Day1

SOFTWAREDEVELOPMENTLIFECYCLE: The Software Development Life Cycle (SDLC) is a structured process used for planning, creating, testing, and deploying software. It provides a systematic approach to software development, ensuring that projects are completed efficiently and meet quality standards.

Here are the typical phases of SDLC: Planning: Define the scope, objectives, and requirements of the project. This phase involves feasibility studies, resource allocation, and risk assessment.

Requirements Analysis: Gather and document user and business needs. This step forms the foundation for the software's features and functionality.

Design: Create architectural and detailed designs for the system. This includes data models, user interfaces, and technical workflows. Implementation (Coding): Developers write the code for the software based on the design documents.

Testing: Validate the software through various tests (e.g., unit, integration, system, and user acceptance testing) to ensure quality and performance. Deployment: Release the software to users, either all at once or in stages.

Maintenance: After deployment, address user feedback, fix issues, and update the software to adapt to changing needs or technologies. DevOps: DevOps is a set of practices and principles that combines development (Dev) and operations (Ops) to enhance collaboration between software developers and IT operations teams. The goal of DevOps is to streamline the software development lifecycle, improve the speed and quality of software delivery, and promote a culture of continuous improvement.

Here are its key components: Collaboration and Communication: DevOps fosters teamwork between development and operations, breaking down traditional silos. Automation: Tasks like testing, deployment, and monitoring are automated to save time and reduce human error. Continuous

Integration (CI) and Continuous Delivery (CD): Frequent integration of code changes and automated deployment processes ensure that software can be delivered quickly and reliably. Infrastructure as Code (IaC): Managing infrastructure using code allows for consistency, scalability, and version control. Monitoring and Feedback: Real-time monitoring tools track system performance, and feedback loops provide insights for improvement.

Agility: DevOps embraces iterative and incremental approaches, allowing organizations to respond quickly to changes and user needs.

DOCKER: Docker is an open-source platform that simplifies the development, deployment, and operation of applications by using containerization. Containers package an application along with all its dependencies, libraries, and configurations, ensuring it runs consistently across different environments. Here are some key points about Docker: Lightweight Containers: Unlike virtual machines (VMs), Docker containers share the host operating system's kernel, making them faster and more efficient. Portability: Containers can run on any system that supports Docker, whether it's your local machine, a data center, or a cloud environment. Isolation: Each container operates independently, ensuring that changes in one container don’t affect others. Version Control: Docker allows versioning of containers, enabling easy rollback to previous versions when needed. Ecosystem and Tools: Docker Engine: The core runtime for building and running containers.

Docker Hub: A registry where developers can find and share container images. Docker Compose: A tool for defining and managing multi-container applications. Integration: Docker integrates with tools like Kubernetes for orchestration and CI/CD pipelines to streamline workflows.

