**Monolithic vs Microservices architecture**

**Monolithic Architecture:**  
Monolithic architecture is a software design in which the entire application is built as a single unit. All parts of the program such as the user interface, business logic, and database are connected together and work as one.

* It is mainly used for small or medium-sized applications where the system is not very complex and does not require frequent changes. Some common examples of monolithic systems are small web applications, school management systems, and simple banking or e-commerce platforms.
* The structure of a monolithic system includes one codebase, one database, and one deployment file. The main advantage of this architecture is that it is simple to develop, test, and deploy. However, it becomes difficult to manage and scale as the application grows. Also, if one part of the program fails, the entire application may stop working.
* **Microservices Architecture:**  
  Microservices architecture is a software design where the application is divided into many small and independent services. Each service performs a specific function and communicates with other services through APIs.
* This architecture is used for large and complex systems that require flexibility, scalability, and frequent updates. Examples of applications using microservices include Netflix, Amazon, Flipkart, and Uber.
* The structure of a microservices system consists of multiple independent services, each having its own code, database, and deployment. The main advantages are that it is easier to scale, maintain, and update individual services without affecting the whole system.
* If one service fails, the others can continue working. However, microservices are more complex to design and require good communication between services and more resources for management.
* **Difference between Monolithic and Microservices Architecture:**  
  Monolithic architecture is a single large unit where all components are connected, while microservices architecture is made up of many small independent services.
* Monolithic systems are easy to build at the beginning but hard to maintain and scale later, while microservices are flexible and scalable but need more setup and management.
* In a monolithic system, one error can stop the whole application, but in a microservices system, other services can still run even if one fails.
* Monolithic architecture is best for small applications, whereas microservices are suitable for large and complex systems.

**Blue-Green deployment environment**

Blue-Green Deployment is a software deployment strategy where two identical environments, called Blue and Green, are maintained. One environment, usually Blue, runs the current production version of the application, while the other, Green, is used to deploy the new version.

Users access the active environment, which is Blue, while the new version is deployed and tested in the inactive Green environment. Once testing is complete and everything works c+orrectly, traffic is switched from Blue to Green, making Green the active environment.

The Blue environment then becomes inactive and can be kept as a backup for rollback if needed. This approach reduces downtime because the switch between environments is almost instant, allows easy rollback if the new version has issues, and ensures safer deployment with minimal risk to users.

Blue-Green Deployment is commonly used in web applications that require zero downtime, cloud applications where updates should not affect users, and continuous deployment pipelines in DevOps.

The structure includes the Blue environment running the current production version, the Green environment hosting the new version for testing, and a load balancer or router that directs user traffic to either environment.

Overall, Blue-Green Deployment provides a smooth and safe way to release software updates, allowing quick switching between environments and easy rollback in case of problems.

**DOCKER**

Docker :- An open-source platform that enables developers to package applications along with all their dependencies into containers.

Containers: A container is a lightweight and executable package that contains everything needed to run an application.

Images: Containers are created from images, which are templates that include the application code and dependencies.

Docker Engine: The core of Docker, which creates and runs containers on the host operating system.

Docker Hub: A repository where developers can share and pull prebuilt container images.

Dockerfile**:**It is a file that describes the steps to create an image quickly.

A Docker volume is a storage mechanism used to store data outside the container’s writable layer, so the data is not lost when the container stops or is removed.

A Docker registry is a server that stores Docker images.  
It helps users share and manage container images easily.  
The most common example of a registry is Docker Hub.

The Docker client is a command-line tool that allows users to interact with Docker by sending commands.It communicates with the Docker daemon, which performs actions like building images and running containers.

**Docker Commands**

**1.docker –version :-** Checks which Docker version is installed.

**2. docker info:-** Shows detailed info about Docker, like number of containers, images, storage, and runtime.

**3. docker pull <image>:-**Downloads an image from Docker Hub.

**4. docker images :** Lists all Docker images on your system

**5. docker run <image> :** Creates and starts a container from an image.

**6. docker ps :** Shows all running containers.

**7. docker ps -a:** Shows all containers, running and stopped.

**8. docker stop <container\_id> :** Stops a running container.

**9. docker start <container\_id> :** Starts a stopped container.

**10. docker restart <container\_id> :** Stops and starts a container again.

**11. docker rm <container\_id> :** Deletes a stopped container.

**12. docker rmi <image> :** Deletes an image from the system.

**13. docker exec -it <container\_id> <command> :** Executes a command inside a running container.

**14. docker logs <container\_id>m :** Shows logs of a running container.

**16. docker build -t <image\_name> <path> :** Builds a new image from a Dockerfile.

**17. docker network ls :** Lists all Docker networks.

**18. docker volume ls :** Lists all Docker volumes.

**Kubernetes Overview**   
Kubernetes is a tool used to manage and control containers (like Docker containers). It helps in running applications that are made up of many containers by handling them automatically. It is often called K8s.  
When many containers are running together, it becomes difficult to start, stop, or restart them manually. Kubernetes makes this easy by doing it automatically. It checks if a container fails and starts a new one, balances the load, and ensures the application always runs smoothly.

Use / Purpose:

* To run and manage containers automatically.
* To start or stop containers when needed.
* To share traffic equally between containers.
* To update applications without stopping them.
* To save time and reduce human work in managing apps.

Structure / Parts:

1. Cluster: The whole Kubernetes system that manages containers.
2. Master Node: The brain of Kubernetes that controls everything.
3. Worker Nodes: The machines where the containers actually run.
4. Pod: The smallest part in Kubernetes that holds one or more containers.
5. Service: Helps different parts (pods) communicate with each other or with users.
6. Deployment: Helps create and update multiple pods automatically.

Advantages:

* Keeps applications running even if some containers fail.
* Can automatically add or remove containers when load changes.
* Makes updating software easier without downtime.
* Works on local systems and cloud platforms.
* Increases reliability and saves time.