# 

## 1. Monolithic vs Microservices Architecture

Software architecture defines how different parts of an application are organized and work together. Two main types are Monolithic and Microservices architectures.

### Monolithic Architecture

In a Monolithic structure, all parts of the app (UI, logic, database) are built together as one single program.

Characteristics:

• One single codebase for the whole app  
• All parts share the same resources  
• Scaling means deploying the entire app again

Advantages:

• Easy to build and deploy (good for small apps)  
• Easier debugging and testing  
• Less setup effort at the start

Disadvantages:

• Hard to scale only one part of the app  
• One small change may affect the whole app  
• Slower updates and less flexibility

Example: In a shopping app, product, payment, and user modules are all built together.

### Microservices Architecture

Here, the app is divided into smaller independent services. Each service handles one function and can be updated separately.

Characteristics:

• Each service has its own code and database  
• Communicates using APIs (like REST)  
• Can be deployed and scaled separately

Advantages:

• Easier to update or scale a specific service  
• Different services can use different technologies  
• Failure in one service doesn’t affect others

Disadvantages:

• Harder to set up and manage  
• Needs better monitoring and DevOps tools  
• Testing across services is more complex

Example: Shopping app with separate services for orders, payments, and inventory.

Key Differences:

• Monolithic = one big app | Microservices = many small apps  
• Monolithic scales as a whole | Microservices scale individually  
• Monolithic uses one tech stack | Microservices can mix technologies

## 2. Blue-Green Deployment

Blue-Green deployment helps release new software without downtime.

Concept: There are two same environments — Blue (live) and Green (testing). After the new version works fine on Green, users are shifted from Blue to Green.

Steps:

1. Create Blue (current) and Green (new) environments  
2. Deploy updates to Green  
3. Test the new version  
4. Redirect users to Green  
5. If any issue, switch back to Blue

Advantages: Zero downtime, easy rollback, safe testing  
Disadvantages: Costs more, database syncing is tough

## 3. Basic Docker Commands

Docker helps developers create containers — small, self-contained packages with everything needed to run an app.

Important Terms: Image, Container, Dockerfile, Docker Hub

Common Commands:

• docker --version → Check Docker version  
• docker pull <image> → Download image  
• docker images → List images  
• docker run <image> → Run a container  
• docker ps → Show running containers  
• docker stop <id> → Stop a container  
• docker start <id> → Restart a container  
• docker exec -it <id> bash → Open terminal inside container  
• docker build -t <tag> . → Build an image  
• docker-compose up → Run multiple containers

Example:  
1. docker pull nginx  
2. docker run -d -p 8080:80 nginx

## 4. Kubernetes Overview

Kubernetes (K8s) manages and automates container deployment and scaling.

Main Parts: Pod, Node, Cluster, Master Node, Kubelet, Service, Deployment

Key Features:  
• Auto-scheduling  
• Self-healing  
• Load balancing  
• Auto-scaling  
• Rolling updates  
• Resource management

Useful Commands:  
• kubectl get pods → List pods  
• kubectl apply -f deployment.yaml → Deploy an app  
• kubectl delete pod <name> → Remove a pod

Example: A company running many microservices can use Kubernetes to manage them easily.

## 5. Docker Implementation (Example)

Example: A simple Python Flask app using Docker.

Steps:  
1. Create Flask app (app.py)  
2. Write Dockerfile  
3. Build image → docker build -t flask-app .  
4. Run container → docker run -d -p 5000:5000 flask-app  
5. Open browser → http://localhost:5000

Explanation:  
• FROM = Base image  
• WORKDIR = Working folder  
• COPY = Copy files  
• RUN = Install packages  
• CMD = Run app