Docker & Containerization

Comprehensive Student Study Plan

Student Name: Faith Gabrielle Gamboa
Section: IV-BCSAD
Title: Assignment 3
Date: October 18, 2025

COURSE OVERVIEW

This comprehensive study plan is designed to take you from Docker basics to advanced containerization concepts. By the end of this course, you will be able to build, deploy, and manage containerized applications in production environments.

LEARNING OBJECTIVES

- Understand the fundamentals of containerization and its benefits over traditional virtualization
- Install, configure, and use Docker on various operating systems
- Create, manage, and optimize Docker images and containers
- Write efficient Dockerfiles following industry best practices
- Implement Docker networking and storage solutions
- Orchestrate multi-container applications using Docker Compose
- Apply security best practices for container deployments
- Deploy containerized applications to production environments

STUDY PLAN

WEEK 1: Introduction to Containerization

Topics

- What is containerization and why it matters in modern software development
- Containers vs Virtual Machines: understanding the key differences and use cases
- Docker architecture and core components (Docker Engine, Docker CLI, Docker Daemon, Docker Desktop)
- Understanding images, containers, and registries
- Installing Docker on Windows, macOS, and Linux systems
- Basic container lifecycle: create, start, stop, and remove

Activities

- Install Docker Desktop on your operating system
- Run your first container using the hello-world image
- Pull and run basic images: nginx, ubuntu, alpine
- Explore Docker Hub and browse official images
- Practice basic Docker CLI commands: docker version, docker info, docker ps, docker images, docker run
- Create a Docker Hub account and explore popular repositories

Resources

- Docker Official Documentation: Get Started Guide (docs.docker.com/get-started)
- Docker Overview and Installation Guide
- Video: "Docker Tutorial for Beginners" by TechWorld with Nana
- Play with Docker (labs.play-with-docker.com) - free online Docker playground
- Article: "What is Docker and Why Should You Care?"

Expected Outcomes

- Understand the fundamental concepts of containerization and its advantages
- Successfully install and configure Docker on your system
- Run basic containers and understand the container lifecycle
- Navigate Docker Hub and pull images from registries
- Execute fundamental Docker CLI commands confidently

WEEK 2: Working with Docker Images and Containers

Topics

- Understanding Docker images and their layered architecture
- Essential Docker commands: run, exec, logs, inspect, ps, top, stats
- Container lifecycle: create, start, stop, restart, pause, remove
- Writing Dockerfiles: basic instructions (FROM, RUN, COPY, CMD, EXPOSE)
- Building custom images using docker build
- Tagging and pushing images to Docker Hub
- Environment variables and port mapping for containers

Activities

- Run containers in both detached and interactive modes
- Access containers using docker exec with bash/sh shells
- Build your first custom image using a simple Node.js or Python web app
- Map container ports to host ports and access web applications
- Tag and push your custom image to Docker Hub
- View and analyze container logs for debugging

Resources

- Docker Command Line Reference Documentation
- Dockerfile Reference Guide
- Video: “Writing Production-Ready Dockerfiles”
- Article: “Docker Images and Containers Explained”
- Practice: Docker Labs – Container Operations

Expected Outcomes

- Master essential Docker CLI commands for daily operations
- Understand how Docker images are structured and layered
- Create and manage custom images using Dockerfiles
- Configure containers with environment variables and port mappings
- Push images to Docker Hub for sharing or deployment

WEEK 3: Docker Networking and Data Management

Topics

- Docker networking fundamentals and concepts
- Network drivers: bridge, host, none, overlay (intro)
- Default vs user-defined bridge networks

- Container-to-container communication and DNS resolution
- Understanding data persistence and mount types: volumes, bind mounts
- Creating and managing Docker volumes
- Sharing data between containers

Activities

- Create custom bridge networks using docker network create
- Connect multiple containers to the same network
- Test inter-container communication using container names
- Set up a web app + database architecture using Docker networks
- Create and mount volumes for data persistence
- Inspect network and volume configurations
- Clean up unused networks and volumes

Resources

- Docker Networking Documentation
- Docker Storage Documentation
- Video: "Docker Networking Tutorial"
- Article: "Managing Data in Docker Containers"
- Practice Lab: Multi-Container Networking Scenarios

Expected Outcomes

- Understand Docker network drivers and their purposes
- Create and configure custom networks for applications
- Enable communication between containers
- Implement persistent data storage using volumes
- Troubleshoot networking and data management issues

WEEK 3: Docker Compose and Best Practices

Topics

- Introduction to Docker Compose and its benefits
- Writing docker-compose.yml files with proper YAML syntax
- Defining services, networks, and volumes in Compose
- Using .env files for environment configuration
- Service dependencies and startup order
- Scaling services horizontally with Compose
- Basic container security practices: non-root users, resource limits, image scanning

