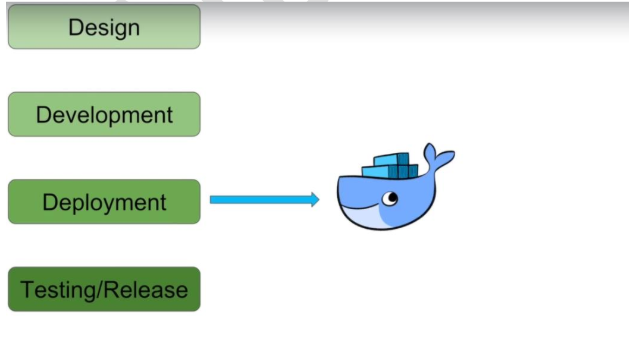
 **DOCKER TUTORIAL – 5**

**What covered in previous lecture:**

* **What is docker ?**
* **Why docker**
* **What is containerisation**
* **Docker setup in linux**
* **What is Dockerfile**
* **Dockerfile Keywords**
* **DockerHub account**
* **Dockerhub Integration with VMs -Docker login**
* **Dockerhub push & pull**
* **Working with Docker Images**
* **Java Dockerfile**
* **Next.js Dockerfile**
* **Python app with Docker**
* **Docker Commands**
* **Create & Containerize your first (Lab)**
* **Docker File – Docker Image**

**What you will Learn in this Tutorial:**

* **Docker Port mapping**
* **Docker Volumes**
* **Docker Compose**
* **Create & Containerize your first (Lab)**
* **Docker File – Docker Image**
* **Overview of Docker Hub**
* **What is docker-compose file?**
* **Why do we need this?**
* **Writing a docker-compose file**
* **Pre-requisite**
* **Starting the services**
* **Scale up or Scale down the services**
* **Inspect the container to see container details**
* **Stopping the services**
* **References**

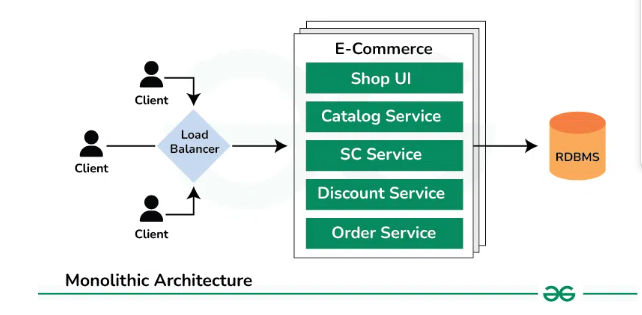
****

**Docker overview**

**Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.**

**What is a Monolithic Architecture?**

A monolithic architecture is a traditional approach to designing software where an entire application is built as a single, indivisible unit. In this architecture, all the different components of the application, such as the user interface, business logic, and data access layer, are tightly integrated and deployed together.



**Advantages of using a Monolithic Architecture**

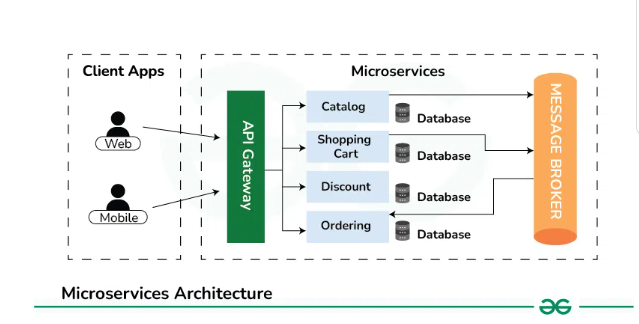
Below are the key advantages of monolithic architecture:

* **Simplicity**
  + With a monolithic architecture, all the code for your application is in one place. This makes it easier to understand how the different parts of your application work together.
  + It also simplifies the development process since developers don’t need to worry about how different services communicate with each other.
* **Development Speed**
  + Since all the parts of your application are tightly integrated, it’s faster to develop new features.
  + Developers can make changes to the codebase without having to worry about breaking other parts of the application.
  + This can lead to quicker development cycles and faster time-to-market for new features.
* **Deployment**
  + Deploying a monolithic application is simpler because you only need to deploy one artifact.
  + This makes it easier to manage deployments and reduces the risk of deployment errors.
  + Additionally, since all the code is in one place, it’s easier to roll back changes if something goes wrong during deployment.
* **Debugging**
  + Debugging and tracing issues in a monolithic application is often easier because everything is connected and in one place.
  + Developers can use tools to trace the flow of execution through the application, making it easier to identify and fix bugs

