week 4

Docker Topics for Research

What is Docker and Why Is It Used?

Overview:
Docker is a platform that automates the deployment of applications inside lightweight, portable containers. Containers package an application with its dependencies, libraries, and runtime, ensuring consistency across different environments.

Key Benefits:

- **Portability:** Containers run consistently on any system with Docker installed.
- **Isolation:** Applications run in isolated environments, reducing conflicts.
- **Scalability:** Enables rapid scaling in cloud-native architectures.
- **Efficiency:** Containers share the host OS kernel, using fewer resources than traditional virtual machines.

Docker Images and Containers

Docker Images:

- **Definition:** Immutable, read-only templates used to create containers.
- **Layers:** Built using a series of layered file systems (each command in a Dockerfile creates a new layer).
- **Usage:** Images can be stored and shared via registries.

Docker Containers:

- **Definition:** Running instances of Docker images.
- **Isolation:** Each container runs independently and is isolated from other containers.
- **Lifecycle:** Containers can be started, stopped, moved, and deleted without impacting the underlying image.

Dockerfile and How to Create It

Dockerfile:

- **Purpose:** A text file that contains a series of instructions to build a Docker image.

- **Basic Commands:**
  - `FROM` : Specifies the base image.
  - `RUN` : Executes commands inside the container during build time.
  - `COPY` / `ADD` : Copies files/directories into the container.
  - `CMD` : Sets the default command to run when the container starts.
  - `EXPOSE` : Declares the port the container listens on.

Example Dockerfile:

```
# Use an official Python runtime as a base image
FROM python:3.9-slim

# Set the working directory
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

Docker Networking Basics

Concepts:

- **Bridge Networks:** Default network for containers on a single host.
- **Host Networks:** Containers share the host's networking namespace.
- **Overlay Networks:** Enable communication across multiple Docker hosts (commonly used in Docker Swarm).
- **Port Mapping:** Expose container ports to the host machine using `-p` flag (e.g., `docker run -p 8080:80 image_name`).

Docker Volumes

Purpose:
Volumes provide persistent storage for containers. Unlike container filesystems, data stored in volumes is not removed when the container is deleted.

Usage:

- **Creating Volumes:** `docker volume create my_volume`
- **Mounting Volumes:** Use the `-v` flag (e.g., `docker run -v my_volume:/data image_name`).

Docker Compose

Overview:
Docker Compose is a tool for defining and running multi-container Docker applications. It uses a YAML file (`docker-compose.yml`) to configure services, networks, and volumes.

Key Features:

- **Service Definition:** Define multiple services in one file.
- **Dependency Management:** Specify service dependencies for proper startup order.
- **Scaling:** Easily scale services using the `docker-compose up --scale` option.

Example `docker-compose.yml` :

