THE YESTERDAY TEAM | CSE-D | 5TH SEM

HOTEL BOOKING APP

PROJECT SUMMARY

REPORT DATE	PROJECT NAME	PROJECT MANAGER
09-11-2024	Continuous Integration, Delivery, and Deployment In AWS, Netlify,GitHub,Dockers	Varun Ramesh Hosapet

EXECUTIVE SUMMARY

This project focused on developing a scalable, reliable hotel booking application, leveraging Docker for containerization to streamline deployment and improve operational efficiency. Using Docker, we packaged the application into isolated containers, ensuring consistent performance across different environments (development, testing, and production) and enabling faster, more reliable deployments.

Key objectives included optimizing resource usage, enhancing scalability, and increasing fault tolerance. Docker allowed us to adopt a microservices architecture, separating components like booking, payment, and user authentication, which simplified maintenance and improved security.

By integrating Docker into our CI/CD pipeline, we achieved continuous integration and delivery, enabling automated testing and deployment. This streamlined workflow significantly reduced deployment time and minimized environment-related issues.

Overall, Docker's impact on the project was transformative, providing a foundation for an agile, resilient, and efficient application that meets modern software standards. The experience gained in Docker and container management will be valuable for future projects, paving the way for more robust and scalable applications.

PROJECT OVERVIEW

TASK	% DONE	DUE DATE	DEVOPS FACILITATOR	MILESTONES
Planning Stage	10%	6.11.2024	Mr Vishal N	Research and Development
Development Stage	20%	6.11.2024	Mr Vishal N	Research and Development
Deployment Stage	20%	7.11.2024	Mr Vishal N	Research and Development
Testing Stage	20%	8.11.2024	Mr Vishal N	Research and Development
Monitoring Stage	20%	9.11.2024	Mr Vishal N	Research and Development
Feedback Stage	10%	9.11.2024	Mr Vishal N	Research and Development

MAN-HOURS

CATEGORY	SPENT	% OF TOTAL	ON TRACK?	NOTES
Planning and Assessment	5	13.9	Yes	Completed with efficiency
Requirements gathering:	2	5.5	Yes	Completed with efficiency
Application assessment:	1.5	4.1	Yes	Completed with efficiency
DevOps strategy planning	1.5	4.1	Yes	Completed with efficiency
Tool selection and configuration	2	5.6	Yes	Completed with efficiency
Infrastructure Setup	9	25	Yes	Completed with efficiency
Cloud infrastructure setup (AWS/Azure/GCP)	3	8.3	Yes	Completed with efficiency
Containerization (Docker):	2	5.5	Yes	Completed with efficiency
Orchestration (Kubernetes)	2	5.5	Yes	Completed with efficiency
Monitoring and logging setup	2	5.5	Yes	Completed with efficiency
Application Integration	7	19.4	Yes	Completed with efficiency
Code repository setup (Git)	1	2.7	Yes	Completed with efficiency
Continuous Integration/Continuous Deployment (CI/CD) pipeline setup	2	5.5	Yes	Completed with efficiency
Automated testing setup	1.5	4.1	Yes	Completed with efficiency
Vulnerability management	1	2.7	Yes	Completed with efficiency
Security and Compliance	1.5	4.1	Yes	Completed with efficiency
Deployment automation	4	11.1	Yes	Completed with efficiency
Security assessment	1	2.7	Yes	Completed with efficiency
Compliance setup	0.5	1.3	Yes	Completed with efficiency
Access control and identity management	0.5	1.3	Yes	Completed with efficiency
Testing and Quality Assurance	5	13.9	Yes	Completed with efficiency
Test planning	1	2.7	Yes	Completed with efficiency
Test execution	1.5	4.1	Yes	Completed with efficiency
Defect tracking and resolution:	1	2.7	Yes	Completed with efficiency
Quality assurance	1.5	4.1	Yes	Completed with efficiency
Deployment and Maintenance	4	11.1	Yes	Completed with efficiency
Deployment planning	1	2.7	Yes	Completed with efficiency
Deployment execution	1	2.7	Yes	Completed with efficiency
Post-deployment monitoring	1	2.7	Yes	Completed with efficiency
Maintenance and support	1	2.7	Yes	Completed with efficiency

