Docker & Kubernetes Masterclass - Typed Notes

1. The Software Development Life Cycle (SDLC)

- Software development is a journey with different stages: planning, development, testing, deployment, and monitoring.
- Today's focus: Development (writing code) and Deployment (making it accessible to users).
- Challenges exist in both stages.

2. Challenges in a "Dockerless" World

Development Challenges:

- Inconsistent Environments: Developers use different operating systems, leading to compatibility issues.
- Dependency Hell: Conflicting software versions cause unexpected errors.
- Example: A Java application relying on a specific version of OpenSSL might not work correctly on different systems.

Deployment Challenges:

- Manual Configuration: Setting up servers and deploying code is timeconsuming and error-prone.
- Scalability Issues: Handling increased traffic can be complex and expensive.
- Downtime during Updates: Deploying new versions can disrupt service for users.

3. Brainstorming Solutions

- How can we ensure consistency and avoid dependency conflicts?
- (Discuss potential solutions, including virtualization, and their limitations)

4. The Java Virtual Machine (JVM) - A Partial Solution

Java addresses some challenges with the JVM (Java Virtual Machine).

- WORA (Write Once, Run Anywhere): Java programs can run on any platform with a JVM (thanks to James Gosling, the father of Java!).
- **Limitations:** JVM needs configuration, and it doesn't solve problems for non-Java applications or complex dependencies.

5. Introducing Docker

- Docker: A platform that packages applications and dependencies into containers.
- Analogy: Like a shipping container, it carries everything the application needs to run anywhere.

Benefits:

- **Consistency:** Application runs the same way everywhere.
- Efficiency: Lightweight and uses fewer resources.
- Isolation: Prevents conflicts and improves security.

6. Unpacking Docker - Key Concepts

- **Docker Image:** A snapshot of your application and its environment.
 - Benefits: Speed, consistency, preservation of dependencies, versioning.
- **Docker Container:** A running instance of an image.
- **Dockerfile:** Instructions for building an image.

7. The Docker Workflow

- 1. Write a Dockerfile.
- 2. Docker builds an image.
- 3. Run the image to create a container.

8. Hands-on with Docker

- Basic Commands: docker run , docker ps , docker images , docker stop
- Dockerfiles:
 - FROM instruction: Specifies the base image, avoiding starting from scratch. (Example: FROM ubuntu:latest)
 - COPY, RUN, CMD instructions (explained with examples)

- **Example:** Create a simple Python application and Dockerfile, build and run.
- Open-source Dockerfile example: Show a real-world Dockerfile and highlight the FROM instruction.

9. The Need for Orchestration

- Challenges of managing multiple microservices: Maintaining replicas, scaling, networking.
- Orchestration: Automating the management and coordination of services.
 (Analogy: conductor in an orchestra)
- Tools: Docker Compose and Kubernetes.

10. Docker Compose - Orchestration for Development

- Solves: Inconsistent development environments across different operating systems.
- Primarily for local development: Limited scalability, simpler networking, single-host focus.
- Hands-on:
 - Example: Web application with a Python web server and PostgreSQL database.
 - o docker-compose.yml: Show and explain the configuration file.
 - Run: docker-compose up -d

11. The Microservices Revolution

- **Monolithic Architecture:** A single, large application with all components tightly coupled.
 - Challenges: Codebase explosion, scalability issues, risky deployments, technology lock-in.
- **Microservices:** Breaking down the application into small, independent services.
 - Characteristics: Loosely coupled, specific business capabilities, lightweight communication.
- Benefits: Improved agility, increased scalability, enhanced resilience, technology diversity.

- Netflix Example: 1000+ microservices, hundreds of deployments per day, DevOps excellence.
- **Docker and Microservices:** A perfect match for containerization, isolation, portability, and scalability.

12. Kubernetes - Orchestration for Production

- **Kubernetes:** The most popular container orchestrator for production.
- Cluster: A set of machines that run containerized applications.
- Pod: The smallest deployable unit in Kubernetes (can contain one or more containers).
- Services: Enable communication between pods.
- Hands-on:
 - Port forwarding: kubectl port-forward
 - Show the application: Access through the forwarded port.
- A Glimpse of Production: Live application with a domain name and SSL. -<u>https//sample.getdevops.services</u>
- Wrap-up: Recap, next steps, encourage further exploration.

Key Takeaways:

- Docker simplifies building, sharing, and running applications in containers.
- Docker Compose helps manage multi-container applications locally.
- Microservices offer advantages in agility, scalability, and resilience.
- Kubernetes orchestrates containers in production environments.