- Best practices for efficient and secure Docker usage

Activities

- Install Docker Compose (if not included in Docker Desktop)
- Create a docker-compose.yml file for a multi-service app (web + database)
- Define service dependencies using depends_on
- Use .env files for service configuration
- Scale services using docker-compose up --scale
- Implement non-root users in Dockerfiles
- Scan images for vulnerabilities using Docker Scout or Trivy

Resources

- Docker Compose Documentation
- Compose File Reference (version 3)
- Docker Security Documentation
- Video: "Docker Compose Tutorial for Beginners"
- Article: "Docker Security Best Practices"

Expected Outcomes

- Write and execute well-structured Docker Compose files
- Orchestrate multi-container applications with ease
- Manage and scale services using Compose commands
- Apply basic security and resource management principles
- Build secure, efficient, and portable containerized applications

HISTORY OF DOCKER AND CONTAINERIZATION

Understanding the evolution of containerization technology helps contextualize Docker's impact on modern software development and deployment practices.

ERA 1: Pre-Docker Foundations (1979-2012)

YEAR	TECHNOLOGY	DESCRIPTION & IMPACT
1979	chroot	Unix V7 introduced chroot system call, allowing the changing of root directory for a process and its children. This provided the earliest form of process isolation,

		enabling the creation of isolated file system environments.
2000	FreeBSD Jails	FreeBSD introduced jails, which allowed system administrators to partition a FreeBSD system into several independent mini-systems called "jails". This provided better security and easier system administration through isolation.
2004	Solaris Containers	Sun Microsystems released Solaris Containers (also known as Zones), combining system resource controls and boundary separation provided by zones. This offered complete runtime environments for applications with resource management capabilities.
2006	Process Containers	Google engineers introduced process containers (later renamed to Control Groups or cgroups) to limit, account for, and isolate resource usage of process collections. This technology was merged into the Linux kernel in 2007.
2008	LXC (Linux Containers)	LXC was released as the first complete implementation of Linux container manager. It combined cgroups and Linux namespaces to create isolated environments. However, it remained complex and primarily used by system administrators rather than developers.

ERA 2: The Docker Revolution (2013-2015)

YEAR	MILESTONE	DESCRIPTION & IMPACT
March 2013	Docker Launch	Solomon Hykes launched Docker as an open-source project at PyCon in Santa Clara, California. Docker was initially built on top of LXC but provided a much simpler, developer-friendly interface. The

		key innovation was making containers accessible and practical for application developers, not just system administrators.
2013	Docker Philosophy	Docker introduced revolutionary concepts: containerizing applications with all dependencies into a single portable unit, "Build once, run anywhere" philosophy, and the idea of immutable infrastructure. This solved the infamous "works on my machine" problem that plagued software development.
June 2014	Docker 1.0	Docker 1.0 was released, marking production readiness. The ecosystem began growing rapidly with Docker Hub (public container registry for sharing images), Docker Compose (for multi-container orchestration), and widespread enterprise adoption.
2014	Industry Adoption	Major technology companies including Google, Microsoft, Amazon, and Red Hat announced support for Docker. The container ecosystem exploded with tools, services, and platforms built around Docker technology.
2015	libcontainer & OCI	Docker moved from LXC to its own execution environment called libcontainer (later runc), giving Docker more control over the container runtime. The Open Container Initiative (OCI) was founded by Docker and industry leaders to create open standards around container formats and runtimes, ensuring interoperability.

ERA 3: Maturity and Ecosystem Growth (2016-Present)

YEAR	DEVELOPMENT	DESCRIPTION & IMPACT
2016	Cross-Platform Docker	Docker for Mac and Docker for Windows were released, making Docker truly accessible on all major operating systems with native-feeling applications. Docker Swarm was introduced as Docker's native clustering and orchestration solution.
2017	Kubernetes Rise	Kubernetes emerged as the dominant container orchestration platform, surpassing Docker Swarm in popularity. The "orchestration wars" concluded with Kubernetes becoming the de facto standard for container orchestration in enterprise environments.
2018	Docker Enterprise	Docker embraced Kubernetes by integrating it into Docker Desktop and Docker Enterprise Edition. This showed Docker's commitment to developer choice and ecosystem collaboration rather than competition.
2019	Docker Restructuring	Docker Inc. underwent significant restructuring, selling Docker Enterprise to Mirantis. The company refocused on the developer experience and the Docker Desktop product, emphasizing its role as the developer's gateway to containers.
2020-2021	Containerization Standard	Container technology became the standard for application deployment across industries. Docker Desktop evolved with improvements to performance, security scanning (Docker Scout), and development workflows. The COVID-19 pandemic accelerated cloud adoption and containerized deployments.
2022-2023	Modern Features	Introduction of Docker Extensions (allowing third-party tools to integrate

		with Docker Desktop), enhanced security features including built-in vulnerability scanning, improved support for ARM architectures (Apple Silicon), and better integration with cloud platforms.
2024-Present	AI/ML Integration	Docker continues evolving with focus on AI/ML workloads, enhanced developer productivity tools, improved build performance with BuildKit, and seamless cloud integration. Container technology is now fundamental to modern software development, DevOps, and cloud-native architectures.