**What is a**[**Microservices Architecture**](https://www.geeksforgeeks.org/microservices/)**?**

In a microservices architecture, an application is built as a collection of small, independent services, each representing a specific business capability. These services are loosely coupled and communicate with each other over a network, often using lightweight protocols like HTTP or messaging queues.

* Each service is responsible for a single functionality or feature of the application and can be developed, deployed, and scaled independently.
* The Microservice architecture has a significant impact on the relationship between the application and the database.
* Instead of sharing a single database with other microservices, each microservice has its own database. It often results in duplication of some data, but having a database per microservice is essential if you want to benefit from this architecture, as it ensures loose coupling.



[**https://github.com/Aseemakram19/maven-app.git**](https://github.com/Aseemakram19/maven-app.git)

[GitHub - Aseemakram19/maven-app](https://github.com/Aseemakram19/maven-app)

**What is docker-compose file?**

**It is a tool which is helpful in defining and running multi-container docker applications.**

**We configure application services into a yaml file i.e docker-compose.yml**

**We can start/stop services with simple commands:**

**docker-compose up : To start the services.**

**Docker-compose up –build**

**docker-compose down : To stop the services.**

**In addition, it is easy to scale up or scale down specific services whenever required.**

**Why do we need this?**

**Docker compose is most suitable with micro-service application architecture:**

**Earlier monolithic application architecture was used. The problem with monolithic applications is that if any single service of the application goes down, the whole application will not be accessible. And this creates huge impact on customer experience.**

**Meanwhile, In micro-service architecture we have a separate application, separate database and separate server for different different services.**

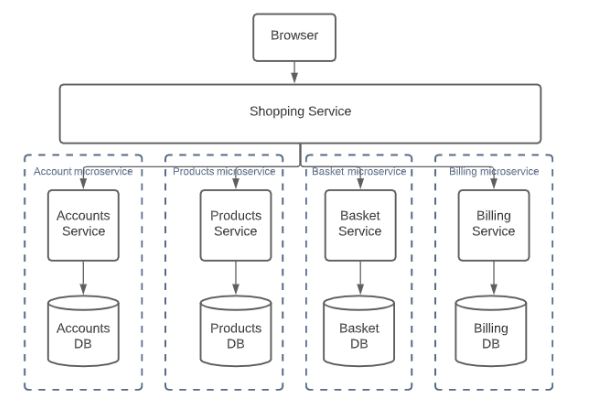
**So like in future if any one service goes down, it doesn’t affect any other service and user can use rest of the services seamlessly.**

**Also in case suppose we want to scale up any specific service only due to heavy traffic or increased demand we can just scale up that specific service and this would be cost efficient as well.**

**For instance consider an example of online shopping application, we have multiple modules for multiple services.**

**Here we have 4 different micro-services for whole application,**

* **Accounts service**
* **Products Service**
* **Basket Service**
* **Billing Service**



**Writing a docker-compose file**

**Pre-requisite**

**Firstly, make sure to have docker-compose installed in you system. However If not already installed, install from here.**

**Create a file with name docker-compose.yml and use the below code snippet to begin,**

**version: "3"**

**services:**

**application:**

**image: app2**

**ports:**

**- "8080:7000"**

**networks:**

**- springboot-db-net**

**depends\_on:**

**- mysqldb**

**volumes:**

**- /data/springboot-app**

**mysqldb:**

**image: mysql:5.7**

**networks:**

**- springboot-db-net**

**environment:**

**- MYSQL\_ROOT\_PASSWORD=root**

**- MYSQL\_DATABASE=sbms**

**volumes:**

**- /data/mysql**

**networks:**

**springboot-db-net:**

**In above code snippet, we are creating two different services, a volume and a network which uses default bridge driver,**

