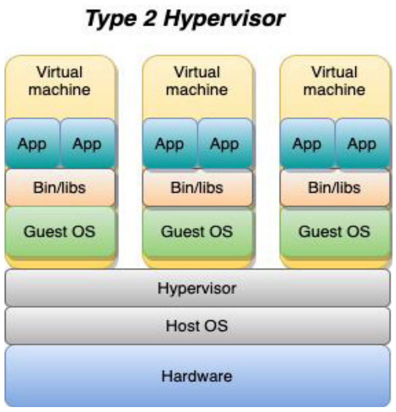
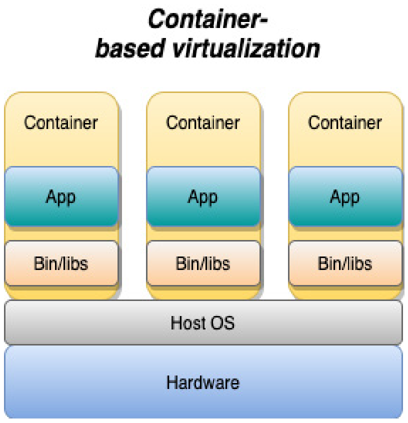


FIGURE 3. Container-based virtualization



This study focuses on designing and implementing a parking management system in a docker container and identifying the challenges and opportunities this deployment will create. Our first goal is distinguishing the differences between containers and virtual machines regarding performance, security, and adaptability. Moreover, we aim to re-create the functionalities of a system in a docker container.
The contributions of this study are as follows:

1. We propose a parking management system architecture developed on docker container technology.
2. We analyze the model parking management system regarding resource management, efficiency, and portability.

Our work reveals a better understanding of the docker technology applicability in real-world systems. The remainder of this paper is organized as follows. Section 2 describes the background and definitions to understand the architecture. Section 3 describes the proposed docker container-based parking management architecture and all its components. In section 4, we discuss the implementation of the use case. Section 5 presents the results. In section 6, we conclude our work.

Background

Docker container

A container is a sandboxed process on the computer that is isolated from all other processes on the host machine. Docker makes these capabilities easy to use. The container becomes the unit for distributing and testing the application. Briefly, the container is an instance of an image that can be created, started, stopped, moved, or deleted using DockerAPI or CLI, which can run on local machines or virtual machines or be deployed in the cloud. Also, it is portable and can be run on any operating system. It is isolated from other containers and has its software, binaries, and configurations.

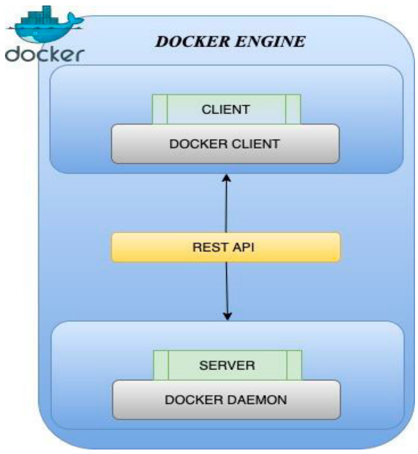
Docker Engine

Unlike traditional virtual machines, docker has no hypervisor. It has a "docker-engine" layer that takes care of everything running in these containers. The Docker engine creates and manages containers. It has two versions: Docker Engine Community and Docker Engine Enterprise. Docker-Engine Community is an open-source code and is complimentary, while Enterprise costs extra because it offers more advanced services. We used Docker Engine Community. The Docker engine is the client-server application whose architecture is a docker client that communicates with a docker daemon. The Docker daemon creates, runs, and deploys containers, and it does so by listening to incoming REST API requests. The REST API enables a communication path for docker objects such as images, plugins, networks, and containers to communicate with the docker daemon. A daemon can communicate with other daemons when necessary. The docker client is the primary channel for users to integrate with docker. When a docker client executes a command, that command is sent to the docker daemon via the REST API to be executed, see Figure 4.

