09-containers-docker-kubernetes-intro-notes.md

Containers, Docker, Microservices, and Kubernetes (Intro Notes)

Source

These notes are formally organized from your rough session notes, keeping the same core content and sequence with slight detail additions.

Session 1: Why Containerization for Microservices?

1) Microservices Context

- A modern application is often split into **microservices**.
- Each service is usually:
    - lightweight,
    - focused on one business concern,
    - independently deployable,
    - easy to update and release continuously.

2) Application Runtime and Portability

- Traditional application setup:
    - App -> Application Server/Web Server -> Language Runtime.
- Key goals:
    - portability,
    - quick deployment,
    - continuous deployment.

3) Scaling Problem Under Heavy Load

When request volume increases, scaling becomes a challenge.

- **Horizontal scaling**: add more servers/instances.
- **Vertical scaling**: increase CPU/RAM/disk of the same machine.

- **Vertical scaling:** increase CPU/RAM/disk of the same machine.

For microservices, horizontal scaling is generally more practical.

4) VM-Based Deployment Limitations

In VM-based deployment, each service may run with a full OS image.

Typical issues noted:

- high upfront allocation (example: 4 GB RAM, 30 GB disk, 2 vCPU),
- OS overhead consumes resources,
- unnecessary OS processes keep running,
- under-utilization of provisioned CPU/RAM/disk,
- at large microservice count, VM count becomes very high.

5) Why Containerization Helps

Containerization packages only what is necessary:

- application,
- runtime,
- essential OS binaries/libraries.

This makes deployment more efficient than full VM-per-service patterns.

Session 2: Core Container Concepts and Docker Basics

1) Packaging Analogy

- Java project -> compiled `.class` files -> packaged as `.jar`.
- Multiple files -> archive (like `.zip`).
- Similarly, containerization packages app + runtime + dependencies in one deployable unit.

2) Container Stack (Conceptual)

- Without containers (simplified):
  - App + Runtime + OS + VM + Physical machine.
- With containers:
  - App + Runtime + essential OS components -> **Container Image**.

Examples from notes:

- Java service image: `jar + JRE + essential binaries`.
- Python service image: `python code + python runtime + essential binaries`.
- Website image: `HTML + Apache/Nginx + essential binaries`.

3) Key Definitions

- **Container Image**: immutable package of app + dependencies + required binaries.
- **Container**: running instance of a container image.
- **Container Engine/Daemon**: software that runs containers (example: Docker).
- **Container Registry**: central repository for images.

Registry examples:

- Docker Hub (default public registry in many setups),
- Azure Container Registry,
- Amazon Elastic Container Registry,
- Google Container Registry / Artifact Registry.

4) Execution Model

Concept flow from notes:

- Physical machine -> VM -> Container engine -> many container instances.

Docker can run multiple copies of same/different images on the same host.

5) Different Container Runtime Behaviors

From your notes, containers can be:

1. run and stop (non-networked batch style),
2. interactive CLI-style and stop after use,
3. background continuous non-networked worker,
4. background timed/finite job that exits,
5. continuous network service (has connectivity via IP/port mapping).

Examples listed:

- MySQL database container,

- Ubuntu bash/terminal container,
- Python interpreter container,
- periodic backup service,
- Apache web server container.

6) Container Features (Important)

1. Container ID.
2. Created from an image.
3. Has an entry command/process (`CMD` / `ENTRYPOINT` behavior matters).
4. Lifecycle state: running or exited.
5. Container writable layer/lifecycle data behavior is distinct from image.
6. May or may not expose network access (depends on workload type).
7. Has a name (you can refer using name or ID).

Session 3: Docker Commands, Dockerfile, and Kubernetes Intro

1) Docker Service and Permissions

Commands noted:

```
sudo service docker status
sudo service docker start
```

If permission issues occur for Docker socket:

```
sudo chown username:username /var/run/docker.sock
```

2) Common Docker Commands (from notes)

```
docker login
docker pull
docker run -it
docker commit
docker push
docker ps -a
docker ps -aq
docker images
docker images -aq
docker rm <containerid>
docker rm $(docker ps -aq)
docker rmi $(docker images -aq)
```

Also mentioned:

- `docker network`,
- bridge gateway example like `172.17.0.1`,
- port mapping basics,
- difference between `docker run`, `docker start`, and `docker exec`.

Quick distinction:

- `docker run`: create + start a new container.
- `docker start`: start an existing stopped container.
- `docker exec`: run a command inside an already running container.

3) Dockerfile Notes

`RUN` vs `CMD` vs `ENTRYPOINT`

- `RUN`: executes during image build.
- `CMD`: default runtime command; can be overridden at `docker run` time.
- `ENTRYPOINT`: main executable; generally treated as fixed startup command.

4) Environment Variables

- Runtime env vars can be passed at `docker run`.
- Default env vars can be defined in Dockerfile via `ENV` so image has base defaults.

Example from notes (formatted):

```
FROM node:20
WORKDIR /app
COPY . /app
RUN npm ci

ENV DBHOST=myhost
ENV DBPASS=password
ENV DBUSER=username
ENV DBPORT=3306
ENV DBPORT=3306

EXPOSE 3000
ENTRYPOINT ["node", "index.js"]
```

Run example from notes:

```
docker run -d -p 8086:3000 nodeapp
```

5) Microservices + Containers -> Need for Orchestration

As container count grows across many VMs/nodes, we must manage:

- identity,
- configuration,
- networking,
- storage,
- security,
- lifecycle,
- connectivity.

This is **container orchestration**.

6) Kubernetes Intro

- Kubernetes is a container orchestration platform.
- It manages containers across a **cluster of nodes (VMs/machines)**.
- It helps standardize deployment and operations for large-scale microservices.

Quick Learning Path (from notes)

- Linux fundamentals -> Docker -> Kubernetes.

Topics You Marked for Deeper Follow-Up

- Docker networking scenarios,
- storage management in Docker,
- Docker Compose,

- config/secrets handling (e.g., ConfigMap concept in Kubernetes),
- building and publishing custom images end-to-end.

Reference from Notes

- https://github.com/vinodh1988/docker-course