**BASAVARAJESWARI GROUP OF INSTITUTIONS**

**Ballari Institute of Technology & Management**

**AUTONOMOUS INSTITUTE UNDER VISVESVARAYA TECHNOLOGICAL UNIVERSITYJNANA SANGAMA, BELAGAVI 590018**

**INTERNSHIP**

**Report On**

# Simple Container Deployment Manager Project

Submitted in partial fulfillment of the requirements for the award of degree of

**Bachelor of Engineering**

**In**

**COMPUTER SCIENCE (DATA SCIENCE) AND ENGINEERING**

## Submitted by

**ULURU PHANI KUMAR**

**3BR23CD101**

## Internship Carried Out By

**EZ TRAININGS & TECHNOLOGIES PVT.LTD HYDERABAD**

**Internal Guide External Guide**

**MRS,PARVATHI VISHAL**

**Asst.prof,CSE DS Technical Trainer**

MRS,**ANUSHYA**

**Asst. prof,CSE DS**

### BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

NACC Accredited Institution\*

**(Recognized by Govt. of Karnataka, approved by AICTE, New Delhi & Affiliated to Visvesvaraya Technological University, Belagavi)**

**"Jnana Gangotri" Campus, No.873/2, Ballari-Hospet Road, Allipur, Ballar1-583 104 (Karnataka) (India) Ph: 08392 – 237100 / 237190, Fax: 08392 – 237197**

**2023-2024**

**BASAVARAJESWARI GROUP OF INSTITUTIONS**

## BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

**Autonomous institute under VISVESVARAYA TECHNOLOGICAL UNIVERSITYJNANA SANGAMA,**

**BELAGAVI 590018**

NACC Accredited Institution\*

**(Recognized by Govt. of Karnataka, approved by AICTE, New Delhi & Affiliated to**

**Visvesvaraya Technological University, Belagavi)**

### "JnanaGangotri" Campus, No.873/2, Ballari-Hospet Road ,Allipur,

**Ballar1-583 104 (Karnataka)(India)**

**Ph: 08392 – 237100 / 237190, Fax: 08392 –237197**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# CERTIFICATE

This is to certify that the Internship entitled **“ Simple Container Deployment Manager Project”** has been successfully completed by **ULURU PHANI KUMAR** bearing USN **3BR23CD101** a bonafide student of Ballari Institute of Technology and Management,

Ballari. For the partial fulfillment of the requirements for the **Bachelor’s Degree in Computer**

**Science (DATA SCIENCE) and Engineering** of the VISVESVARAYA TECHNOLOGICAL UNIVERSITY,

Belagavi during the academic year 2023-2024.

**Signature of Internship Signature of HOD**

**Co-ordinator**

**MRS,PARVATHI DR ARADHANA D**

**Asst. prof, CSE -DS Prof. and HOD(CSE)**

MRS,**ANUSHYA**

**Asst. prof, CSE-DS**

**DECLARATION**

I, **ULURU PHANI KUMAR,** second year student of Computer Science (DATA

SCIENCE) and Engineering, Ballari Institute of Technology, Ballari, declare that Internship entitled **EVENT MANAGEMENT PLATFORM** is a part of Internship Training successfully carried out by **EZ TECHNOLOGIES & TRAININGS PVT.LTD ,Hyderabad** at “**BITM,BALLARI”.** This report is submitted in partial fulfillment of the requirements for the award of the degree, Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belagavi.

**Date : Signature of the Student**

**Place :**

**ACKNOWLEDGEMENT**

The satisfactions that a company the successful completion of my internship on “Simple Container Deployment Manager Project ” would be incomplete without the mention of people who made it possible, whose noble gesture, affection, guidance, encouragement and support crowned my efforts with success. It is my privilege to express my gratitude and respect to all those who inspired me in the completion of my internship.

I am grateful to our respective coordinator **“MRS,PARVATHI (Asst.prof,CSE-DS) , MRS,ANUSHYA (Asst.prof,CSE-DS)”** for his noble gesture, support co-ordination and valuable suggestions given to me in the completion of Internship.

I also thank DR,**Aradhana D,** H.O.D. Department of **Computer science Data Science and engineering** for extending all his valuable support and encouragement.