Docker Build

Docker Build is one of the most used features of Docker Engine. Whenever we build an image, Docker Build is used. Docker Build is a crucial part of the software development lifecycle, allowing us to package and ship code anywhere. Docker Engine uses a client-server architecture and consists of multiple components and tools. The most common method of executing a build is by issuing a docker build command. The CLI sends the request to the Docker Engine, which, in turn, executes the build. Two components in the Engine can be used to build an image. Starting with the 18.09 release, Engine ships with Moby BuildKit, the new component for running builds. BuildKit is the evolution of Legacy Builder, which comes with new and greatly improved functionality that can be a powerful tool to improve the performance of building or reusing Dockerfiles and provide support for complex scenarios. Docker Buildx is a CLI plugin that extends the docker command with full feature support provided by the BuildKit Builder Toolkit. Docker Buildx provides the same user experience as docker builds with many new features such as instance creation, building against multiple nodes simultaneously, output configuration, internal build caching, and target platform specification. In addition, Buildx also supports new features yet to be available for regular docker builds, such as building manifest lists, distributed in-memory storage, and exporting build results to OCI images. Docker Build is much more than a simple build command. It is not only about packaging code but is an entire ecosystem of tools and features that support everyday workflow tasks and provide more complex and advanced scripting support.

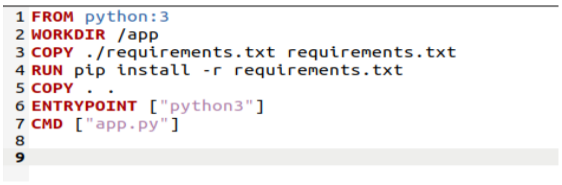
FIGURE 4. Communication between docker daemon and docker client



Docker file

Docker uses a docker file to build the images that build containers. The docker file is a text file that contains commands with instructions on how to merge containers. Docker reads instructions from top to bottom, see Figure 5, where docker starts by reading the command FROM ubuntu:20.04. Docker builds images by reading instructions from a Dockerfile. This text file contains instructions that adhere to a specific format needed to compile the application into a container image. Docker files are an important development for building images and can facilitate the automated building of multi-layered images based on your unique configurations. Dockerfiles can start simple and grow with needs and support images that require complex instruction.

FIGURE 5. Docker file example



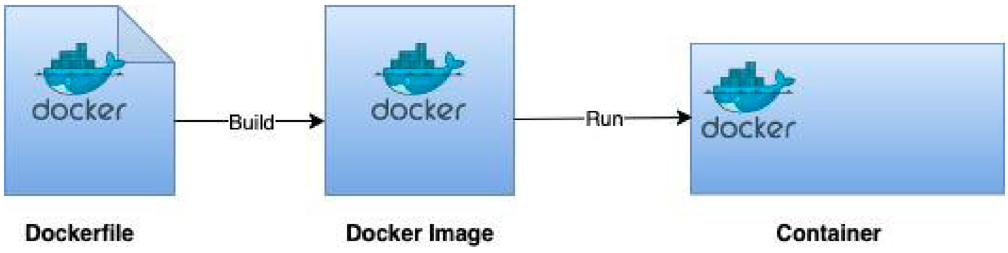
```
FROM python:3
WORKDIR /app
COPY ./requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
ENTRYPOINT ["python3"]
CMD ["app.py"]


```

Docker Compose

Docker compose an orchestration tool used by Docker to run multiple containers simultaneously. The advantage of docker-compose is that it makes it possible to split all services across their containers instead of having them in a single container.

The logic of this technology lies in creating a multi-container solution, which makes containers more manageable and easier to build and maintain. To realize this, it is necessary to maintain a particular service in the respective container and not by interacting in a standard container. A YAML file is used to build these multi-containers, specifying the configurations required to implement the services needed in the container. Services specified in a YAML file are based on a docker file. The docker architecture is given in Figure 6.

FIGURE 6. Docker container build process



Docker Image

When a container runs, it uses an isolated file system. An image container powers this file system. Since the image contains the container's filesystem, it should contain everything needed to run an application—all configurations, scripts, binaries, etc. The image also contains other configurations for the container, such as environment variables, a default command to run, and other metadata.

