## Docker Notes

### Monolithic vs. Microservices:

**Monolithic Architecture:**

* Entire application in a single code base.
* Enhancements require more time due to navigating the entire codebase.
* Developers need expertise in all modules.
* Compatibility issues may arise during module upgrades.
* Faster time-to-market but challenging maintenance.

**Microservices Architecture:**

* Application divided into independent services with individual codebases.
* Enhancements and updates are developed and deployed independently.
* Developers can specialize in individual microservices.
* Microservices can be updated without affecting the entire application.
* More complex but allows for better maintenance and scalability.

### Docker:

* Docker is a containerization platform that allows developers to create, deploy, and run applications in isolated and consistent environments known as containers.
* Containers package applications, their dependencies, and runtime, ensuring portability and consistency across various environments.

### Benefits of Docker:

* **Portability:** Docker containers are platform-agnostic and run consistently from development to production.
* **Efficiency:** Containers are lightweight and share the host OS kernel, reducing resource overhead.
* **Isolation:** Containers are isolated, preventing issues in one container from affecting others.
* **Scalability:** Docker facilitates easy scaling by adding or removing containers as needed.
* **Fast Deployment:** Containers can be quickly created, started, and stopped, making application deployment efficient.
* **Version Control:** Docker images support versioning and rollback, ensuring consistent behavior.
* **Ecosystem:** Docker offers a rich set of tools and resources for container management and orchestration.

### Pros of Docker:

* **Consistency:** Docker ensures consistent environments from development to production.
* **Resource Efficiency:** Containers use fewer resources compared to virtual machines.
* **Easy Scalability:** Docker allows for easy scaling of containers.
* **Fast Start-Up:** Containers launch rapidly, ideal for microservices architectures.
* **Isolation:** Containers are isolated and do not interfere with each other.
* **DevOps Integration:** Docker integrates well with DevOps practices, enabling continuous integration and continuous deployment (CI/CD).

### Cons of Docker:

* **Learning Curve:** Docker has a learning curve, especially for complex setups.
* **Security Concerns:** Inadequate container security practices can pose risks.
* **Overhead:** While less than virtual machines, containers still have some resource overhead.

### Basic Docker Architecture:

* **Docker Engine:** Comprising the Docker daemon (server) and Docker client (CLI), it runs containers.
* **Docker Images:** Read-only templates defining the application and its dependencies.
* **Docker Containers:** Runnable instances of Docker images.
* **Docker Hub:** A public registry for sharing Docker images, with the option for private registries.
* **Docker Compose:** A tool for defining and running multi-container applications using a YAML file.
* **Docker Swarm and Kubernetes:** Container orchestration tools for managing and scaling containers across clusters.

### Docker Commands and Examples:

**Image and Container Operations:**

1. **Pull Image:** docker pull <image\_name> - Download a Docker image from a registry.
   * Example: docker pull ubuntu:20.04
2. **List Images:** docker images - View the list of downloaded images.
   * Example: docker images
3. **Create Container:** docker create --name <container\_name> <image\_name> - Create a container from an image.
   * Example: docker create --name my\_container ubuntu:20.04
4. **Start Container:** docker start <container\_id> - Start a container.
   * Example: docker start my\_container
5. **Stop Container:** docker stop <container\_id> - Stop a running container.
   * Example: docker stop my\_container
6. **Remove Container:** docker rm <container\_id> - Remove a stopped container.
   * Example: docker rm my\_container
7. **List Running Containers:** docker ps - List currently running containers.
   * Example: docker ps
8. **List Containers:** docker ps -a - List all containers, including stopped ones.
   * Example: docker ps -a
9. **Run Container in Background:** docker run -d --name <container\_name> -p <host\_port>:<container\_port> <image\_name> - Create and start a container in the background.
   * Example: docker run -d --name web\_server -p 8080:80 nginx
10. **Execute Commands:** docker exec <container\_id> <command> - Run a command inside a running container.
    * Example: docker exec web\_server ls -l /usr/share/nginx/html
11. **Check Container Logs:** docker logs <container\_name> - View container logs.
    * Example: docker logs web\_server
12. **Interactive Shell:** docker exec -it <container\_id> /bin/bash - Access an interactive shell inside a container.
    * Example: docker exec -it web\_server /bin/bash
13. **Copy Files:** docker cp <file> <container\_id>:<destination\_path> - Copy files between your host and a container.
    * Example: docker cp index.html web\_server:/usr/share/nginx/html/

