ASSIGNMENT

Topic:

CONTAINERS 101 – KUBE ACADEMY

Submitted To:

Ms. Navya Mol K T
Assistant professor
Department of Computer Application
Ajce

Submitted By:

Akash J Thomas
S3 RMCA A
Roll_no:06

Overview:

1. Container concepts

2. Anatomy of a container

3. Build, run and test a container image

4. Screenshot

Container Concepts:

In the realm of computing, a "container" typically denotes a compact, self-contained, and executable software package encompassing all essential components for running a software application. This includes the application code, runtime, system tools, libraries, and configuration settings. Containers are highly regarded in software development and deployment due to their ability to provide a consistent and isolated environment, simplifying the development, testing, and deployment of software across diverse environments.

Key container concepts include:

1. Containerization: The process of packaging an application and its dependencies into a container image, ensuring portability and consistency.

2. Container Image: A self-contained package that holds an application and all its dependencies, often based on a predefined base image.

3. Docker: A well-known containerization platform with tools and a client-server architecture.

4. Kubernetes: A widely-used container orchestration platform automating application deployment, scaling, and management.

5. Container Registry: A repository for storing and sharing container images, such as Docker Hub or Amazon ECR.

6. Container Orchestration: The automated management of containerized applications, exemplified by Kubernetes.

7. Microservices: An architectural approach involving breaking down applications into smaller, independent services running in containers.

8. Isolation: Containers offer process and file system isolation for enhanced security.

9. Portability: Containers are highly portable and can be deployed in various environments without compatibility concerns.

10. Resource Efficiency: Containers share the host OS kernel, making them resource-efficient compared to traditional virtual machines.

11. Lifecycle Management: Containers are easily started, stopped, and removed, simplifying application management.

12. Orchestration Tools: Besides Kubernetes, alternatives like Docker Swarm, Apache Mesos, and Amazon ECS assist in managing containerized applications.

13. Security: Ensuring container security involves keeping containers updated, maintaining proper isolation, and using trusted images.

Anatomy of Containers:

The structure of a container refers to its core components, encapsulating everything needed to run an application, including code, runtime, dependencies, and configuration. Key elements of container anatomy include:

1. Container Image:

   - Application Code: The application code written in any programming language.

   - Runtime: The runtime environment (e.g., Python, Node.js) or execution environment (e.g., JVM for Java).

   - System Libraries: Necessary system libraries and dependencies.

2. Filesystem:

   - Containers possess their isolated filesystem, typically read-only, with a separate read-write layer for persistent data.

3. Container Configuration:

   - Defined in a configuration file (e.g., Dockerfile for Docker containers) specifying environment variables, network settings, ports, and entrypoints.

4. Operating System Kernel:

   - Containers share the host OS kernel for efficiency and essential services.

5. Container Runtime:

   - Responsible for running and managing containers, including Docker, containerd, and rkt

6. Isolation:

   - Achieved through technologies like namespaces and cgroups, providing process and resource isolation.

7. Network Namespace:

- Each container has its network stack, IP address, and network interfaces, often connected to bridge networks.

8. Process Isolation:

- Containers run as separate processes within isolated namespaces to prevent direct interaction between containers.

9. Container Orchestration:

- Tools like Kubernetes coordinate multiple containers for deploying, scaling, and managing distributed applications.

10. Container Registry:

- Container images are typically stored in registries such as Docker Hub for versioning and distribution.

Building, Running, and Testing a Container Image:

The process involves several steps, illustrated using Docker as a popular containerization tool. Ensure Docker is installed on your system before proceeding.

1. Create Your Application:

- Begin with the application or service you wish to containerize, having the application code and dependencies ready.

2. Write a Dockerfile:

- In the same directory as your application code, create a Dockerfile to specify how the container image should be built.

3. Build the Container Image:

- Navigate to the directory with your Dockerfile and application code in a terminal. Build the image with the command `docker build -t my-node-app:1.0`.

4. Run the Container:

- Once the image is built, start a container with the command `docker run -p 8080:80 my-node-app:1.0`.

5. Test the Container:

- Access your application in the container via a web browser or tools like `curl` at `http://localhost:8080`.

6. Cleanup:

- After testing, stop and remove the container using `docker stop <container_id>` and `docker rm <container_id>`. Find the `<container_id>` with `docker ps -a`.

Screenshot