Docker Application Architecture for Parking Management System

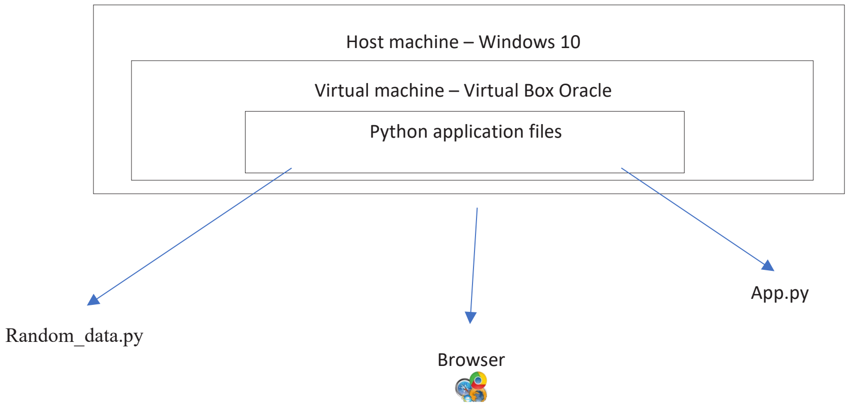
System architecture

To test the hosting of a web app in a virtual machine and containerized in a Docker, we took as an example the construction of an application that displays free parking spaces in the points in Tirana. First, we study the types of parking systems and how we can apply them. Recently, parking systems are changing in significant ways, driven primarily by the adoption of new technologies. New apps and digital services are making it easier for people to find and pay for parking, while sensor-connected car features are helping cities monitor where spaces are most needed. Emerging innovations such as artificial intelligence, virtual reality, and blockchain are transforming the industry.

Parking is a significant problem for cities. Many drivers circle for almost an hour, hoping to find a place. In contrast, other drivers abandon their cars on the side of the road, blocking emergency vehicles and even other pedestrian and bicycle lanes. Parking apps make it easier than ever to find and pay for parking. Drivers use their phones to find available parking spots near their destination, reserve the spot, and then pay automatically with a mobile wallet like Apple Pay or Google Pay. Apps also organize carpooling, making parking more manageable and efficient for everyone.

We developed a parking management system based on docker containers that can be run locally in an enterprise data center or any cloud environment shown in Figure 7. Our host machine runs on Windows 10 operating system (OS). Windows 10 OS supports virtual machine applications such as Oracle Virtual Box. We selected an Ubuntu 22.04 .iso image to implement the parking management system application. Moreover, we developed two Python scripts. The first script, named random_data.py, automatically generates random data and shows them in the corresponding tables of parking points. The second script, named app.py, connects to the database and, using specific buttons for the different parking points, updates the data for the user who will access them. We run the scripts on two different docker containers, and the data for the regular users is accessible through a public website.

FIGURE 7. System architecture model


Host machine – Windows 10
Virtual machine – Virtual Box Oracle
Python application files
Random_data.py
Browser
App.py

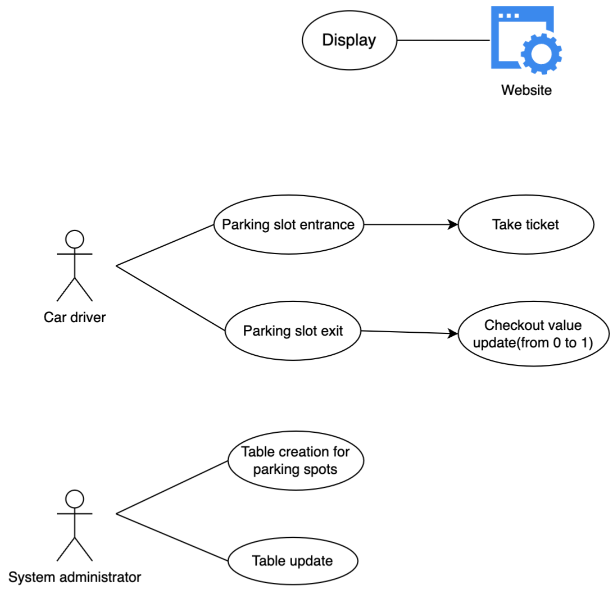
System components

This study deals with a model of a parking system that contains seven different points in the city of Tirana. Below is the corresponding UML diagram. The parking system has three actors, the primary named system administrator, the visual presentation(website), and the driver. The visual display actor visually displays free parking spaces for web users. It counts the number of cars entering the parking system.

The car/driver generates data for the visual presentation when entering the parking point and gets a ticket which takes the value 0. Then when the car leaves the checkout parking point, it will take the value 1.

The administrator actor has full access to the database, which is built in MySQL. The system administrator deals with architecture configurations, building the database, building tables for seven parking spots of the parking system, and updating the number of free spaces with current values.