```yaml
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
  redis:
    image: "redis:alpine"
```

Docker Registry

Definition:
A Docker registry is a repository for Docker images. The most well-known public registry is Docker Hub.

Usage:

- **Pushing an Image:** `docker push username/image_name`
- **Pulling an Image:** `docker pull image_name`

- **Private Registries:** Organizations may host private registries for proprietary images.

Docker Swarm

Overview:
Docker Swarm is Docker’s native clustering and orchestration solution. It enables the management of a cluster of Docker nodes as a single virtual system.

Features:

- **Service Deployment:** Distributes containers across multiple nodes.
- **Scaling:** Easily scale services up or down.
- **Load Balancing:** Built-in load balancing among containers.

Basic Commands:

- **Initialize Swarm:** `docker swarm init`
- **Deploy a Service:** `docker service create --replicas 3 -p 80:80 image_name`

Docker Commands

Common Docker Commands:

- **Image Management:** `docker build` , `docker pull` , `docker push`
- **Container Management:** `docker run` , `docker stop` , `docker rm` , `docker logs`
- **System Information:** `docker info` , `docker ps`

These commands form the daily toolbox for managing Dockerized applications.

Docker vs. Virtual Machines

Comparison:

- **Resource Efficiency:** Containers share the host OS kernel, making them lighter than VMs which require full OS installations.
- **Isolation:** VMs provide stronger isolation, but containers offer sufficient isolation for most microservices.
- **Portability:** Both technologies support portability, but containers tend to be more portable due to their smaller size.
- **Use Cases:** VMs are often used for full operating system environments, whereas Docker is ideal for microservices and agile deployments.

Kubernetes (K8s) Topics for Research

What is Kubernetes and Why Is It Used?

Overview:
Kubernetes is an open-source container orchestration platform designed to automate deployment, scaling, and management of containerized applications. It addresses the challenges of running containers in production across clusters of hosts.

Key Benefits:

- **Scalability:** Automatically scales applications based on demand.
- **Self-Healing:** Restarts failed containers and reschedules them as needed.
- **Declarative Configuration:** Uses YAML or JSON files to define the desired state.
- **Extensibility:** Supports custom resources and controllers for advanced use cases.

Kubernetes Pods and Deployments

Pods:

- **Definition:** The smallest deployable units in Kubernetes, representing one or more containers that share the same network namespace and storage.
- **Lifecycle:** Pods are ephemeral and can be replaced by new instances.

Deployments:

- **Definition:** A higher-level abstraction that manages pod replicas and ensures the desired number of pods are running.
- **Features:** Supports rolling updates and rollbacks, ensuring continuous delivery.

Services in Kubernetes

Definition:
Services provide stable networking and load balancing to pods. They decouple application endpoints from pod lifecycles.

Types of Services:

- **ClusterIP:** Exposes the service on an internal IP in the cluster.
- **NodePort:** Exposes the service on a static port on each node.
- **LoadBalancer:** Creates an external load balancer in supported cloud environments.
- **Headless Services:** Directly expose pod IPs for use cases like stateful applications.

ConfigMaps and Secrets

ConfigMaps:

- **Purpose:** Store non-sensitive configuration data in key-value pairs.
- **Usage:** Inject configurations into pods without baking them into container images.

Secrets:

- **Purpose:** Store sensitive data (e.g., passwords, tokens) securely.
- **Usage:** Secrets are base64-encoded and managed separately to maintain security best practices.

Namespaces in Kubernetes

Definition:
Namespaces provide a mechanism for isolating groups of resources within a single Kubernetes cluster. They help organize and manage resources in multi-team or multi-project environments.

Usage:

- **Separation:** Different teams or projects can operate in separate namespaces.
- **Resource Quotas:** Enforce limits on resource usage per namespace.

Kubernetes Volumes

Overview:
Volumes in Kubernetes provide persistent storage for pods. Unlike container storage, Kubernetes volumes are independent of pod lifecycles.

Types:

- **emptyDir:** Temporary storage that is erased when a pod is removed.
- **PersistentVolume (PV) and PersistentVolumeClaim (PVC):** Provide dynamic or static provisioning of storage that persists beyond pod restarts.

Autoscaling in Kubernetes

Horizontal Pod Autoscaler (HPA):

- **Definition:** Automatically scales the number of pod replicas based on observed CPU utilization or custom metrics.
- **Benefits:** Optimizes resource usage and maintains application performance during variable

loads.

Vertical Pod Autoscaler (VPA):

- **Definition:** Adjusts the resource limits (CPU and memory) of pods based on their usage patterns.
- **Use Case:** Ideal for workloads with fluctuating resource demands.

Role-Based Access Control (RBAC)

Overview:
RBAC in Kubernetes restricts access to cluster resources based on the roles assigned to users or service accounts. This enhances security and ensures only authorized users can perform specific actions.

Components:

- **Roles/ClusterRoles:** Define permissions at the namespace or cluster level.
- **RoleBindings/ClusterRoleBindings:** Associate roles with users, groups, or service accounts.

Helm Charts (Package Management)

Helm:
Helm is the package manager for Kubernetes, simplifying the deployment of complex applications.

Helm Charts:

- **Definition:** Pre-configured templates that define Kubernetes resources (e.g., deployments, services).
- **Benefits:** Streamline application deployment, versioning, and management.

Usage:

- **Installation:** `helm install my-release chart_name`
- **Upgrading:** `helm upgrade my-release chart_name`
- **Rollback:** Easily revert to a previous release version if needed.

Basic Kubernetes Commands

Key kubectl Commands:

- **Cluster Information:**

```
kubectl cluster-info
```

- **Managing Pods:**

```
kubectl get pods
kubectl describe pod <pod-name>
kubectl logs <pod-name>
```

- **Deployments and Services:**

```
kubectl get deployments
kubectl apply -f deployment.yaml
kubectl expose deployment <deployment-name> --port=80 --target-port=8080
```

- **Namespaces and Resources:**

```
kubectl get namespaces
kubectl config set-context --current --namespace=<namespace>
```