|  |  |
| --- | --- |
| Internship Project Title | Develop docker container using docker compose for application development using Java stack |
| Name of the Company | Tcs-ion |
| Name of the Industry Mentor | Kavita\_Iyer, Abhijit\_Lahiri |
| Name of the Institute | Teerthankar Mahaveer University,Moradabad |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Start Date | End Date | | Total Effort (hrs.) | | Project Environment | Tools used |
| 01-07-2024 | 29-09-2024 | | 125 | | Windows 10, Android 9,java | Docker, IDE or Code Editor,Trivyetc |
| Milestone # | 1  2  3  4  5  6  7 | Milestone: | | **Setup the Java Application**  **Database Container Setup**  **Container Build and Run**  **Environment Configuration & Networking**  **Logging and Monitoring**  **Data Persistence and Backups**  **CI/CD Integration** | | |

|  |
| --- |
| * *Acknowledgements*   *I would like to express my gratitude to my mentors and colleagues for their invaluable guidance and support throughout this project. Special thanks to [Debashish Roy] for their continuous encouragement and insightful feedback.* |

|  |
| --- |
| * *Objective*   The objective of this project is todevelop docker container using docker compose for application development using Java stack |

|  |
| --- |
| * *Introduction / Description of Internship*   During my internship, I worked on developing a Docker container using Docker Compose for a Java-based application development stack. This project aims to provide a comprehensive understanding of containerization, Docker, and Docker Compose. You will create a Docker container that can be used for Java-based application development, testing, and deployment. |

|  |
| --- |
| * *Internship Activities* * **Set Up Your Java Application :** Let's assume you have a simple Java application, such as a Spring Boot application. * **Create a Dockerfile:** The docker file is used to create a image of your java application. * **Create a docker-compose.yml File**: This file is used to define and manage multi-container Docker applications. * **Build and Run the Docker Containers**: Build and Run the Docker Containers. * **Access Your Application :**Once the containers are up and running, your application should be accessible at http://localhost:8080. |

|  |
| --- |
| * *Approach / Methodology* * **Define Application Requirements**: Identify the components of your Java stack (e.g., Java, Maven/Gradle, database). Determine the required development tools and environments.. * **Design Container Architecture:**Plan the architecture including all necessary services (e.g., application server, database, message broker).Define the interaction between these services. * **Write Docker file for each services**: Create Dockerfiles that define the environment for each component.Optimize the Dockerfile for faster builds and smaller images. * **Create Docker Compose Configuration** :Use Docker Compose to orchestrate multi-container applications.Configure the services to interact with each other. * **Enable Development Workflow:**Set up volume mounts for code sharing between the host and containers.Configure environment variables for development settings. Include tools for live reloads, debugging, and testing. * **Test and Iterate:**Perform testing within the container environment.Make necessary adjustments to Dockerfiles and Docker Compose as required. |

|  |
| --- |
| * *Assumptions* * The development environment requires tools like Maven or Gradle for building the Java application, along with a JDK (Java Development Kit) appropriate for the project (e.g., JDK 17). * The application will require a relational database (e.g., MySQL, PostgreSQL) for data persistence, which will be containerized as a separate service. |

|  |
| --- |
| * *Exceptions / Exclusions* * when the container runs out of memory, either because the JVM heap size is too large or because the container does not have enough allocated memory. * when the application or Docker process tries to access a file or directory without the proper permissions. It’s common when dealing with mounted volumes or when the application is trying to access a system resource. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| * *Charts, Table, Diagrams*   Docker**Performance Metrics Dashboard (Chart)**   * **Type**: Line/Bar Chart * **Description**: A chart displaying CPU, memory usage, network I/O, and storage performance metrics of Docker containers running Java applications over time. * **Purpose**: To monitor resource usage and identify performance bottlenecks during load testing or production. * **Tools**: Prometheus (data collection) and Grafana (visualization).   EXAMPLE:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Metric | |  | | --- | | Container (1) | | Container(2) | Container(3) |  | | CPU USAGE | 70% | 55% | 80% |  | | MEMORY USAGE | 600mb | 550mb | 750mb |  | | NETWORK I/O | 100MB/S | 120MB/s | 90MB/S |  |   **Diagram**:   * X-axis: Time * Y-axis: Resource usage (CPU, Memory, etc.) * Multiple lines representing different containers.   **Horizontal Pod Autoscaler Metrics (Chart)**   * **Type**: Line Chart * **Description**: A chart that tracks the number of containers or pods over time based on CPU or memory usage. * **Purpose**: To show how Kubernetes or Docker Swarm auto-scales based on resource utilization. * **Metrics**:   + CPU usage   + Memory usage   + Number of containers/pods  |  |  |  | | --- | --- | --- | | TIME | CPU USAGE | PODS ACTIVE | | 0 | 50% | 2 | | 5 | 75% | 3 | | 10 | 90% | 4 | |

