Containerization

Question:

What is a Container? How does it differ from Virtualization?

Containerization

Container

Is a lightweight, portable box that holds everything an application needs to run smoothly, such as code, libraries, and system tools.

Containerization

Virtualization

Virtualization is creating multiple computers within a single computer. Each virtual machine acts like an independent computer with its own operating system (OS), applications, and resources sharing same underlying hardware.

Containerization

Differences

	Container	Virtualization
OS Level	Shares host OS & Kernel	Independent OS & Kernel
Resource Utilization	Lightweight	High Overload
Isolation	Only process level	Stronger Isolation
Portability & Consistency	Highly Consistent and Portable	Less Standardized and less Portable
Use Cases	Microservices, Faster deployments	Legacy Apps, Full isolation

Containerization

Question:

What are the main components of Docker architecture?

Containerization

Docker Engine
Runtime environment for Docker containers

Docker Daemon:

- Manages docker objects
- Receives docker API requests
- Container lifecycle

REST API:

- Provide endpoints
- Allows interaction with Docker daemon
- Enables users to manage containers

CLI:

- Interacts with users
- Translates the commands to API requests

Containerization

Docker Engine
Runtime environment for Docker containers

Functionalities

- Container Management
- Image Handling
- Networking and Volumes
- Interfacing with Host OS

Containerization

Docker Objects
Core building blocks

- Images – Templates with application libraries and code
- Containers – Runnable instances of docker images
- Volumes – Persistent data storage
- Networks – Enable secure communication
- Services – Scale applications across multiple containers

Containerization

Docker Registries

Repositories to store docker images

Public Registries

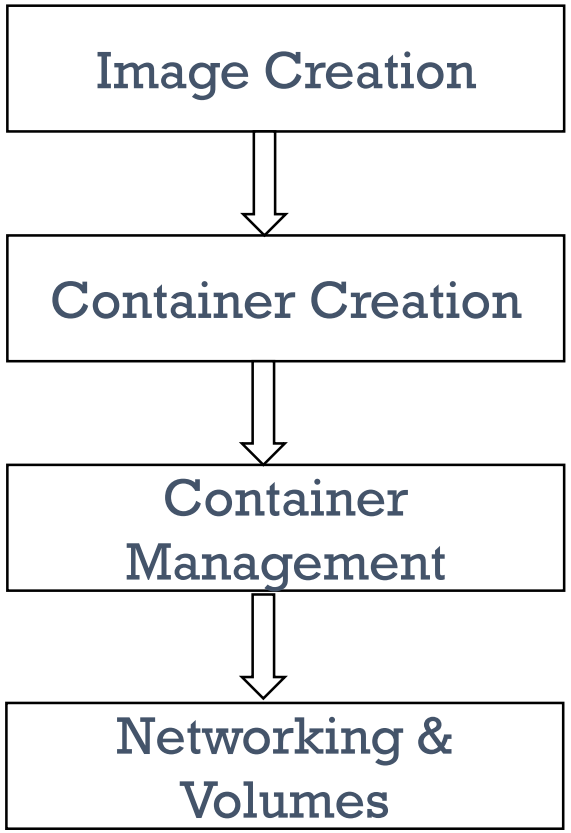
- Docker Hub
- Community Repositories

Private Registries

- Self Hosted
- Third Party

Containerization

Flow of Operation



Containerization

Question:

What is the purpose of a Dockerfile? What are the components of a Dockerfile?

Containerization

Dockerfile - Set of instructions to build a docker image

Purpose -

- Standardized Builds
- Automated Image Creation
- Efficient Deployment

Containerization

Components of a Dockerfile

- Base Image
- RUN Instruction
- COPY and ADD instruction
- Execution and Runtime Configuration
- Expose Instruction
- ENV Instruction
- WORKDIR Instruction
- User Configuration

```
# Use a base image from Docker Hub
FROM ubuntu:latest

# Set working directory
WORKDIR /app

# Copy application code
COPY . /app

# Install dependencies
RUN apt-get update && apt-get install -y \
    package1 \
    package2

# Expose port 8080
EXPOSE 8080

# Set the default command to run when the container starts
CMD ["node", "app.js"]
```

Containerization

Example docker run command

```
docker run -d \
    --name my_container \
    -p 8080:80 \
    -v /host/folder:/container/folder \
    -e ENV_VARIABLE=value \
    --restart=unless-stopped \
    my_image:latest
```

Containerization

Question:

What are some best practices for Docker container security?

Containerization

Best Practices -

- **Use Official Images and Regular Updates**
- **Minimal Image Scope**
- **Container Isolation and Limitation**
- **Secure Image Building**
- **Implement Image Scanning and Vulnerability Management**
- **Secure Configuration and Secrets Management**
- **Network Segmentation and Firewalls**
- **Container Runtime Security**
- **CI/CD Security**
- **Regular Audits**

Containerization

Tools to scan Docker containers -

- Clair
- Trivy
- Anchore Enginer
- Synk Container
- OpenSCAP

Containerization

Question:

What are some best practices for Docker container security?