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Chapter No.** | **Chapter Name** | **Page No.** |
| **1** | **Company Profile** | **01** |
| **2** | **Day to day activity(student diary extract)** | **02** |
| **3** | **Abstract** | **03** |
| **4** | **Introduction** | **06** |
| **5** | Project Overview, Objectives ,Key concepts | **07** |
| **6** | **Implementation details and Real-Life Application and Benefits** | **16** |
| **7** | Purposes of a Simple Container Deployment Manager | **18** |
| **8** | **Conclusion** | **20** |
| **9** | **References** | **21** |

**Report on Simple Container Deployment Manager Project**

INTRODUCTION

In today's fast-paced software development landscape, the need for efficient deployment, management, and scaling of applications has become increasingly important. Containerization has emerged as a powerful solution to these challenges, allowing developers to package applications along with their dependencies into isolated units known as containers. This approach ensures consistency across various environments, from development to production, and facilitates rapid deployment and scaling.

The **Simple Container Deployment Manager** project is a Proof of Concept (POC) aimed at showcasing the principles and benefits of container management through a Python-based application. By leveraging Object-Oriented Programming (OOP) concepts, this project implements a structured system for managing Docker containers. It includes functionalities for creating, reading, updating, and deleting container specifications (CRUD operations) while simulating the deployment and monitoring of these containers.

This project not only highlights the practical applications of containerization but also serves as a foundational framework for more complex container orchestration tools. By effectively managing isolated application environments, the Simple Container Deployment Manager empowers developers to streamline their workflows, enhance application reliability, and optimize resource utilization in a microservices architecture.

## Project Overview

The **Simple Container Deployment Manager** project is a Proof of Concept (POC) designed to manage Docker containers through a structured Python-based application. This project focuses on implementing Object-Oriented Programming (OOP) principles and data structures to perform CRUD (Create, Read, Update, Delete) operations for container specifications and simulate container deployment management and status tracking.

## Objectives

The main objectives of this project are:

1. To create a system that allows for the management of Docker containers without the need for external frameworks or databases.
2. To implement CRUD operations for container specifications.
3. To simulate deployment, status tracking, and lifecycle management of Docker containers.
4. To illustrate the benefits of containerization and orchestration in software deployment.

## Key Concepts

### Containerization

Containerization allows applications to run in isolated environments, ensuring consistency across different stages of development, testing, and production. Each container contains all necessary dependencies, enabling seamless deployment and scalability.

### OOP Principles

The project utilizes OOP principles such as encapsulation, inheritance, and polymorphism to manage container specifications effectively. Two primary classes were defined:

* **ContainerSpec**: Represents the specifications of individual containers.
* **ContainerManager**: Handles the management of multiple containers, including CRUD operations and deployment status tracking.

**STEPS TO CREATE THE PROGRAM**

### 1. Class Design:

We'll define two main classes:

1. **ContainerSpec**: This class will store details of a container (name, image, etc.).
2. **ContainerManager**: This class will manage the containers, handle the CRUD operations for container specs, and simulate tracking their statuses.

### 2. CRUD Operations:

* **Create**: Add a new container specification.
* **Read**: Retrieve details of an existing container.
* **Update**: Modify the details of an existing container.
* **Delete**: Remove a container specification.

### 3. Tracking Container Status:

A function will simulate deploying, starting, and stopping containers and provide feedback on their status (e.g., "running", "stopped").

### Code:

Python

Version=3.18.2

import random

import time

# Class representing the specifications of a container

class ContainerSpec:

def \_\_init\_\_(self, container\_id, name, image, ports, env\_vars):

self.container\_id = container\_id # Unique ID for the container

self.name = name # Name of the container

self.image = image # Docker image to use

self.ports = ports # Port mappings

self.env\_vars = env\_vars # Environment variables

self.status = 'created' # Initial status of the container (e.g., 'created')

def \_\_str\_\_(self):

return f"Container ID: {self.container\_id}, Name: {self.name}, Image: {self.image}, Ports: {self.ports}, Env Vars: {self.env\_vars}, Status: {self.status}"

# Class to manage the containers and their statuses

class ContainerManager:

def \_\_init\_\_(self):

self.containers = {} # Dictionary to store container\_id -> ContainerSpec

# Create a new container specification and add it to the list

def create\_container(self, name, image, ports, env\_vars):

container\_id = random.randint(1000, 9999) # Generate a random container ID

new\_container = ContainerSpec(container\_id, name, image, ports, env\_vars)

self.containers[container\_id] = new\_container

print(f"Container {name} created with ID {container\_id}")

return container\_id

# Read details of an existing container

def get\_container(self, container\_id):

if container\_id in self.containers:

return self.containers[container\_id]

else:

return "Container not found"

