Docker and Kubernetes Detailed Guide

Step 1: Learn Docker in Detail

1.1. What is Docker?

Docker is a platform that allows you to develop, ship, and run applications inside containers. These containers encapsulate everything the application needs to run, ensuring it works seamlessly across different environments, such as development, testing, and production.
- Container: A lightweight, standalone, executable package that contains everything to run the application—code, runtime, system tools, libraries, and settings.
- Docker Image: A read-only template used to create a Docker container. It includes the application code and everything needed to run it.
- Docker Container: A running instance of a Docker image. It's a lightweight and isolated process.

1.2. Install Docker

Windows/macOS: You can download and install Docker Desktop from Docker's official site.
Linux: Install Docker using your distribution's package manager.

Check installation:

```
$ docker --version
```

1.3. Docker Images and Containers

Creating an Image: To build a Docker image from your application, you need to write a Dockerfile. Here's a breakdown of key Dockerfile commands:

- FROM: Specifies the base image.
- COPY: Copies files from your local directory to the container.
- RUN: Executes commands in the container (e.g., install dependencies).
- CMD: Specifies the command to run when the container starts.

Example Dockerfile for a Node.js application:

```
FROM node:14
WORKDIR /app
COPY . .
RUN npm install
CMD ['npm', 'start']
```

Building an Image:

```
$ docker build -t my-node-app .
```

Running a Container:

```
$ docker run -p 3000:3000 my-node-app
```

1.4. Docker Containers: Management Commands

List Containers: Shows all running containers.

```
$ docker ps
```

Stop a Container: Stop a running container.

```
$ docker stop <container_id>
```

Remove a Container: Delete a container.

```
$ docker rm <container_id>
```

Remove an Image:

```
$ docker rmi <image_name>
```

1.5. Docker Compose

Docker Compose is a tool that allows you to define and manage multi-container applications. It uses a docker-compose.yml file to specify services, networks, and volumes.

Example docker-compose.yml for a multi-container application (Node.js and PostgreSQL):

```
version: '3'
services:
  web:
    build: .
    ports:
      - '3000:3000'
  db:
    image: postgres
    environment:
      POSTGRES_PASSWORD: example
```

Running with Docker Compose:

```
$ docker-compose up
```

Stopping Compose Services:

```
$ docker-compose down
```

1.6. Docker Volumes and Networks

Volumes: Used to persist data across container restarts.

```
$ docker volume create my-volume
$ docker run -v my-volume:/data my-container
```

Networking: Docker allows containers to communicate over networks.

1.7. Advanced Docker Topics

Multi-Stage Builds: Dockerfile optimization to reduce image size.
Security: Ensure secure Docker images by scanning for vulnerabilities, using minimal base images, and restricting container capabilities.

Step 2: Learn Kubernetes in Detail

2.1. What is Kubernetes?

Kubernetes (K8s) is an open-source platform designed to automate the deployment, scaling, and management of containerized applications. It orchestrates containers running across clusters of machines.

2.2. Kubernetes Cluster Architecture

- Master Node: The brain of the Kubernetes cluster, managing and scheduling containers.
- Worker Node: Machines that run the actual containers (pods).

Master Node Components: API Server, Scheduler, Controller Manager, etcd.
Worker Node Components: Kubelet, Kube Proxy.

2.3. Kubernetes Concepts and Resources

- Pod: The smallest deployable unit in Kubernetes, which can host one or more containers.

- Deployment: A higher-level abstraction that manages stateless applications.

- Service: A logical abstraction that defines a set of pods and a policy by which to access them.

- ReplicaSet: Ensures the desired number of pod replicas are running.

- StatefulSet: Manages stateful applications.

- Namespace: Provides a way to divide resources across a cluster.

2.4. Kubernetes Setup

Use tools like Minikube for local development or set up Kubernetes on a cloud provider (e.g., Google Kubernetes Engine, Azure Kubernetes Service).

Install kubectl:

```
$ kubectl version
```

2.5. Basic Kubernetes Commands

- Get Cluster Information:

```
$ kubectl cluster-info
```

- Get Pods in a Namespace:

```
$ kubectl get pods
```

- View Pod Logs:

```
$ kubectl logs <pod_name>
```

- Create Resources:

```
$ kubectl apply -f <yaml_file>
```

- Delete Resources:

```
$ kubectl delete -f <yaml_file>
```

- Describe Resources:

```
$ kubectl describe pod <pod_name>
```

2.6. Kubernetes Deployments

Create a Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
```

```
    - name: node-app
      image: node:14
      ports:
        - containerPort: 3000
```

Apply the Deployment:

```
$ kubectl apply -f deployment.yaml
```

2.7. Services in Kubernetes

ClusterIP: Exposes the service inside the cluster.

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  selector:
    app: myapp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  type: ClusterIP
```

NodePort & LoadBalancer: Expose the service outside the cluster.

2.8. Persistent Storage

PersistentVolume (PV): Represents a piece of storage in the cluster.
PersistentVolumeClaim (PVC): A request for storage by a user.
Example PVC:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myapp-pvc
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
    - ReadWriteOnce
```

2.9. Helm: Kubernetes Package Manager

Helm simplifies Kubernetes deployments by managing complex applications using reusable charts.

Install Helm:

```
$ brew install helm
```

Install a Chart:

```
$ helm install myapp bitnami/nginx
```

2.10. Scaling Applications

Horizontal Pod Autoscaler (HPA): Automatically scales the number of pods based on CPU or memory usage.

Example HPA:

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: myapp-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: AverageUtilization
          averageUtilization: 80
```