FIGURE 8. UML diagram of system actors



System configuration

The machine we are working on has a Windows 10 operating system with 8GB of Ram and four processors. On this machine, we install the required virtual machine. The virtual machine we use is Virtual Box Oracle. VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for Enterprise and home use. VirtualBox is a highly feature-rich, high-performance product for enterprise customers and the only professional solution freely available as Open-Source Software under the GNU General Public License (GPL) version 3 terms. Currently, VirtualBox runs on Windows, Linux, Macintosh, and Solaris hosts and supports many guests operating systems.

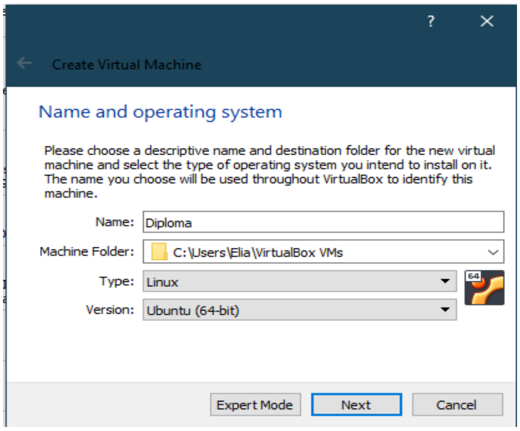
Advantages of Ubuntu:

1. Open Source. This operating system is free to download, use and distribute. This makes it even more necessary and preferred by users.
2. Security. Ubuntu is one of the most secure operating systems with a built-in firewall and virus protection software.
3. Accessibility. Being among the few systems with translations of up to 50 languages makes it even more accessible to different individuals.

Installing a virtual machine

- First, we download an image of the operating system we will use and VirtualBox, which serves to virtualize the environment where we want to do the testing.
- After both applications are downloaded, we start with installing the virtual machine like most other applications.
- After installing VirtualBox as an application on our physical machine, we move on to creating the virtual environment where we choose the name; we will give the machine and the operating.
- After that, we will choose the parameters that the virtual machine will have, including the amount of memory, CPU, and storage as below, considering the size and complexity of the application we want to develop in this machine.
  - 4Gb Memory
  - 2 CPUs
  - 50 GB Storage system will have along with the selected version.
  - After we have given the specifications and gone through all the steps, we can execute the run command on the machine we just built.

FIGURE 9. Operating System selection



Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: Diploma

Machine Folder: C:\Users\Elia\VirtualBox VMs

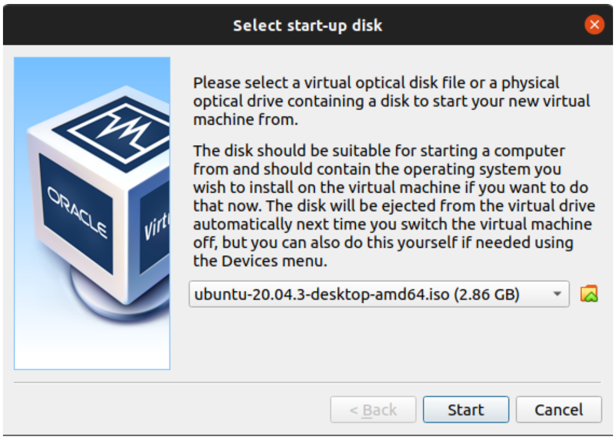
Type: Linux

Version: Ubuntu (64-bit)

Installing Ubuntu OS on Virtual Box

- When we turn on the machine, a startup disc will be created (which is practically the image of the Ubuntu operating system that we downloaded).
- Select the system we want to use and click start.

FIGURE 10. Installing Ubuntu



Select start-up disk

Please select a virtual optical disk file or a physical optical drive containing a disk to start your new virtual machine from.

The disk should be suitable for starting a computer from and should contain the operating system you wish to install on the virtual machine if you want to do that now. The disk will be ejected from the virtual drive automatically next time you switch the virtual machine off, but you can also do this yourself if needed using the Devices menu.

ubuntu-20.04.3-desktop-amd64.iso (2.86 GB)

- After that, we follow the steps of installing the system, and after we finish with this part, we give the last details to work as simply as possible.
- Guest Additions is another software that unlocks some more advanced features of VirtualBox. This includes a better integration between the virtual machine and the host machine, making it possible to use the copy-paste option from one machine to another, adjust the image quality, and share folders.