# Update details of an existing container

def update\_container(self, container\_id, name=None, image=None, ports=None, env\_vars=None):

if container\_id in self.containers:

container = self.containers[container\_id]

if name:

container.name = name

if image:

container.image = image

if ports:

container.ports = ports

if env\_vars:

container.env\_vars = env\_vars

print(f"Container {container\_id} updated successfully.")

else:

print("Container not found")

# Delete a container specification

def delete\_container(self, container\_id):

if container\_id in self.containers:

del self.containers[container\_id]

print(f"Container {container\_id} deleted successfully.")

else:

print("Container not found")

# Simulate deploying a container (starting it)

def deploy\_container(self, container\_id):

if container\_id in self.containers:

container = self.containers[container\_id]

container.status = 'running'

print(f"Container {container\_id} is now running.")

else:

print("Container not found")

# Simulate stopping a container

def stop\_container(self, container\_id):

if container\_id in self.containers:

container = self.containers[container\_id]

container.status = 'stopped'

print(f"Container {container\_id} has been stopped.")

else:

print("Container not found")

# Track the status of a container

def track\_status(self, container\_id):

if container\_id in self.containers:

container = self.containers[container\_id]

print(f"Container {container\_id} status: {container.status}")

return container.status

else:

return "Container not found"

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

manager = ContainerManager()

# Create a container

container\_id = manager.create\_container(name="web\_app", image="nginx:latest", ports={80: 8080}, env\_vars={"ENV": "prod"})

# Get container details

print(manager.get\_container(container\_id))

# Update the container

manager.update\_container(container\_id, name="updated\_web\_app")

# Deploy the container

manager.deploy\_container(container\_id)

# Track the container's status

manager.track\_status(container\_id)

# Stop the container

manager.stop\_container(container\_id)

# Track the status again

manager.track\_status(container\_id)

# Delete the container

manager.delete\_container(container\_id)

def display\_menu():

print("\nContainer Management System")

print("1. Create Container")

print("2. Get Container Details")

print("3. Update Container")

print("4. Deploy Container")

print("5. Stop Container")

print("6. Track Container Status")

print("7. Delete Container")

print("8. Exit")

def main():

manager = ContainerManager()

while True:

display\_menu()

choice = input("Enter your choice (1-8): ")

if choice == '1':

name = input("Enter container name: ")

image = input("Enter Docker image: ")

ports = input("Enter port mappings (e.g., {80: 8080}): ")

env\_vars = input("Enter environment variables (e.g., {'ENV': 'prod'}): ")

container\_id = manager.create\_container(name, image, eval(ports), eval(env\_vars))

elif choice == '2':

container\_id = input("Enter container ID: ")

print(manager.get\_container(container\_id))

elif choice == '3':

container\_id = input("Enter container ID: ")

name = input("Enter new name (leave blank to keep current): ")

image = input("Enter new image (leave blank to keep current): ")

ports = input("Enter new ports (leave blank to keep current): ")

env\_vars = input("Enter new environment variables (leave blank to keep current): ")

manager.update\_container(container\_id,

name if name else None,

image if image else None,

eval(ports) if ports else None,

eval(env\_vars) if env\_vars else None)

elif choice == '4':

container\_id = input("Enter container ID: ")

manager.deploy\_container(container\_id)

elif choice == '5':

container\_id = input("Enter container ID: ")

manager.stop\_container(container\_id)

elif choice == '6':

container\_id = input("Enter container ID: ")

manager.track\_status(container\_id)

elif choice == '7':

container\_id = input("Enter container ID: ")

manager.delete\_container(container\_id)

elif choice == '8':

print("Exiting the Container Management System.")

break

else:

print("Invalid choice. Please try again.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

***OUTPUT***

Container web\_app created with ID 7720

Container ID: 7720, Name: web\_app, Image: nginx:latest, Ports: {80: 8080}, Env Vars: {'ENV': 'prod'}, Status: created

Container 7720 updated successfully.

Container 7720 is now running.

Container 7720 status: running

Container 7720 has been stopped.

Container 7720 status: stopped

Container 7720 deleted successfully.

Container Management System

1. Create Container

2. Get Container Details

3. Update Container

4. Deploy Container

5. Stop Container

6. Track Container Status

7. Delete Container

8. Exit

Enter your choice (1-8): 1

Enter container name: 99

Enter Docker image: 321

Enter port mappings (e.g., {80: 8080}): 24897

Enter environment variables (e.g., {'ENV': 'prod'}): 54

Container 99 created with ID 7674

Container Management System

1. Create Container

2. Get Container Details

3. Update Container

4. Deploy Container

5. Stop Container

6. Track Container Status

7. Delete Container

8. Exit

Enter your choice (1-8): 2

Enter container ID: 7674

Container not found

Container Management System

1. Create Container

2. Get Container Details

3. Update Container

4. Deploy Container

5. Stop Container

6. Track Container Status

7. Delete Container

8. Exit

Enter your choice (1-8): 3

Enter container ID: 7674

Enter new name (leave blank to keep current): 4567

Enter new image (leave blank to keep current): 25

Enter new ports (leave blank to keep current): 646

Enter new environment variables (leave blank to keep current): 32123

Container not found

Container Management System

1. Create Container

2. Get Container Details

3. Update Container

4. Deploy Container

5. Stop Container

6. Track Container Status

7. Delete Container

8. Exit

Enter your choice (1-8): 4567

Invalid choice. Please try again.

Container Management System

1. Create Container

2. Get Container Details

3. Update Container

4. Deploy Container

5. Stop Container

6. Track Container Status

7. Delete Container

8. Exit

Enter your choice (1-8): 8

### Breakdown:

1. **ContainerSpec class**:
   * Stores container details such as container\_id, name, image, ports, env\_vars, and status.
2. **ContainerManager class**:
   * Manages the containers:
     + **CRUD operations**: create\_container, get\_container, update\_container, delete\_container.
     + **Deployment operations**: deploy\_container, stop\_container, track\_status.
3. **Simulated Actions**:
   * deploy\_container and stop\_container simulate starting and stopping the container.
   * track\_status checks and returns the current status of the container.

### Notes:

* This solution mocks Docker operations with simple status changes and does not interact with Docker itself.
* The ContainerManager allows for managing multiple containers using their unique IDs.

This is a basic implementation of a container manager for a POC. You can further extend it by integrating actual Docker APIs (using Python libraries like docker-py) or adding persistence to store container specifications.

## Implementation Details

### Class Definitions

1. **ContainerSpec Class**
   * Attributes: container\_id, name, image, ports, env\_vars, status.
   * Responsibilities: Store container details and provide a string representation of the container.
2. **ContainerManager Class**
   * Attributes: containers (a dictionary to store container specifications).
   * Methods:
     + create\_container: Creates a new container specification and assigns a unique ID.
     + get\_container: Retrieves details of a specific container by ID.
     + update\_container: Modifies existing container specifications.
     + delete\_container: Removes a container specification.
     + deploy\_container: Simulates the deployment of a container.
     + stop\_container: Simulates stopping a container.
     + track\_status: Reports the current status of a container.

## Real-Life Application and Benefits

### Use Case

This container manager could be employed in various environments, including:

* **Development and Testing**: Developers can deploy and manage application services quickly.
* **Microservices Architecture**: Each service can be managed as a separate container, enhancing modularity and ease of updates.
* **Cloud Deployments**: This manager can serve as a lightweight orchestration tool for deploying applications on cloud platforms.

Imagine a **food truck festival** where different food trucks serve various types of cuisine. Each truck is self-contained, has its own kitchen, cooking equipment, and serves a specific menu. The organizer of the festival needs a way to manage and monitor each food truck’s setup, operation, and status during the festival.

1. **Container Specs as Food Trucks**:
   * Each food truck represents a **Docker container** that runs an application (in this case, it could be a web service, a database, etc.).
   * Each truck (container) has specific properties:
     + **Truck Name (Container Name)**: The name of the food truck (container), e.g., "Taco Truck".
     + **Type of Cuisine (Container Image)**: The type of service the truck provides (Docker image), e.g., "nginx

" could represent a web server, like "Taco Truck" serves tacos.

* + - **Stall Location (Port Mappings)**: Where the truck is located in the festival grounds, mapped to the host ports where the container is accessible.
    - **Menu/Ingredients (Environment Variables)**: The configuration or setup each truck needs to operate. For example, food ingredients can be likened to environment variables like "ENV=prod".