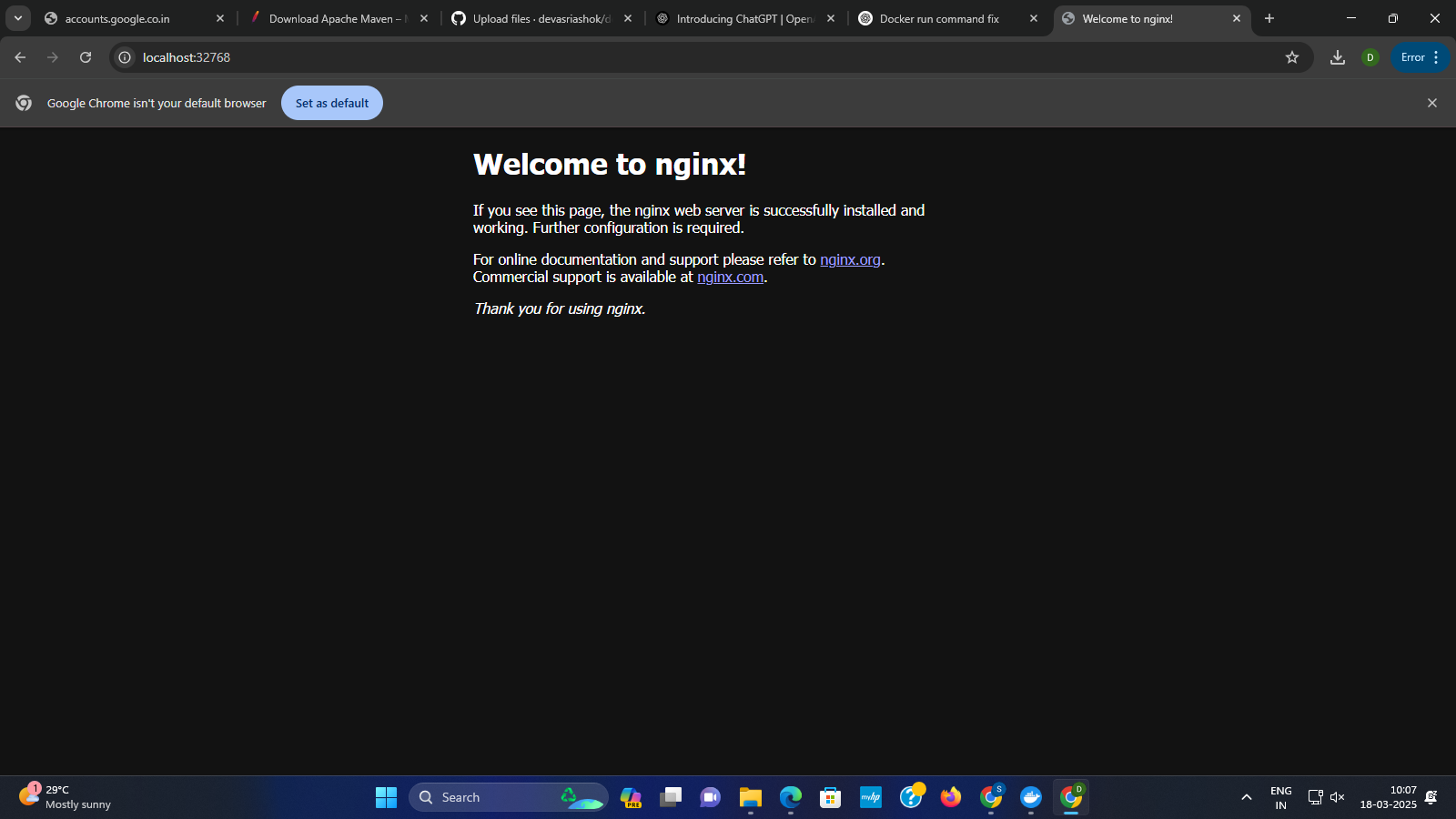
Nginx: Nginx (pronounced "engine X") is a powerful, open-source web server and reverse proxy server. It is widely used for its high performance, scalability, and ability to handle concurrent connections.

Here’s an overview of Nginx: WebServer: Nginx serves static content like HTML, CSS, and JavaScript files efficiently, making it a popular choice for hosting websites. Reverse Proxy: It acts as a gateway, forwarding client requests to backend servers (e.g., application servers) and returning the responses to the clients.

