**Name –Kamal D. Agrahari TE IT - A | Batch – B**

**ID – VU4F2223028 Subject – Adv DevOps Lab**

Experiment No. 0 (Prerequisite)

**Knowledge of Linux Operating System:**

Linux-based operating systems are open-source, Unix-like operating systems built around the Linux kernel, which was created by Linus Torvalds in 1991. Here are some key features and aspects of Linux-based operating systems:

1. **Kernel**

Linux Kernel: The core component of a Linux-based OS, responsible for managing system resources, hardware, and communication between hardware and software.

1. **Open Source**

Freedom and Flexibility: Linux is open source, meaning its source code is freely available for anyone to view, modify, and distribute. This encourages innovation and customization.

1. **Distributions (Distros)**

Variety: Numerous distributions (distros) exist, each tailored for different uses and preferences. Popular ones include Ubuntu, Fedora, CentOS, Debian, and Arch Linux.

Customization: Distros can be customized to suit different environments, from personal desktops to enterprise servers.

1. **Package Management**

Software Packages: Applications and system components are often distributed in packages, which can be managed using package managers like apt (for Debian-based distros), yum or dnf (for Red Hat-based distros), and pacman (for Arch-based distros).

1. **Command Line Interface (CLI)**

Terminal: Powerful command-line tools and scripting capabilities are a hallmark of Linux, enabling advanced users to perform complex tasks efficiently.

1. **Graphical User Interface (GUI)**

Desktop Environments: While Linux is often associated with the CLI, it also supports numerous desktop environments like GNOME, KDE Plasma, XFCE, and more, providing a user-friendly graphical interface.

1. **Security**

Permissions and Ownership: Linux uses a robust permissions system, allowing fine-grained control over who can access and modify files and processes.

Stability and Security: Known for stability and security, Linux is less prone to viruses and malware compared to some other operating systems.

1. **Community and Support**

Strong Community: A large, active community contributes to Linux's development, offers support through forums, mailing lists, and online resources.

Professional Support: Many distributions also offer professional support and services, making Linux suitable for enterprise use.

1. **Performance and Efficiency**

Resource Management: Linux is known for its efficient use of system resources, making it suitable for a wide range of devices from low-power embedded systems to high-performance servers.

1. **Use Cases**

Servers: Widely used in web servers, databases, and cloud environments due to its reliability and scalability.

Desktops: Offers a robust environment for daily use, with applications for productivity, multimedia, and development.

Embedded Systems: Powers a wide range of devices including smartphones (e.g., Android), routers, smart TVs, and IoT devices.

* Examples of Linux-Based Operating Systems
  + Ubuntu: Known for its user-friendliness and extensive community support.
  + Fedora: Often used by developers for its cutting-edge features and innovation.
  + CentOS: Favored by enterprises for its stability and similarity to Red Hat Enterprise Linux.
  + Debian: Known for its stability and extensive software repository.
  + Arch Linux: Preferred by advanced users for its simplicity and customization capabilities.

**Some basic command used in Linux:**

1. **ls**: Lists files and directories in the current directory.

Example: ls

1. **cd**: Changes the current directory.

Example: cd /path/to/directory

1. **pwd**: Prints the current working directory.

Example: pwd

1. **touch**: Creates an empty file or updates the timestamp of an existing file.

Example: touch filename

1. **cp**: Copies files or directories.

Example: cp source\_file destination\_file

1. **mv**: Moves or renames files or directories.

Example: mv old\_name new\_name

1. **rm**: Removes files or directories.

Example: rm filename

1. **mkdir**: Creates a new directory.

Example: mkdir directory\_name

1. **rmdir**: Removes an empty directory.

Example: rmdir directory\_name

1. **cat**: Concatenates and displays the content of files.

Example: cat filename

1. **echo**: Displays a line of text or a variable value.

Example: echo "Hello, World!"

1. **chmod**: Changes the permissions of a file or directory.

Example: chmod 755 filename

1. **chown**: Changes the ownership of a file or directory.

Example; chown user:group filename

1. **find**: Searches for files and directories in a directory hierarchy.

Example: find /path -name filename

1. **grep**: Searches for text patterns within files.

Example: grep "search\_term" filename

1. **ps**: Displays information about active processes.

Example: ps aux

1. **kill**: Terminates a process by PID.

Example: kill PID

1. **df**: Displays disk space usage of file systems.

Example: df -h

1. **du**: Shows disk usage of files and directories.

Example: du -sh /path/to/directory

1. **tar**: Archives and extracts files using tar.

Example: tar -cvf archive\_name.tar /path/to/directory

tar -xvf archive\_name.tar

**Installation and configuration of services and command line basics**

We will focus on Installation and Configuration of Services and Command Line Basics. These

are essential skills for DevOps professionals as they are responsible for setting up and managing

various services on a Linux system. Let's explore each point in more detail:

1. **Experience in Installing and Configuring Software Services on a Linux System**:

Gain hands-on experience in installing and configuring common software services on a

Linux system to support different types of applications.

DevOps professionals often need to install and configure software services like web

servers (e.g., Apache, Nginx), databases (e.g., MySQL, PostgreSQL), and other

applications to support the development and deployment of software applications.

1. **Web Servers (e.g., Apache, Nginx):**

Learn how to install and configure Apache and Nginx web servers to host websites and

serve content over the internet.

Web servers are crucial components for hosting websites and web applications. Apache

and Nginx are popular web servers used in DevOps environments.

1. **Databases (e.g., MySQL, PostgreSQL):**

Explore how to install and configure MySQL and PostgreSQL databases to store and

retrieve data for web applications.

Databases are essential for storing and managing data in software applications. MySQL

and PostgreSQL are widely used relational database management systems in DevOps.

1. **Other Applications:**

Familiarize yourself with the installation and configuration process of various

applications commonly used in DevOps environments.

Besides web servers and databases, DevOps professionals may need to install and

configure other applications and tools, such as caching servers (e.g., Redis, Memcached),

messaging systems (e.g., RabbitMQ, Apache Kafka), and more.

1. **Understanding the Basics of the Command-Line Interface (CLI):**

Understand the significance of the CLI in managing Linux systems, automating tasks,

and executing commands for various operations.

The command-line interface (CLI) is a text-based interface used to interact with the

operating system and execute commands.

1. **Common Commands for File Operations:**

Master essential commands like mkdir, cp, mv, rm, and others to efficiently manage files

and directories.

The CLI offers powerful commands for file and directory manipulation, such as creating,

copying, moving, renaming, and deleting files and directories.

1. **Common Commands for Text Processing:**

Learn how to use commands like grep to search for patterns, sed for text substitution, and

awk for text manipulation.

DevOps professionals often need to work with text files and process their contents. The

CLI provides commands like grep, sed, and awk for text processing tasks.

1. **Package Management:**

Understand package management tools and commands to install, update, and remove

software packages required for various applications.

Package managers like apt (for Debian-based systems) and yum (for Red Hat-based

systems) simplify the installation and management of software packages on Linux.

We will focus on the Basics of Computer Networks, which are fundamental for understanding

how modern software applications communicate over networks.

**Basics of Computer Networks**

1. Types of Networks
   * Local Area Network (LAN): A network confined to a small geographic area, like a single building or campus.
   * Wide Area Network (WAN): Spans large geographic areas, often composed of multiple LANs. The Internet is the largest WAN.
   * Metropolitan Area Network (MAN): Covers a city or large campus.
   * Personal Area Network (PAN): Very small, often within a single room, like Bluetooth devices connecting to a smartphone.
2. Network Topologies
   * Bus Topology: All devices share a single communication line.
   * Star Topology: All devices connect to a central hub.
   * Ring Topology: Devices form a ring where each device has exactly two neighbors.
   * Mesh Topology: Devices are interconnected with many redundant connections.
   * Hybrid Topology: Combines two or more different topologies.
3. Networking Devices
   * Router: Connects multiple networks and directs data packets between them.
   * Switch: Connects devices within a LAN and uses MAC addresses to forward data to the correct destination.
   * Hub: Basic networking device that broadcasts incoming data packets to all connected devices.
   * Modem: Converts digital data from a computer into a format suitable for a transmission medium and vice versa.
   * Access Point: Allows wireless devices to connect to a wired network.
4. IP Addressing
   * IPv4: 32-bit address, represented in dotted-decimal format (e.g., 192.168.1.1).
   * IPv6: 128-bit address, represented in hexadecimal format (e.g., 2001:0db8:85a3:0000:0000:8a2e:0370:7334).
5. Subnetting
   * Subnet Mask: Divides an IP address into a network and host portion (e.g., 255.255.255.0).
   * CIDR Notation: Specifies an IP address and its associated routing prefix (e.g., 192.168.1.0/24).
6. Protocols
   * TCP/IP: Fundamental protocol suite for the Internet, composed of:
     + TCP (Transmission Control Protocol): Ensures reliable, ordered, and error-checked delivery of data.
     + IP (Internet Protocol): Handles addressing and routing.
   * UDP (User Datagram Protocol): A simpler, connectionless protocol used for applications where speed is crucial, and error correction is handled by the application.
   * HTTP/HTTPS: Protocols for web communication.
   * FTP: Protocol for file transfers.
   * SMTP: Protocol for sending emails.
   * DNS: Resolves human-readable domain names to IP addresses.
7. Network Models
   * OSI Model (Open Systems Interconnection): Conceptual framework with 7 layers:
     + Physical: Hardware, cables, switches.
     + Data Link: MAC addresses, Ethernet.
     + Network: IP addresses, routing.
     + Transport: TCP/UDP, port numbers.
     + Session: Dialog control.
     + Presentation: Data translation, encryption.
     + Application: Network services, applications.
   * TCP/IP Model: Practical framework with 4 layers:
     + Network Interface: Combines Physical and Data Link layers.
     + Internet: Corresponds to the Network layer.
     + Transport: Same as OSI's Transport layer.
     + Application: Combines Session, Presentation, and Application layers.
8. Security
   * Firewall: Controls incoming and outgoing network traffic based on predetermined security rules.
   * VPN (Virtual Private Network): Secure connection over a less secure network, such as the Internet.
   * Encryption: Converts data into a secure format to prevent unauthorized access.
9. Wireless Networking
   * Wi-Fi: Common wireless networking technology.
   * SSID (Service Set Identifier): Name of a wireless network.
   * WPA/WPA2: Security protocols for securing Wi-Fi connections.