1. **Container Manager as the Festival Organizer**:
   * The **Container Manager** is the festival organizer responsible for:
     + **Deploying (Starting)** each food truck: Ensuring each truck is operational, serving food to customers (the container runs a service).
     + **Tracking Status**: Monitoring the status of each food truck to ensure they are working properly (checking if containers are "running" or "stopped").
     + **Updating Information**: If a truck wants to change its menu or location, the organizer updates these details (updating container specifications).
     + **Removing (Deleting)** a truck: Once the festival ends, the organizer shuts down and removes food trucks (deletes containers).

### Purposes of a Simple Container Deployment Manager:

1. **Application Deployment**:
   * Just like food trucks serve food to customers, software applications need to be deployed and made available for users or other systems. The manager helps ensure that each application (container) is deployed in the right configuration.
2. **Consistent Environments**:
   * Each food truck comes pre-configured with its own kitchen and supplies, just like each Docker container runs a consistent environment (with specific software versions, dependencies, and configurations). This avoids conflicts that may arise when running the same application in different environments.
3. **Scaling Applications**:
   * In the same way that a festival can add more food trucks to meet higher customer demand, a container deployment manager can add or duplicate containers to scale applications based on load.
4. **Tracking Status and Monitoring**:
   * A key part of managing food trucks is ensuring they’re operational throughout the festival. In the same way, a container manager tracks the status of containers to ensure applications are running smoothly, and can restart them if they stop (track statuses like "running", "stopped", etc.).
5. **Ease of Updates and Maintenance**:
   * If a food truck changes its menu mid-festival, the organizer needs to update this information quickly. Similarly, the container manager can update container configurations (like changing environment variables or container images) and redeploy the application without significant downtime.
6. **Isolated Environments**:
   * Each food truck is self-contained, meaning issues with one truck don’t affect the others. In the container world, this is the same as **container isolation**: containers run independently, so an issue in one container (e.g., a crash) won’t bring down other containers running different services.

### Practical Example:

Let's say you're managing a microservices-based web application. Each service runs in a separate container:

* **Web Frontend** runs in a container using an nginx image.
* **User Authentication Service** runs in another container.
* **Database** runs in yet another container using a PostgreSQL image.

Your **Container Deployment Manager** would allow you to:

* **Deploy** these containers individually based on the specifications you've set (ports, environment variables).
* **Monitor** whether each service (container) is running properly.
* **Scale** the web frontend if there is increased traffic by deploying additional instances.
* **Stop** or remove containers when they’re no longer needed (e.g., after a low-traffic period or maintenance).

### Summary:

A **Simple Container Deployment Manager** is essentially a system that helps you organize, deploy, monitor, and manage isolated software environments (containers). In real life, it’s like managing self-contained services (like food trucks) in a festival, ensuring they run independently, can be scaled as needed, and are easy to update or take down when necessary.

This is especially useful in modern software development, where applications are often composed of many small, independently running services (microservices) that need to be efficiently managed and tracked.

### Benefits

1. **Consistency**: Each container provides a consistent environment, eliminating "it works on my machine" issues.
2. **Scalability**: Containers can be easily scaled up or down based on demand.
3. **Isolation**: Containers run independently, so issues in one do not affect others.
4. **Rapid Deployment**: Containers can be deployed, stopped, and restarted quickly, facilitating agile development practices.

## Conclusion

The **Simple Container Deployment Manager** POC effectively demonstrates the principles of containerization and OOP in a manageable, easy-to-understand format. It serves as a foundation for more complex container orchestration tools and highlights the potential benefits of using Docker containers in modern software development practices. Further extensions could include integration with actual Docker APIs and persistent storage solutions to enhance functionality.

The Simple Container Deployment Manager provides valuable support for a cloud services provider by:

● Effectively managing container specifications to ensure that all deployed containers are optimized for performance and resource usage.

● Streamlining the deployment process, allowing for quick and efficient updates to client applications.

● Monitoring container status, enabling proactive maintenance and ensuring high availability of services offered to clients.

This system helps maintain a robust infrastructure, allowing the cloud provider to deliver reliable containerized applications to its customers, enhancing both service quality and client satisfaction

REFERENCES

[WWW.GOOGLE.COM](http://WWW.GOOGLE.COM)

WWW.CHATGPT.COM

WWW.NINGA AI.COM