|  |
| --- |
| * *Algorithms* * **Bin Packing Algorithm (for Resource Scheduling)** * **Objective**: Efficiently allocate resources (e.g., CPU, memory) to containers while minimizing the number of nodes. * **How it works**:   + Containers (or tasks) are assigned to the node with the least available resources, provided the container's resource requirements can be met.   + Attempts to "pack" containers into fewer nodes to minimize the number of active nodes. * **Use Case**: Kubernetes and Docker Swarm use variations of this algorithm to place containers optimally across nodes in a cluster.   **Steps**:   1. For each container, find the node with sufficient available resources. 2. Check if the container's resource demands fit within the node's remaining capacity. 3. Place the container on the node that leaves the least remaining space after the container is scheduled.   ­­­------  **Round Robin (for Load Balancing)**   * **Objective**: Distribute incoming requests evenly among containers. * **How it works**:   + Requests are distributed sequentially to each container in a circular manner.   + After reaching the last container, the next request is sent to the first container, and so on. * **Use Case**: Load balancing among multiple instances of a Java microservice running in Docker containers.   **Steps**:   1. Track the list of containers handling requests. 2. For each new request, assign it to the next container in the list. 3. When the last container in the list is reached, loop back to the first container.   **Least Connections Algorithm (for Load Balancing)**   * **Objective**: Direct new requests to the container with the fewest active connections. * **How it works**:   + A load balancer monitors the number of active connections for each container.   + Requests are sent to the container that has the least number of active connections to ensure balanced load distribution. * **Use Case**: Useful in Docker-based applications where containers can experience uneven traffic loads.   **Steps**:   1. For each request, query the number of active connections for each container. 2. Identify the container with the least number of active connections. 3. Route the new request to the identified container. |

|  |
| --- |
| * *Challenges & Opportunities* * **Challenges**: Docker and Docker Compose introduce new concepts like containerization, orchestration, and networking, which can be complex for developers who are new to these technologies.. * **Opportunities**: Once mastered, these tools significantly simplify development, testing, and deployment processes. |

|  |
| --- |
| * *Risk Vs Reward* * **Risk**: Containers, if not properly managed, can introduce security vulnerabilities. Public images might contain outdated software or security flaws, and improper configuration can lead to unauthorized access. * **Reward**: Containers provide process and resource isolation, meaning that issues in one container (e.g., memory leaks or crashes) do not affect other containers. This isolation is crucial for maintaining system stability and security. |

|  |
| --- |
| * *Reflections on the Internship*   During the internship, learning to use Docker and Docker Compose provided me with hands-on experience in containerization, a key technology in modern software development. This experience helped me understand how to package and deploy Java applications consistently across different environments, which is crucial for any developer entering the industry today. |

|  |
| --- |
| * *Recommendations* * Ensure a solid understanding of Docker fundamentals, including creating Dockerfiles, managing images, and working with containers. * Explore and utilize the full range of Docker Compose features, such as defining multi-container applications, using environment variables, and managing service dependencies. * This will help in creating robust and maintainable configurations. |

|  |
| --- |
| * *Outcome / Conclusion*   The use of Docker and Docker Compose during the internship provided a comprehensive learning experience, enhancing both technical skills and professional growth. The outcomes reflect improved development efficiency, a stronger understanding of modern development tools, and better preparedness for future career opportunities in software development and operations. |

|  |
| --- |
| * *Enhancement Scope* * Implement advanced networking setups such as overlay networks for multi-host communication or use network policies to control traffic between containers. |

|  |
| --- |
| * *Link to code and executable file*   [***https://github.com/BhumikaJain-25/docker/blob/main/Containerizing-an-application-using-docker.md***](https://github.com/BhumikaJain-25/docker/blob/main/Containerizing-an-application-using-docker.md) |

|  |
| --- |
| * *Research questions and responses* * **How can Docker and Docker Compose be optimized to improve the performance, security, and scalability of Java-based microservices applications in development and production environments?**   + This research investigates methods to enhance Docker and Docker Compose configurations for Java microservices applications, focusing on optimizing performance, bolstering security, and ensuring scalability. By examining best practices for image optimization, security configurations, and scalability strategies, the study aims to identify effective techniques for improving containerized application performance and security while facilitating seamless scaling. The findings will provide actionable insights for leveraging Docker and Docker Compose more effectively in both development and production scenarios.. |