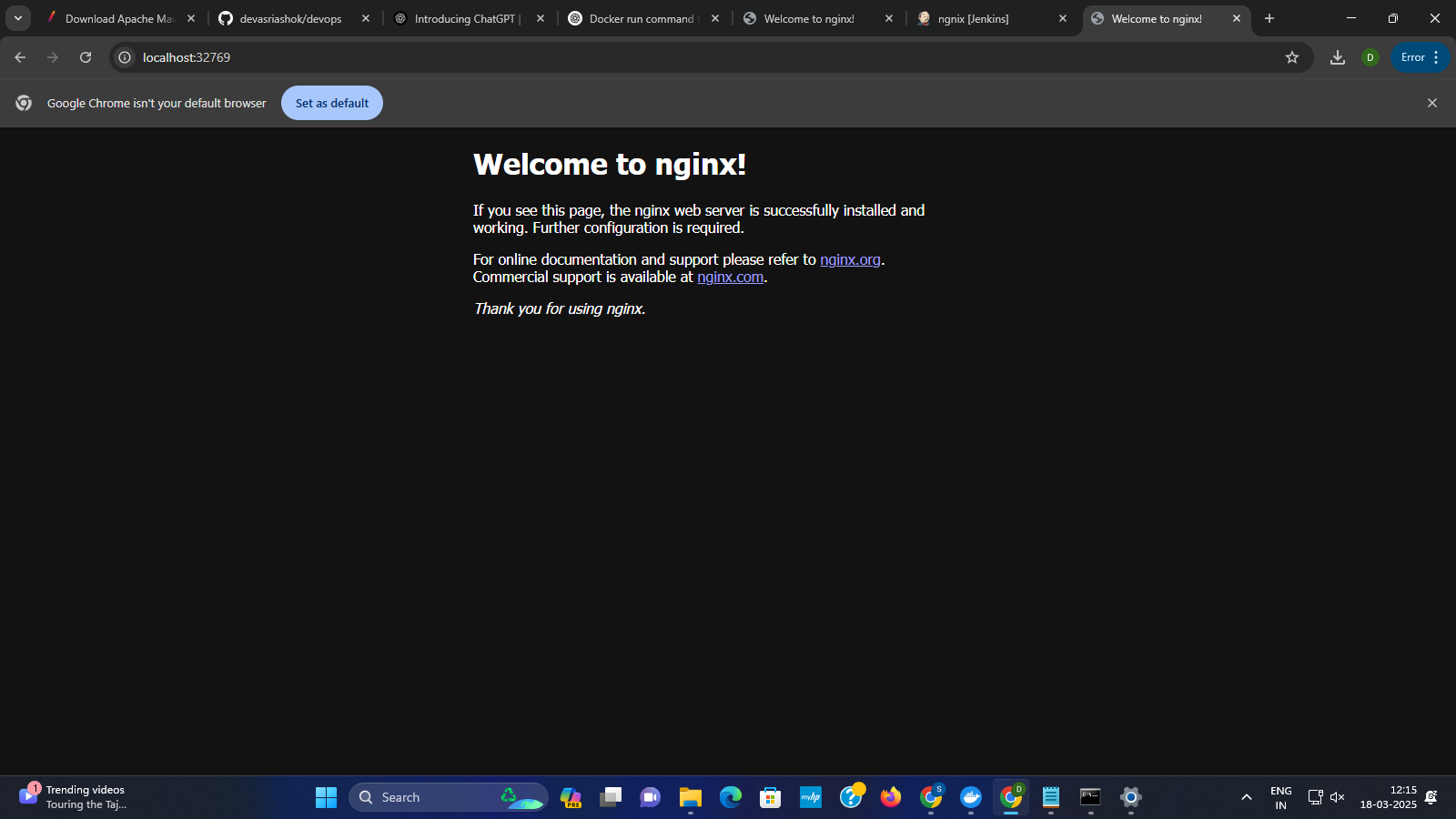
Load Balancing: Nginx distributes incoming traffic across multiple servers to ensure high availability and efficient utilization of resources. HTTP/HTTPS Support: It supports modern web protocols and secure connections using SSL/TLS.

Scalability: Nginx is designed to handle a large number of simultaneous connections with minimal resource consumption, making it ideal for high traffic sites. Use Cases: Serving static and dynamic web content. Acting as an API gateway. Caching and improving web performance. Enabling secure and scalable web architectures.

Nginx using docker:



Nginx using Jenkins:



Images in docker hub:

