DevOps

Sessions 9 & 10

- Introduction to DevOps
- DevOps ecosystem
- DevOps phases
- Introduction to containerisation
- Introduction to docker
- Creating docker images using Dockerfile
- Container life cycle

DevOps

Instead of working in silos, both teams collaborate throughout the entire software lifecycle—from planning and coding to testing, deployment, and monitoring.

Challenges in a Regular Application Deployment

- Many a time, the Ops team is unaware of the tech parts required for the app: which version of the framework that has been used, OS versions, dependencies, plugins and many more. Even a change in a config file, they are not aware of it. It used to take days together to resolve these minor issues.
- Integration of multiple config files was a humongous task and would take weeks to resolve them.
- Apps are expected to be created partially, deployed and tested on the current setup instead of creating the whole app and finally testing it. Only during the final deployments, things were figured out and explained to the developers of the expectations which might have altered the actual development.
- The solution to all such problems is : Devs + Ops = DevOps.
- Both the teams are expected to work from the beginning till the end of the Product closure. From sprint planning, collaborative work, resolving deployment issues, config managements and many more. In this case, both the teams are aware of each other's work. The ops team will be with the dev team during the first cycle itself and their dependencies will gradually reduce over the time.
- As both the teams work together hand in hand, they understand the environment on both the ends, resolve the issues quickly instead of waiting till the end, the final deployment will be easier as both the teams are aware of the environment and they know on how to solve the issues if raised.

Introduction to DevOps

- DevOps is a modern approach to software development that combines Development (Dev) and Operations (Ops) to improve the speed, quality, and efficiency of software delivery.
- It focuses on collaboration, automation, and continuous integration and delivery (CI/CD) to deliver applications and services faster and more reliably

Objectives of DevOps:

- Faster software delivery
- Improved collaboration between development and operations teams
- Automation of manual tasks (e.g., testing, deployment)
- Continuous feedback for improvement
- High system stability and performance

Main Components of DevOps:

- Continuous Integration (CI): Developers frequently integrate their code into a shared repository. Automated tests are run to detect bugs early.
- Continuous Delivery (CD): Software is automatically delivered to production or staging environments after successful CI.

- Infrastructure as Code (IaC) : Infrastructure (like servers and networks) is managed using code and automation tools.
- Monitoring and Logging : Tools track the performance of the application and identify any issues in real time.

Stage	Popular Tools
Planning & Tracking	Jira, Trello, Azure Boards
Source Control	Git, GitHub, GitLab, Bitbucket
Build & Integration	Jenkins, CircleCI, Travis CI
Testing	Selenium, JUnit, Postman, Cypress
Deployment	Docker, Kubernetes, Ansible, Terraform
Monitoring	Prometheus, Grafana, New Relic, Splunk

DevOps phases

- The DevOps lifecycle is a continuous loop of practices that integrates development and operations teams to deliver high-quality software faster and more reliably.

7 C's of DevOps:

Continuous Development

- Involves planning and coding.
- Code is written in small, manageable chunks and stored in version control systems like Git.

Continuous Integration (CI)

- Developers frequently merge code into a shared repository.
- Automated builds and tests are triggered to detect issues early.

Continuous Testing

- Automated tests (unit, integration, UI) are run to ensure code quality.
- Tools like Selenium, JUnit, and Cypress are commonly used.

Continuous Deployment/Delivery (CD)

- Code that passes tests is automatically deployed to production or staging environments.
- Ensures faster and more reliable releases.

Continuous Monitoring

- Real-time monitoring of applications and infrastructure.
- Tools like Prometheus, Grafana, and New Relic help detect performance issues and outages.

Continuous Feedback

- Feedback from users and monitoring tools is gathered to improve future iterations.
- Helps teams stay aligned with user needs and expectations.

Continuous Operations

- Focuses on maintaining system uptime and performance.
- Includes automated scaling, failover, and recovery processes.

This lifecycle forms an infinite loop of improvement

Introduction to Containerisation

- Containerisation is the process of lightweight bundling an application together with all its dependencies (libraries, configuration files, binaries) into a single unit called a container.
- This container can run consistently across different environments—whether it's a developer's laptop, a test server, or a cloud platform.
- Simply put, It is a efficient way to package and run applications—especially useful in modern cloud-native development.

How It Works?

- A Dockerfile defines the configuration of the container.
- The application and its dependencies are built into a container image.
- This image is run as a container on any system with a container runtime.

Why Use Containers?

