**VILLENA, LANS CLARENCE S.**

**IV - ACSAD**

# Assignment# 3 - Study Dockers and Containerization

## Week 1 – Foundations & Setup

- Study the evolution of containerization: chroot, Namespaces, Control Groups (cgroups), and LXC (Linux Containers).
- Understand the fundamental difference between containers and virtual machines (VMs).
- Read about Docker's origin and the factors that led to its widespread popularity.
- Install Docker Engine or Docker Desktop and successfully run your first container (e.g., "Hello World").
- Master fundamental commands: docker run, docker ps, docker stop, docker rm, and docker images.

## Week 2 – Building & Managing Images

- Dive into Dockerfile syntax and understand how image layering works to enable caching.
- Study and apply image best practices: using minimal base images (like Alpine), employing multi-stage builds, and leveraging .dockerignore.
- Learn and implement the key Dockerfile instructions: FROM, RUN, CMD, ENTRYPOINT, COPY, ADD, WORKDIR, and EXPOSE.
- Build, tag, push, and pull images from a central repository like Docker Hub or a private registry.

## Week 3 – Networking & Storage

- Master Container Networking: Understand the difference between Bridge, Host, and None network drivers.
- Practice Port Mapping using the -p flag to expose container ports to the host machine.
- Learn how to enable inter-container communication on a shared user-defined network.

- Understand Data Persistence and master the difference between Volumes (named/anonymous) and Bind Mounts.
- Practice creating and mounting volumes to persist data (e.g., a database file).
- Use docker system prune and other commands to effectively clean up unused resources.

## Week 4 – Orchestration Fundamentals (Docker Compose)

- Be introduced to the concept of Container Orchestration and why it's necessary for multi-container apps.
- Learn the fundamentals of Docker Compose and its purpose in a development environment.
- Understand the structure of the docker-compose.yml file, focusing on services, networks, and volumes blocks.
- Build and deploy a multi-container application (e.g., a web app and a database) using docker compose up.
- Master basic Compose commands: docker compose up, docker compose down, docker compose build, and docker compose logs.

## Week 5 – Security & Production Readiness

- Dive into basic Container Security: The critical practice of running processes as a non-root user using the USER instruction.
- Research tools and methods for security scanning of Docker images (e.g., Trivy or Snyk).
- Understand the importance of Secret Management and why secrets should never be in the Dockerfile (e.g., introduction to Docker Secrets/Kubernetes Secrets).
- Implement Health Checks using the HEALTHCHECK instruction to determine application status, not just process status.
- Learn about Resource Constraints and how to limit CPU and memory usage for a container.

## Week 6 – Project & Next Steps

- Consolidate all knowledge by designing and building a small portfolio project (e.g., a simple microservice or a full-stack CRUD application).
- Run the complete project using a well-structured Docker Compose file.

- Practice essential Troubleshooting techniques, including analyzing container logs, checking network connectivity, and resolving volume permission errors.
- Identify the next logical step in the journey: research the fundamental concepts of Kubernetes (Pods, Deployments, Services) as the industry standard for production orchestration

## History and Best Practices in Implementing Docker and Containerization

- The history of containerization predates Docker by decades, rooted in Unix-based isolation techniques like chroot (1979) and later, full OS-level virtualization like FreeBSD Jails and Solaris Zones. However, the modern era truly began with Linux Containers (LXC) around 2008, which utilized core Linux kernel features—Namespaces for process isolation and Control Groups (cgroups) for resource limiting—to create lightweight, isolated environments. Docker, launched in 2013, did not invent the technology but made it accessible. By introducing the standardized Docker Image Format, the build standard known as the Dockerfile, and the distribution platform Docker Hub, Docker turned a complex kernel technology into an intuitive developer tool. This rise in popularity quickly necessitated standardization, leading to the formation of the Open Container Initiative (OCI) in 2015 to ensure portability. Finally, the complexity of managing thousands of containers in production led to the dominance of orchestration tools, with Kubernetes (K8s) emerging as the industry standard.
- Effective implementation relies on several key best practices, starting with Image Creation. Developers should always use small, official base images (like Alpine) and employ multi-stage builds to ensure the final image is lightweight and contains only necessary runtime components, thereby reducing the attack surface. To enhance security, all processes should run as a non-root user via the USER instruction, and sensitive data must never be hardcoded into the image; dedicated secret management tools must be used instead. For production-ready containers, two runtime practices are critical: first, use Named Volumes or Bind Mounts for all persistent data (like databases or logs) to ensure data outlives the container; and second, define strict Resource Constraints (CPU and memory limits) to prevent any single container from destabilizing the host system. While Docker Compose is excellent for development, Kubernetes remains the essential tool for production, handling the complexities of scaling, self-healing, load balancing, and automated rollouts