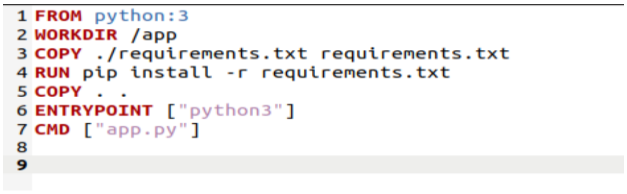
Docker installation

We must have a file with the program's requirements to publish a project in Python and install them on our machine. This file contains the packages and dependencies and their respective versions to run the project. Run the command below in root.

- pip freeze > requirements.txt

One good thing about containerization with Docker is that I can package the application with all the runtime dependencies required to make it self-contained. Therefore, the application works without worrying about incompatibilities with the environment where it will be hosted.

FIGURE 11. Docker file creation



```
FROM python:3
WORKDIR /app
COPY ./requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
ENTRYPOINT ["python3"]
CMD ["app.py"]
```

To install Docker, we first need a Dockerfile. FROM python:3.6-buster

WORKDIR /app

COPY requirements.txt.

RUN pip install -r requirements.txt

COPY. .

CMD [“python3”, “app.py”]

FROM python:3.6-buster - Since Docker allows us to use existing images, we install a Python image and install it on top of our Docker image.

WORKDIR /app – defining this folder as the directory where we will work.

COPY requirements.txt. ; COPY. . – copy every file in this directory

CMD [“python3”, “we_app.py”] – defines which application will be executed

• Building the Docker image

docker image build -t flask_docker .

• Starting the container

After we finish building the image, we try to launch it

docker run -p 5000:5000 -d flask_docker

Building the Flask application

Flask is a free and open-source Python framework designed to help developers build secure, scalable, and maintainable web applications. Flask is based on wrkzeug and uses Jinja2 as a template engine.

Flask is built with extensions and Python packages that add functionality to a Flask application.

There are different methods to install Flask on Ubuntu.

Flask packages are included in the Ubuntu repository and can be installed using the apt package manager. This is the easiest way to install Flask on Ubuntu 20.04, but not as flexible as installing it in a virtual environment.

Virtual environments allow us to create an isolated environment for different Python projects. This way, we can have many different Flask environments on a single machine and install a specific module version on a per-project basis without worrying about it affecting other Flask installations.

First, we create a directory that will contain the application.

• mkdir flask_app

The next step is to install Flask in this directory.

• cd flask_docker

• pip install flask

After success What is MySQL?

MySQL is a relational database management system (RDBMS) developed by Oracle that is based on Structured Query Language (SQL).

A database is a structured collection of data. It can be anything from a simple shopping list to a photo gallery or a place to hold large amounts of information on a corporate network. In particular, a relational database is a digital store that collects data and organizes it according to the relational model. In this model, tables consist of rows and columns, and the data elements' relationships follow a strict logical structure. An RDBMS is a set of software tools to implement, manage, and query such a database.

MySQL is integral to many of the most popular software suites for building and maintaining everything from customer-facing web applications to robust data-driven B2B services. Its open-source nature, stability, and rich feature set, coupled with continued development and support from Oracle, have meant that critical web organizations like Facebook, Flickr, Twitter, Wikipedia, and YouTube all use MySQL support.

Benefits of MySQL

Ease of use: Since it supports the SQL language, users do not need to be technical experts to access the database. It can be easily accessed by users with basic SQL knowledge and experience with other relational databases fully installing Flask; the next step is to install MySQL.

Free of Cost: Another benefit of using this database is that the user does not have to spend money to pay the license fee, as it is free of cost and available on the official website for download.

Customizable code: Since it is an open-source tool, software developers can customize and use the source code according to their applications. The source code is freely available to web users.

Security: It offers one of the most secure databases in the world and is therefore used by popular web applications such as Facebook, Twitter, Instagram, etc. Its various security features, such as Firewall, Encryption, and User Authentication, are the basis for protecting sensitive user information from intruders.

Better performance: Supports the multi-engine storage feature, which facilitates database administrators to configure the database to balance the workload. Hence, it makes the database perfect in terms of performance.

High Availability: It provides 24*7 hours availability and offers solutions like Master/Slave Replication and specialized Cluster servers.