Containerization

Bridge Network – Containers on the same host can communicate with each other

Characteristics -

- Each containers gets its own IP Address
- Communicate using container name or IP
- Devices outside the network can only communicate via port mapping

Containerization

Bridge Network – Containers on the same host can communicate with each other

Usage -

- Local Application Testing and Development
- Containerized Environment

Command: `docker run --network bridge my_container`

Containerization

Host Network – Container share the network namespace with host system

Characteristics -

- Container's share host's network stack
- Use host's network interfaces directly
- Use host's ports directly with no need to port mapping

Containerization

Host Network – Container share the network namespace with host system

Usage:

- High Performance
- Used with applications that need direct access to host network

Command: `docker run --network host my_container`

Containerization

Overlay Network – Allows communication between containers running on different hosts or nodes within a cluster

Characteristics -

- Network spanning across multiple hosts
- Containers on different hosts communicate
- Overlay encapsulation for communication

Containerization

Overlay Network – Allows communication between containers running on different hosts or nodes within a cluster

Usage :

- Used in Kubernetes or Docker Swarm
- Simplify deployment of applications

Command: `docker network create –driver=overlay my_overlay_network`

Containerization

MacVLAN Network – It allows containers to have their own MAC addresses, making them appear as individual physical devices on the network.

Characteristics -

- Individual MAC Address
- Direct Network Access
- Isolation

Containerization

MacVLAN Network – It allows containers to have their own MAC addresses, making them appear as individual physical devices on the network.

Usage -

- Legacy Applications
- Network Segmentation
- Performance centric applications

Command: `docker network create -d macvlan --subnet=192.168.1.0/24 --gateway=192.168.1.1 -o parent=eth0 my_macvlan_network`

Containerization

None Network – Used when containers don't require a network.

Characteristics -

- Isolate network entirely from network

Containerization

Question:

You have a legacy application. How would you containerize it using Docker, considering dependencies and compatibility issues?

Containerization

- Dependency Analysis
- Prepare a Dockerfile
- Resolve Compatibility Issues
- Data and Configuration Handling
- Testing and Validation
- Security Considerations
- Documentation and Maintenance
- Incremental Approach

Containerization

Legacy Application Setup

- Python Flash Application
- MySQL Database

Containerization

Dockerfile

```
# Use a Python base image
FROM python:3.9

# Set working directory in the container
WORKDIR /app

# Copy application files to the container
COPY . /app

# Install dependencies
RUN pip install -r requirements.txt

# Expose port
EXPOSE 5000

# Define startup command
CMD ["python", "app.py"]
```

Containerization

Docker-compose file

```
version: '3.8'

services:
  web:
    build: .
    ports:
      - "5000:5000"
    depends_on:
      - db
    environment:
      - DB_HOST=db
      - DB_USER=root
      - DB_PASSWORD=my-secret-pw
      - DB_NAME=my_database

  db:
    image: mysql:5.7
    environment:
      - MYSQL_ROOT_PASSWORD=my-secret-pw
      - MYSQL_DATABASE=my_database
    volumes:
      - db-data:/var/lib/mysql

volumes:
  db-data:
```

Containerization

Flask Configuration Update

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://username:password@localhost/my_database'

db = SQLAlchemy(app)

# ...rest of the Flask app code
```

Containerization

Add environment variables

```
import os
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

# Set up MySQL connection using environment variables
app.config['SQLALCHEMY_DATABASE_URI'] = f"mysql://{os.environ['DB_USER']}:{os.environ['DB_PASSWORD']}@{os.environ['DB_HOST']}/{os.environ['DB_NAME']}"

db = SQLAlchemy(app)

# ...rest of the Flask app code
```

Containerization

Run the tests

```
http://localhost:5000
```

Containerization

Question:

Suppose a containerized application faces performance issues. What steps would you take to troubleshoot and optimize the application performance?

Containerization

Resource Contention

Cause

- Over Utilization
- Noisy Neighbors
- Spike in the Workload
- Resource Leaks

Containerization

Resource Contention

Impact

- Performance Degradation
- Instability
- Contention Cascades

Containerization

Resource Contention

Measures

- Resource Limitation
- Resource Requests
- Horizontal Scaling
- Auto-Scaling
- Isolation and Prioritization
- Continuous Monitoring

Containerization

Storage I/O Bottlenecks

Cause

- Shared Storage
- Inefficient Access Patterns
- I/O Intensive Workloads
- Slow Storage Medium

Containerization

Storage I/O Bottlenecks

Impact

- Slow Application Response
- Increase Latency
- Degraded Throughput

Containerization

Storage I/O Bottlenecks

Measures

- Use Faster Storage Solutions
- Optimize the Access Patterns
- Distributed I/O
- Resource Isolation and Prioritization

Containerization

Overhead from Abstraction Layer

Cause

- Virtualization or Containerization Layers
- Resource Management
- Control Plane Operations
- Security and Isolation

Containerization

Overhead from Abstraction Layer

Impact

- Reduced Performance
- Resource Fragmentation
- Increased Latency

Containerization

Overhead from Abstraction Layer

Measures

- Optimize Container Runtimes
- Tune Container Configuration
- Right sizing and Resource Allocation
- Container Image Optimization

Containerization

Question:

A production docker container crashes frequently. What approach would you take to fix it?

Containerization

Logging and Error Analysis

- Container Logs

  ```
  docker logs <container-id>
  ```

- Docker Events

  ```
  docker events --since <container-id>
  ```

Containerization

Resource Utilization Check

- Resource Monitoring

```
docker stats <container-id>
```

Containerization

Container Configuration Inspection

- Docker Inspect

```
docker inspect <container-id>
```

Containerization

Health Checks and Probes

- Health Check examination

```
docker inspect <container-id>
```

Containerization

Container File System Analysis

- File System examination

```
docker exec –it <container-id> bash
```

Containerization

Memory and Resource Evaluation

- **Resource Adjustments**

```
docker update –memory <container-id>
```

Containerization

Replication and Testing

Documentation and Community Research

Logging and Documentation