BEST PRACTICES FOR IMPLEMENTING DOCKER

Following industry best practices ensures your containerized applications are secure, efficient, and maintainable in production environments.

1. DOCKERFILE BEST PRACTICES

- Use Official and Minimal Base Images
    - Start with trusted official images (e.g., node:18-alpine) or minimal ones like Alpine or Distroless for smaller, more secure images.

```
FROM node:18-alpine
# or ultra-minimal
FROM gcr.io/distroless/nodejs18-debian11
```

- Optimize Layer Caching
    - Order Dockerfile steps wisely — copy dependencies first, then app code. Combine RUN commands.

```
COPY package*.json ./
RUN npm ci --only=production
COPY . .
```

- Use Multi-Stage Builds
  - Separate build and runtime stages for smaller images.

```
FROM node:18 AS builder
WORKDIR /app
COPY . .
RUN npm ci && npm run build

FROM node:18-alpine
COPY --from=builder /app/dist ./dist
CMD ["node", "dist/index.js"]
```

- Add a .dockerignore File
  - Exclude unnecessary files like node_modules, .git, .env, *.log, etc. to speed up builds and protect sensitive data.

- Avoid Running as Root
  - Use a non-root user inside containers.

```
RUN addgroup -S app && adduser -S app -G app
USER app
```

2. CONTAINER MANAGEMENT BEST PRACTICES

- One Process per Container
  - Run one service per container for scalability and easy troubleshooting.

- Implement Health Checks
  - Automatically detect and restart unhealthy containers.

```
HEALTHCHECK CMD curl -f http://localhost:3000/health || exit 1
```

- Set Resource Limits
  - Prevent containers from overusing system resources.

```
docker run --memory="512m" --cpus="1.0" myapp
```

- Use Named Volumes for Persistence
    - Store data outside containers for durability.

```
docker volume create app-data
```

```
docker run -v app-data:/app/data myapp
```

3. SECURITY BEST PRACTICES

- Scan Images Regularly
    - Use tools like Docker Scout or Trivy to find vulnerabilities.

```
trivy image myapp:latest
```

- Never Store Secrets in Images
    - Don't hardcode credentials; use .env or secret management tools.

- Pin Image Versions
    - Avoid latest; use fixed tags for reproducible builds.

```
FROM node:18.17.0-alpine
```

- Keep Images Updated
    - Rebuild regularly with updated dependencies to patch vulnerabilities.

- Enable Docker Content Trust
    - Ensure image authenticity.

```
export DOCKER_CONTENT_TRUST=1
```

4. NETWORKING BEST PRACTICES

- Use Custom Bridge Networks
    - Allows containers to communicate by name instead of IP.

```
docker network create my-app-net
```

```
docker run --network my-app-net --name web nginx
```

- Minimize Exposed Ports
    - Expose only required ports to reduce attack surface.

- Segment Networks
    - Use separate networks for frontend, backend, and database layers for better isolation.

5. PRODUCT DEPLOYMENT BEST PRACTICES

- Use Orchestration Tools
    - Leverage Kubernetes, Swarm, or ECS for scaling, health checks, and high availability.

- Centralize Logging
    - Use log drivers or external logging systems for easier monitoring.

REFERENCES

Docker, Inc. (2024). *Docker documentation: Best practices for writing Dockerfiles*. Docker Docs. https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

Docker, Inc. (2024). *Docker storage overview*. Docker Docs. https://docs.docker.com/storage/

Docker, Inc. (2024). *Docker networking overview*. Docker Docs. https://docs.docker.com/network/

Docker, Inc. (2024). *Docker security best practices*. Docker Docs. https://docs.docker.com/security/

Docker, Inc. (2024). *Docker Compose overview*. Docker Docs. https://docs.docker.com/compose/

Docker, Inc. (2024). *Docker Scout: Scan images for vulnerabilities*. Docker Docs. https://docs.docker.com/scout/

CIS Center for Internet Security. (2023). *CIS Docker Benchmark v1.6.0*. Center for Internet Security. https://www.cisecurity.org/benchmark/docker

TechWorld with Nana. (2023). *Docker tutorial for beginners [Video]*. YouTube. https://www.youtube.com/watch?v=pTFZFxd4hOI

Anchore. (2023). *Container security best practices guide*. Anchore Blog. https://anchore.com/blog/container-security-best-practices/

Snyk. (2024). *Docker security cheat sheet*. Snyk Blog. https://snyk.io/learn/docker-security-best-practices/