Platform Friendly: It is a platform-friendly database that supports several platforms like Microsoft Windows, Oracle Solaris, AIX, Symbian, Linux, MAC OS, etc.

User-friendly interface: It has a user-friendly interface with many self-management features and various automated processes like configuration and administration-related tasks, which allows users to get work done effectively right from the start.

Installing MySQL on Ubuntu 20.0.4 LTS

First, we need to update the packages that the system itself has:

sudo apt update

Then we install the mysql server package:

sudo apt install mysql-server

Make sure the server is running using the systemctl start command:

sudo systemctl start mysql.service

The default data for this installation will be:

User: root

Password: ‘root ‘

First, we created a database called PARKING

CREATE PARKING DATABASE;

USE PARKING;

To store the data of each parking point, we build a table for each point with ticket and checkout columns.

CREATE TABLE liqeni_artificial (bileta int, checkout int);

CREATE TABLE asllan_rusi (bileta int, checkout int);

CREATE TABLE 7_xhuxhat (bileta int, checkout int);

CREATE TABLE bulevardi_i_ri (bileta int, checkout int);

CREATE TABLE sheshi_italia (bileta int, checkout int);

CREATE TABLE sheshi_skenderbej (bileta int, checkout int);

CREATE TABLE qyteti_studenti (bileta int, checkout int);

CREATE TABLE stadiumi_dinamo (bileta int, checkout int);

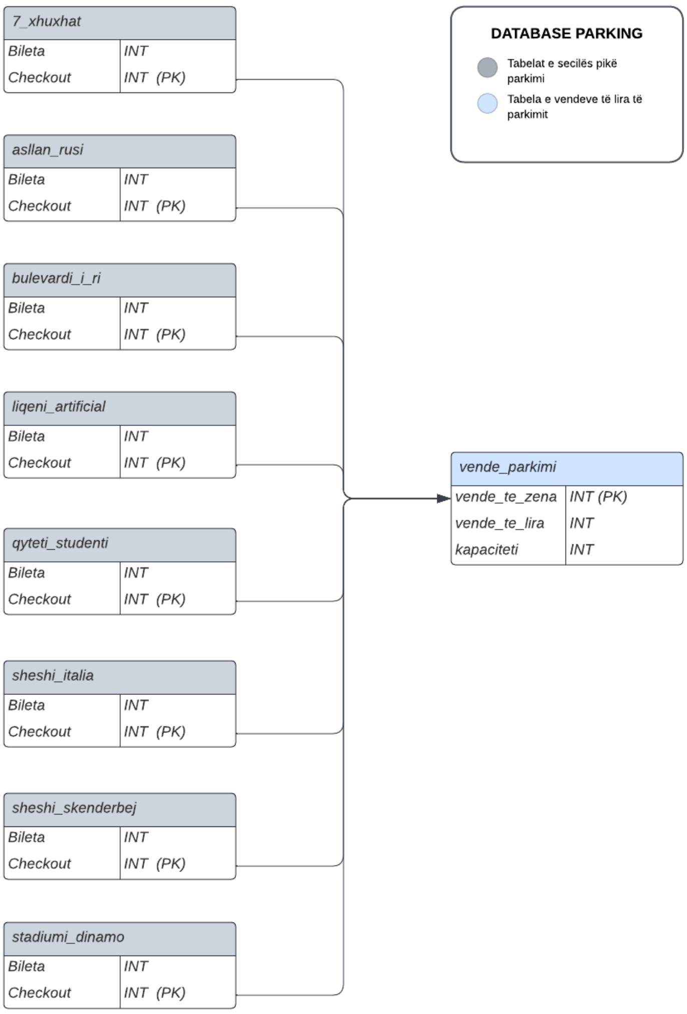
The ticket received a random number from 1-2000. Checkout I received a value of 1 when the ticket was scanned to exit and a value of 0 when the car was still parked (randomly generated values). I got the parking points online from the site http://tiranaparking.al/.

CREATE TABLE vende_parkimi (vende_tw_lira int, vende_tw_zwna int, kapaciteti int);

The last table created is parking_places. This table will have columns named vacant_seats, occupied_seats, and capacity. The capacity will have a value of 2000 and indicate the maximum number of parking spaces. The other two fields will be automatically updated via the web button functions. Occupied_seats will be the number of generated tickets with checkout value = 0, while free_seats will be calculated as capacity – occupied_seats.