**Basic Networking Commands in Linux**

1. **ifconfig**: Displays or configures a network interface.

Example: ifconfig

1. **ip**: A more modern replacement for ifconfig. It displays or configures IP addresses, routes, and interfaces.

Example: ip addr show

ip route show

ip link show

1. ping: Tests connectivity to another networked device.

Example: ping <hostname or IP address>

1. **traceroute**: Shows the route packets take to a network host.

Example: traceroute <hostname or IP address>

1. **netstat**: Displays network connections, routing tables, interface statistics, masquerade connections, and multicast memberships.

Example: netstat -a

netstat -r

1. **ss**: Similar to netstat but provides more detailed information and is faster.

Example: ss -tuln

1. **hostname**: Displays or sets the system's hostname.

Example: hostname

1. **nslookup**: Queries DNS servers for DNS lookup.

Example: nslookup <hostname>

1. **dig**: A powerful DNS lookup utility.

Example: dig <hostname>

1. **route**: Displays or modifies the IP routing table.

Example: route -n

1. **arp**: Displays or modifies the ARP table.

Example: arp -a

1. **curl**: Transfers data from or to a server, supporting various protocols.

Example: curl <URL>

**Software Development Life cycle**

The Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop, and test high-quality software. It aims to produce software that meets or exceeds customer expectations, is completed within times and cost estimates, and works effectively and efficiently in the current and planned IT infrastructure. The SDLC typically consists of the following phases:

1. Planning:
   * Identify the scope and purpose of the project.
   * Define objectives and goals.
   * Conduct feasibility studies (technical, economic, and operational feasibility).
   * Develop a project management plan and schedule.
2. Requirements Analysis:
   * Gather detailed business and user requirements.
   * Create requirement specificatins.
   * Perform requirements validation to ensure accuracy and completeness.
   * Engage stakeholders to gather their needs and expectations.
3. Design:
   * Define the architecture and design of the software.
   * Create high-level and detailed design documents (including data models, interface designs, and algorithm designs).
   * Determine hardware and system requirements.
   * Plan for security, scalability, and performance.
4. Implementation (Coding):
   * Translate design documents into source code.
   * Follow coding standards and guidelines.
   * Perform unit testing on individual components.
   * Use version control systems to manage code changes.
5. Testing:
   * Develop test plans and test cases.
   * Conduct various types of testing (unit testing, integration testing, system testing, user acceptance testing).
   * Identify and fix defects.
   * Ensure the software meets quality standards and requirements.
6. Deployment:
   * Plan for deployment and release.
   * Deploy the software to the production environment.
   * Conduct final testing and validation.
   * Provide training and documentation to users and support staff.
7. Maintenance:
   * Monitor the software for issues and performance.
   * Provide ongoing support and bug fixes.
   * Implement updates and enhancements as needed.
   * Ensure the software continues to meet user needs and requirements.

**Models of SDLC**

There are several models of the SDLC, each with its own approach to handling the different phases. Some of the most common models include:

* Waterfall Model: Sequential phase-wise development model. Each phase must be completed before the next one begins.
* V-Model (Verification and Validation Model): Extension of the Waterfall model with a corresponding testing phase for each development stage.
* Iterative Model: Develops software in small segments, allowing for repeated refinement through multiple iterations.
* Spiral Model: Combines iterative development with risk assessment and mitigation. Each cycle in the spiral represents a phase in the process.
* Agile Model: Focuses on iterative development, collaboration, and flexibility. Involves continuous planning, testing, and integration.
* Scrum: An Agile framework that structures development in sprints, typically 2-4 weeks long, with a focus on delivering functional software incrementally.
* DevOps: Integrates development (Dev) and operations (Ops) for continuous delivery and deployment, emphasizing collaboration, automation, and monitoring.