- Portability : "Write once, run anywhere" becomes a reality.
- Speed : Containers start up in seconds.
- Scalability : Ideal for microservices and cloud-native apps.
- Efficiency : Multiple containers can run on the same machine with minimal overhead.
- Fault Isolation : If one container fails, others remain unaffected.

Popular Tools - Docker, Kubernetes, Podman, CRI-O, containerd

Introduction to Docker

- Docker is an open-source containerization platform that allows developers to package applications along with all their dependencies into containers.
- These containers ensure that the application runs the same across all environments – development, testing, and production.
- Docker simplifies the process of building, shipping, and running applications, making it a key tool in DevOps and cloud-native development.

Why Docker?

- Before Docker, developers faced the common issue of “It works on my machine.” Docker solves this by packaging everything an application needs into a lightweight container, ensuring consistency across all systems.

Key Features of Docker:

- Portability - Docker containers can run on any platform that supports Docker – Windows, Linux, or the cloud.
- Lightweight - Containers share the host OS, making them faster and more efficient than virtual machines.
- Isolation - Each container runs independently without interfering with others.
- Fast Deployment - Applications start quickly, enabling faster development and scaling.

Keywords

- Docker Engine - The core runtime that manages containers.
- Dockerfile - A script containing instructions to build a Docker image.
- Docker Image - A snapshot of the application and its environment, used to run containers.
- Docker Container - The running instance of a Docker image.
- Docker Hub - A cloud-based registry to share and download Docker images.

Basic Docker Workflow:

- Write a Dockerfile that defines how to set up the application.
- Build a Docker image using the Dockerfile.
- Run the image as a container.
- Deploy or scale the container on any system or cloud.

Example Use Case:

A web developer can containerize a Node.js application with MongoDB and run it across different environments without worrying about installation issues.

Creating docker images using Dockerfile

- A Dockerfile is a plain text file that contains a set of instructions used to build a Docker image.
- These images are templates for running containers that include the application and all required dependencies.

Docker Commands

- FROM- Specifies the base image (e.g., ubuntu, node)
- RUN - Executes a command (e.g., install packages)
- COPY - Copies files from host to the image
- WORKDIR - Sets the working directory for subsequent commands
- CMD - Defines the default command to run in the container
- EXPOSE - Indicates the port on which the container listens

Steps to Build and Run an Image

Steps for installing a Java App as a container inside a Docker

1. Create a folder called javaApp
2. Develop the requried code for the javaApp. create a Sample class and do a hello world program.
3. Build the App and test it.
4. U must create the Dockerfile, a text file with no extensions. Provide the appropriate instructions

```
Openjdk:11

WORKDIR /var/www/java

COPY . /var/www/java/

RUN javac SampleFile.java

CMD exec java SampleFile
```

- U must restart the machine, then U can download the MSI file of the Docker from
- Install the Docker Desktop from the link https://www.docker.com/products/docker-desktop/
- If required, U can add Path environment variable for executing Docker commands from the cmd.
- Docker starts immediately after installation, however U can change the settings to start on request instead of Auto start.
- If the Cmd is already opened, U may have to restart it to execute the docker commands.

Using MongoDb from the Docker

Run the following commands in the order

```
docker pull mongodb/mongodb-community-server

docker run --name mongo -d mongodb/mongodb-community-server:latest

docker container ls

docker exec -it mongo mongosh

show dbs
```

Run the Docker commands to create the image and execute the Container.

```
docker build -t java-app .

docker run java-app
```

Options:

- -t => to run the app in a virtual terminal so that U can see the results.
- -i => Use this option if U want the app run in interactive mode, when U expect User inputs from the Console Window.

Lab

- Install and configure docker
- Create docker image using Dockerfile
- Start docker container
- Connect to docker container
- Copy the website code to the container
- Use docker management commands to
  - o List the images
  - o List the containers
  - o Start and stop container
  - o Remove container and image

Session 11

- Introduction to YAML
- Introduction to Docker Swarm and Docker Stack
- Introduction to Kubernetes
- Creating Kubernetes cluster
- Creating service in Kubernetes
- Deploying an application using dashboard

Introduction to YAML

- YAML (YAML Ain't Markup Language) is a human-readable data serialization format often used for configuration files in DevOps tools like Docker, Kubernetes, and Ansible.

Features:

- Uses indentation (spaces, not tabs) to represent structure
- Supports key-value pairs, lists, and nested data
- File extension: `.yaml` or `.yml`

Introduction to Docker Swarm and Docker Stack

- Docker Swarm is Docker's native clustering and orchestration tool.
- It allows you to manage a group of Docker engines as a single virtual system.
    - docker swarm init : Initializes a Swarm
    - docker service create : Deploys services in the Swarm
- Docker Stack Docker Stack is used to deploy multi-service applications defined in a docker-compose.yml file into a Swarm.