**Image Management:**

1. **Delete Image:** docker rmi <image\_name> - Remove a downloaded image.
   * Example: docker rmi nginx
2. **Inspect Container:** docker inspect <container\_id> - View detailed information about a container.
   * Example: docker inspect web\_server
3. **Create Custom Image:** docker commit <container\_id> <image\_name:tag> - Create a custom image from a container.
   * Example: docker commit web\_server my\_custom\_image:1.0
4. **Login to Docker Hub:** docker login - Authenticate with Docker Hub.
   * Example: docker login
5. **Push Image to Docker Hub:** docker push <image\_name:tag> - Upload a custom image to Docker Hub.
   * Example: docker push my\_custom\_image:1.0
6. **Build Image from Dockerfile:** docker build -t <image\_name:tag> <path\_to\_Dockerfile> - Build an image from a Dockerfile.
   * Example: docker build -t my\_docker\_image:1.0 .
7. **Create and Start a Container from Image:** docker run -d --name <container\_name> <image\_name:tag> - Create and start a container in the background from an image.
   * Example: docker run -d --name web\_app my\_docker\_image:1.0

**Docker Networking:**

1. **List Docker Networks:** docker network ls - List available Docker networks.
   * Example: docker network ls
2. **Create Custom Network:** docker network create <network\_name> - Create a custom network.
   * Example: docker network create my\_custom\_network
3. **Bridge Network:** Default network mode for container communication.
4. **Host Network:** Uses the host’s network stack; no port mapping required.
5. **None Network:** Isolates a container with no network access.

**Additional Docker Commands:**

1. **Export a Container:** docker export <container\_id> > <container\_filename>.tar - Export a container as a TAR file.
   * Example: docker export my\_container > my\_container.tar
2. **Clean Up System Resources:** docker system prune -a - Remove all unused data, including stopped containers, networks, and images.
   * Example: docker system prune -a
3. **Attach to a Running Container:** docker attach <container\_id> - Attach to a running container’s standard input, output, and error.
   * Example: docker attach my\_container
4. **Pause a Running Container:** docker pause <container\_id> - Pause a running container.
   * Example: docker pause my\_container
5. **Unpause a Paused Container:** docker unpause <container\_id> - Unpause a paused container.
   * Example: docker unpause my\_container
6. **Rename a Container:** docker rename <old\_name> <new\_name> - Rename a container.
   * Example: docker rename old\_container new\_container
7. **Port Mapping:** docker run -d -p <host\_port>:<container\_port> <image\_name> - Map host ports to container ports.
   * Example: docker run -d -p 8080:80 my\_web\_app
8. **View Container Resource Usage:** docker stats <container\_id> - View real-time resource usage of a running container.
   * Example: docker stats my\_container
9. **Docker Compose Up:** docker-compose up - Start services defined in a docker-compose.yml file.
   * Example: docker-compose up
10. **Docker Compose Down:** docker-compose down - Stop and remove services defined in a docker-compose.yml file.
    * Example: docker-compose down
11. **Docker Compose Build:** docker-compose build - Build services defined in a docker-compose.yml file.
    * Example: docker-compose build
12. **Docker Compose Logs:** docker-compose logs - View the logs of services defined in a docker-compose.yml file.
    * Example: docker-compose logs
13. **Docker Swarm Initialize:** docker swarm init - Initialize Docker Swarm on a manager node.
    * Example: docker swarm init
14. **Docker Swarm Join:** docker swarm join --token <token> <manager\_ip>:<port> - Join a worker node to a Docker Swarm.
    * Example: docker swarm join --token SWMTKN-1-abc123def456 192.168.1.100:2377
15. **List Docker Nodes:** docker node ls - List nodes in a Docker Swarm.
    * Example: docker node ls