**Cloud Computing and DevOps Ecosystem**

Cloud computing and DevOps are two interconnected paradigms that have significantly transformed the IT landscape, enabling faster, more reliable, and scalable software delivery. Here’s an overview of both and how they intersect:

**Cloud Computing**

Cloud computing refers to the delivery of computing services—servers, storage, databases, networking, software, analytics, and intelligence—over the Internet (“the cloud”) to offer faster innovation, flexible resources, and economies of scale.

Types of Cloud Services

1. Infrastructure as a Service (IaaS):

Provides virtualized computing resources over the internet.

Examples: Amazon Web Services (AWS) EC2, Google Cloud Platform (GCP) Compute Engine, Microsoft Azure Virtual Machines.

1. Platform as a Service (PaaS):

Provides a platform allowing customers to develop, run, and manage applications without dealing with the underlying infrastructure.

Examples: AWS Elastic Beanstalk, Google App Engine, Azure App Services.

1. Software as a Service (SaaS):

Delivers software applications over the internet, on a subscription basis.

Examples: Google Workspace, Microsoft Office 365, Salesforce.

1. Function as a Service (FaaS):

A serverless computing model where developers can run individual functions or pieces of code without managing servers.

Examples: AWS Lambda, Google Cloud Functions, Azure Functions.

Deployment Models

1. Public Cloud: Services delivered over the public internet and shared across multiple organizations.
2. Private Cloud: Cloud infrastructure dedicated to a single organization.
3. Hybrid Cloud: A combination of public and private clouds, allowing data and applications to be shared between them.
4. Multi-Cloud: Use of multiple cloud computing services in a single heterogeneous architecture.

**DevOps**

DevOps is a set of practices that combines software development (Dev) and IT operations (Ops) aimed at shortening the development lifecycle and providing continuous delivery with high software quality.

Key Components of DevOps

1. Continuous Integration (CI):
   * Practice of merging all developers’ working copies to a shared mainline several times a day.
   * Tools: Jenkins, CircleCI, Travis CI.
2. Continuous Delivery (CD):
   * Extends CI by automatically deploying all code changes to a testing and/or production environment.
   * Tools: Spinnaker, AWS CodePipeline, Azure DevOps.
3. Continuous Deployment:
   * Automatically deploys every change that passes all stages of the production pipeline to the end users.
   * Tools: GitLab CI/CD, Octopus Deploy.
4. Infrastructure as Code (IaC):
   * Managing and provisioning computing infrastructure through machine-readable definition files.
   * Tools: Terraform, AWS CloudFormation, Ansible.
5. Configuration Management:
   * Maintaining consistency of a product’s performance and functionality.
   * Tools: Chef, Puppet, SaltStack.
6. Monitoring and Logging:
   * Continuous monitoring of applications and infrastructure, and logging to analyze and respond to system performance.
   * Tools: Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana).
7. Collaboration and Communication:
   * Enhancing collaboration among teams with the use of tools and practices.
   * Tools: Slack, Microsoft Teams, Jira.

**Cloud Computing in DevOps**

The synergy between cloud computing and DevOps can be observed in the following ways:

1. Scalability:

Cloud resources can be scaled up or down based on demand, facilitating the DevOps practice of handling varying workloads.

1. Automation:

Cloud platforms provide tools for automation, crucial for DevOps practices like CI/CD and IaC.

1. Cost-Efficiency:

Cloud’s pay-as-you-go model aligns with DevOps by reducing the need for large upfront investments in infrastructure.

1. Agility:

Cloud services enable rapid provisioning and deployment, supporting the agile methodology inherent in DevOps.

1. Global Accessibility:

Cloud services can be accessed globally, allowing distributed DevOps teams to collaborate more effectively.

1. Integrated Tools:

Cloud providers often offer integrated DevOps tools and services, simplifying the setup and management of DevOps pipelines.

Ecosystem Tools

* AWS DevOps:

AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy, AWS CodePipeline.

* Azure DevOps:

Azure Repos, Azure Pipelines, Azure Boards, Azure Test Plans, Azure Artifacts.

* Google Cloud DevOps:

Google Cloud Build, Google Cloud Deployment Manager, Google Cloud Operations Suite.