Introduction to Kubernetes

- It is a container management tool developed by Google. its main purpose is helping in managing the containerized apps on various platforms of cloud. It can also reside in Virtual servers and local servers. It is said to be one of the most popular containerization management tools.
- It is purely cloud based and comes with host of communication and automation tools that are used to maintain and manage large scale containers as one unit.
- K8s maintains clusters for managing the services. These services are grouped into nodes and pods. A service can have one or more nodes and each node can have one or more PODs. Each POD represents an independent container of microservices.
- K8s maintains these services using multiple clustors. When an App is required to be loaded, it will be loaded into Primary Clustor. When, on for some reasons, the primary clustors fail to load, it immediately invokes a secondary clustor(backup) and the repo of the secondary clustor will be provided and the app will not break down.

How K8s work?

- It a linux based Environment that shares lots of resources required to manage complex Apps. It is primarily used for distributed computing apps where the K8s abstract the underlying infrastructure and hardware resources and offers std and consistent UI that a DevOps Engineer can monitor from a common place.
- The UI tool looks simple yet allowing to perform complex operations.
- It works similar to Jenkins where one can monitor multiple apps, clusters and allocate the resources required for each of the application.
- The DevOps person can determine the amount of resources that each app needs and allocate them by either scaling up or scaling down the resources for the app to run smoothly.

Microservices:

1. They are small scale apps that are created while breaking down large app into smaller modular units which are designed to work independently of one another.
2. With the new development methodologies, where the final product is never understood and only small updations of app happen over short intervals of time, microservices are the way to go.
3. UR Service is hosted insider the serverless environment registered under a service broker who publishes your services and allows customer to discover the services based on their search engines and the service providers they are having business with.
4. Microservices can be developed under various technologies like Java-Springboot, .NET CORE and .NET WEB API and other open source projects like NODEJS-EXPRESS , NESTJS and many more.
5. Our example will be on .NET CORE to develop an ASP.NET CORE WEB API Project that will connect a SQL server database. We will have 2 Docker containers that will have our App in one Docker image and the SQL server in another docker image. They both will be integrated using a language called YAML. We use Docker Compose tool to orchestrate the interaction between the .NET CORE and SQL server database.
6. U can try creating a REACT App that consumes this REST API. U will additional middleware like CORS and other DI tools.

How to create Microservices

- VS 2022 should be selected. Create a ASP.NET CORE Web API project
- Better go for CODE FIRST Approach to create the data classes. Include the required EF Tools
- Implement the DBContext and provide the required DI feature into Program.cs
- If connecting to SQL server, we will create an YAML file for downloading image of SQL server in the Docker compose.
- Expose the service to be called by any Application.

Creating a Web API microservice:

Implementation of the service:

1. Create a new ASP.NET Core Web API project WebApiSqlServerDockerDemo in Visual Studio 2022 using .NET 7.0 and make sure you enable the Docker support while you are creating the project.
2. Open the NuGet package manager and search and install the following packages in your project.
    - Microsoft.EntityFrameworkCore.SqlServer
    - Microsoft.EntityFrameworkCore.Design
    - Microsoft.EntityFrameworkCore.Tools
3. Create a Models folder in the project root folder and create the following Product entity in that folder with the code shared.
4. create a folder called Data in our project and create a new class called OnlineShopDbContext in this folder with the code shared.
5. Define our database connection string and we can save the connection string in the appsettings.json file.
6. Register the SQL Server database provider in Program.cs file using the UseSqlServer method. The UseSqlServer method requires a database connection string and we can read and pass this information using the GetConnectionString method.

7. Implement the controller class.
8. Add Docker compose support in Visual Studio, right-click on the Web API project in the solution explorer and choose and choose Add > Container Orchestrator Support… option from the menu.
9. Follow the steps to choose Docker Compose and Linux machine
10. Build the Application with the Docker Compose as Start up to allow the containers created in the Docker
11. Run the Application.
12. U can configure the Sql Server from the SSMS

Issues with K8s:

- It needs a heavy infrastructure to showcase any application
- The complete pipeline is done by a team of testers, Devops Engineers and QAT members.
- It is a collaborative work to make UR services hosted in K8s Server. It is not user friendly like Jenkins.
- But there are many 3rd party UI tools that can manage this infrastructure.

Lab :

- Configure Kubernetes
- Configure Kubernetes Dashboard
- Setup a Kubernetes cluster
- Access application using Kubernetes service
- Deploy the website using Dashboard