The capacity is set to a fixed figure of 2000; the occupied places will be automatically updated according to the parking points and the number of cars with the checkout value 0; the free places will be the difference between the capacity and the occupied places.

FIGURE 12. Database connection



Results

Docker provides the ability to package and run an application in an isolated environment called a container. Isolation and security allow us to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so we do not need to rely on what is currently installed on the host. We can easily share containers as we work and ensure everyone we share them with has the same container that works the same way. Docker provides tools and a platform to manage the lifecycle of containers. Dockers make a fast and stable distribution of applications. Docker streamlines the development lifecycle by allowing developers to work in standardized environments using local containers that deliver applications and services. Containers are great for continuous integration and continuous delivery workflows. Developers use Docker to push their applications to a test environment and run automated and manual tests. When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation. When testing is complete, getting customer acceptance is as simple as pushing the updated image to the production environment. Docker allows for portable workloads. Docker containers can run on laptops, physical or virtual machines in a data center, cloud providers, or other mixed environments. Dockers make it easier to dynamically manage workloads and scale up or down applications and services as business needs dictate in near real-time. Docker is easy and fast. It provides a stable and cost-effective alternative to hypervisor-based virtual machines, so we can use more of our server capacity to achieve business goals. Docker is perfect for high-density environments and small and medium deployments with limited resources.

Conclusion

Parking is expected to grow as a new industry integrated with technology. Parking management systems need integration and analytical solutions that combine data from distributed parking slots, which use local storage. However, cities need to grow more; thus, more resource-efficient solutions must be implemented. Our study shows that virtualization is an innovative solution that provides scalability, portability, and security. We provide a solution that can easily be implemented in Tirana city, needing more parking points and management. Our docker container-based web application impressively performs as better as the traditional solutions in terms of hosting services and performs better in terms of process management. As a future work, we aim to provide a central solution that can be implemented nationally to offer parking solutions in every major city of Albania.

References

R. Morabito, J. Kjällman and M. Komu,. (2015). *"Hypervisors vs. Lightweight Virtualization: A Performance Comparison," pp. 386-393.* IEEE International Conference on Cloud Engineering.

N. Jain & S. Choudhary. (2016). *"Overview of virtualization in cloud computing," pp. 1–4.* Symposium on Colossal Data Analysis and Networking (CDAN).

"Overview of Docker Compose." (2021). Retrieved from https://docs.docker.com/compose/

"Dockerfile reference." (2021). Retrieved from https://docs.docker.com/engine/reference/builder/

Desai, P. R. (2014). *"A survey of performance comparison betWeen virtual machines and containers" pp.55–59.* International Journal of Computer Science and Engineering.

Scheepers, M. J. (2014). *"Virtualization and containerization of application infrastructure: A comparison" pp. 1–7.* in 21st Twente Student Conference on IT.

Bernstein, D. (2014). *"Containers and Cloud: From LXC to Docker to Kubernetes," vol. 1, no. 3, pp. 81-84.* IEEE Cloud Computing.

W. Felter, A. Ferreira, R. Rajamony and J. Rubio. (2015). *"An updated performance comparison of virtual machines and Linux containers" pp. 171-172.* IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS).

Joy, A. M. (2015). *"Performance comparison betWeen Linux containers and virtual machines," pp. 342-346.* International Conference on Advances in Computer Engineering and Applications.

S. Soltesz, H. Potzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. (2007). *"Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," vol 41, pp. 275–287.* SIGOPS Operating Systems Review.

Naik, N. (2016). *"Migrating from Virtualization to Dockerization in the Cloud: Simulation and Evaluation of Distributed Systems" pp. 1–8.* IEEE 10th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments (MESOCA).

P. Sharma, L. Chaufournier, P. Shenoy, and Y. Tay. (2016). *"Containers and virtual machines at scale: A comparative study" pp. 1-13.* Proceedings of the 17th International Middleware Conference.

O. Rudyy, M. Garcia-Gasulla, F. Mantovani, A. Santiago, R. Sirvent and M. Vázquez. (2019). *"Containers in HPC: A Scalability and Portability Study in Production Biological Simulations" pp. 567-577.* IEEE International Parallel and Distributed Processing Symposium (IPDPS).

"What is a container." (2021). Retrieved from https://azure.microsoft.com/en-us/overvieW/What-is-a-container/