STAKEHOLDERS

STAKEHOLDER	USN	KEY RESPONSBILITY AREA
Sanmith Adwik	4NI22CS193	Containerization (Docker)
Sathvik Prasad M	4NI22CS195	Code Repository Setup (Git)
Varun R Hosapet	4NI23CS423	DevOps Strategy Planning

Shravan M Hiremath	4NI22CS202	**Deployment Automation**
Suraj Uday Shanbag	4NI22CS230	**Cloud Infrastructure Setup**
Chethan U	4NI23CS402	**Requirements Gathering**

PROJECT OVERVIEW

The DevOps project aimed to improve the efficiency, reliability, and scalability of Simple Hotel Booking software development and deployment processes. The project focused on implementing DevOps practices, automating CI/CD pipelines, and ensuring continuous monitoring and feedback.

KEY OBJECTIVES:

1. Ensure Consistent Deployments: Achieve consistent application behavior across development, testing, and production environments.
2. Improve Scalability: Enable easy scaling of the app to handle peak user loads by deploying additional containers as needed.
3. Enhance Deployment Speed: Reduce the time required for deploying new versions or updates.
4. Optimize Resource Usage: Minimize CPU and memory consumption through efficient containerization.
5. Increase Fault Tolerance: Quickly recover from failures by restarting individual containers without affecting the entire application.
6. Streamline CI/CD: Integrate Docker into the CI/CD pipeline to automate testing and deployment processes.
7. Facilitate Microservices Architecture: Use Docker to isolate services (e.g., booking, payment) for easier management and maintenance.

BENEFITS:

- Efficient Deployment: Docker allows the app to be packaged with all its dependencies, ensuring it runs consistently across different environments. This minimizes issues related to environment setup, saving time for both development and production teams.
- Scalability: Docker containers can be easily scaled up or down, which is useful for handling fluctuating loads in a hotel booking app. For instance, during peak travel seasons, additional containers can be deployed to manage increased user activity.
- Isolation and Security: Each microservice of the hotel booking app (like booking, payment, user authentication) can run in its own Docker container. This isolation improves security and reduces the risk of issues in one component affecting others.

LESSONS LEARNED:

- Containerization Best Practices: Working with Docker taught us the importance of keeping containers lightweight and avoiding the use of unnecessary packages. This resulted in faster startup times and lower resource usage.
- Efficient Multi-Stage Builds: To optimize the Docker images, we implemented multi-stage builds, reducing the final image size. This was a valuable lesson in creating efficient and deployable container images.
- Troubleshooting and Debugging: Docker introduced new debugging challenges, such as handling network issues between containers. Learning to use Docker commands for troubleshooting and working with logs within containers was essential for resolving these issues.

FUTURE RECOMMENDATIONS:

- Automate Scaling with Kubernetes: To fully leverage Docker, integrating Kubernetes would allow for automated scaling, load balancing, and management of containerized services. This would help manage the hotel booking app's resources dynamically and efficiently.
- Implement Docker Swarm for Redundancy: Adding Docker Swarm could provide built-in load balancing and redundancy, which would ensure that the app remains available even if certain containers or services fail.
- Adopt Advanced Monitoring Tools: Using tools like Prometheus or Grafana with Docker can enhance monitoring of container health, resource utilization, and potential bottlenecks, making it easier to manage and optimize containerized services.

CONCLUSION:

- Docker significantly improved the development and deployment process for the hotel booking app. By using Docker, we achieved a portable, scalable, and efficient solution that meets modern application deployment standards. Docker's containerization allowed us to break down the app into microservices, improving maintainability and reliability. This experience has given the team a solid foundation in Docker and containerized deployments, which will be invaluable in future projects.

METRICS:

1. Deployment Speed: Time to deploy a new version.
2. Environment Consistency: Number of environment-related issues.
3. Resource Utilization: CPU and memory usage per container.
4. Scalability: Peak concurrent users handled.
5. Recovery Time: Time to restore services after failure.
6. Build Efficiency: Average Docker image size and build time.
7. Deployment Frequency: Successful deployments per week/month.