* **webserver: We have used nginx image and mapped it to port 9090:80 on a network webapp-network**
* **database: We have used redis image on a network webapp-network.**

**version: '3'**

**services:**

**webserver:**

**image: nginx**

**ports:**

**- 9090:80**

**volumes:**

**- .:/code**

**- logVolume:/var/log**

**networks:**

**- webapp-network**

**database:**

**image: redis**

**networks:**

**- webapp-network**

**networks:**

**webapp-network:**

**driver: bridge**

**volumes:**

**logVolume: {}**

**In above code snippet, we are creating two different services, a volume and a network which uses default bridge driver,**

* **webserver: We have used nginx image and mapped it to port 9090:80 on a network webapp-network**
* **database: We have used redis image on a network webapp-network.**

**LAB**

**1. AWS Account - Free**

**2. EC2 Create Ubuntu**

**3. Docker installation**

**4. Code App – github repo**

**5. Docker File**

**6. Docker Images**

**7. Docker Container**

**Permissions**

**sudo apt install docker.io -y**

**sudo usermod -aG docker ubuntu**

**sudo usermod -aG docker jenkins**

**groups jenkins**

**sudo chown ubuntu:docker /var/run/docker.sock**

**sudo chmod 660 /var/run/docker.sock**

**Starting the services**

**docker-compose up -d**

**groups Jenkins**

**Scale up or Scale down the services**

**Scaling up database service**

**Similarly, scaling down database services**

**Inspect the container to see container details**

**docker container inspect <containerID>**

**This will give a json response containing all the details related to container. We can find network and volume associated with the container.**

* [**Stopping the services**](https://blog.nashtechglobal.com/understanding-the-docker-compose-file/#elementor-toc__heading-anchor-7)

**docker-compose down**

**sudo chown root:docker /var/run/docker.sock**

**sudo chmod 660 /var/run/docker.sock**

**sudo systemctl start docker**

**sudo systemctl enable docker**

**sudo docker build . -t flask-app:latest**

**sudo docker run -d -p 7000:7000 flask-app:latest**

**Docker Basic Commands**

**1. List Running Containers**

**docker ps**

**This command lists all currently running containers.**

1. **Check Docker Version**

**docker --version**

**This command checks the version of Docker installed on your system.**

1. **Get Detailed System-Wide Information**

**docker info**

**This command provides detailed information about the Docker installation, including the number of containers, images, storage driver, kernel version, operating system, and more.**

1. **Docker service status**

* **systemctl status docker**
* **systemclt start docker**
* **systemctl stop docker**

1. **Search an Image in local and Docker Hub Repository**

**docker search ubuntu**

1. **Pull an Image from Docker Hub**

**docker pull <image\_name>**

**For example, to pull the latest Ubuntu image:**

**docker pull ubuntu:latest**

1. **List Docker Images**

**docker images**

**This command lists all Docker images available on your local machine.**

1. **Run a Docker Container**

**docker run -it <image\_name> /bin/bash**

**For example, to run a container with the Ubuntu image:**

**docker run -it ubuntu /bin/bash**

**The -it flags keep the container interactive.**

1. **List All Containers**

**docker ps -a**

**This command lists all containers, including those that are stopped.**

1. **Stop a Running Container**

**You can get the <container\_id> from the docker ps command.**

**docker stop <container\_id>**

1. **Remove a Container**

**docker rm <container\_id>**

1. **To kill all running containers:**

**Docker kill $ (docker ps -q)**

1. **To delete all stopped containers:**

**docker rm $(docker ps -a -q)**

1. **Use the -f flag to force remove a running container:**

**docker rm -f <container\_id>**

1. **Remove an Image**

**docker rmi <image\_id>**

**You can get the <image\_id> from the docker images command.**

1. **To delete all images:**

**docker rmi $(docker images -q)**

1. **Build an Image from a Dockerfile**

**docker build -t <image\_name> .**

**The -t flag tags the image with a name. The . specifies the current directory containing the Dockerfile.**

1. **View Container Logs**

**docker logs <container\_id>**

1. **Execute a Command in a Running Container**

**docker exec -it <container\_id> <command>**

1. **For example, to open a bash shell in a running container:**

**docker exec -it <container\_id> /bin/bash**

