**DEVOPS NOTES-1**

**As a DevOps Engineer, the role involves encouraging teamwork between development and operations teams to simplify processes and ensure smooth software delivery. Automate and manage continuous integration and deployment pipelines, automate infrastructure setup using tools like Terraform or Ansible, and setting up monitoring and logging systems to maintain application health. Security practices are integrated into the CI/CD process, and** **Automate repetitive tasks using scripts and tools to enhance efficiency. The role also involves managing cloud infrastructure, using containerization tools like Docker and Kubernetes, respond to incidents, troubleshoot issues, and ensure quick recovery. and maintaining clear documentation for consistency. Key skills required include proficiency in Linux, scripting, cloud platforms, and a strong focus on automation and problem-solving.**

**### \*\*DevOps Roles and Responsibilities\*\***

**1. \*\*Collaboration\*\*: Bridge the gap between development and operations for smoother processes.**

**2. \*\*CI/CD Pipelines\*\*: Automate and manage continuous integration and deployment pipelines.**

**3. \*\*Infrastructure as Code (IaC)\*\*: Automate infrastructure setup using tools like Terraform or Ansible.**

**4. \*\*Monitoring and Logging\*\*: Implement and maintain monitoring and logging systems to ensure application health.**

**5. \*\*Security\*\*: Integrate security practices into the CI/CD pipeline.**

**6. \*\*Automation\*\*: Automate repetitive tasks using scripts and tools.**

**7. \*\*Cloud and Containers\*\*: Manage cloud infrastructure and use Docker/Kubernetes for containerization.**

**8. \*\*Incident Management\*\*: Respond to incidents, troubleshoot issues, and ensure quick recovery.**

9. \*\*Documentation\*\*: Maintain clear documentation of processes and infrastructure.

### \*\*Key Skills\*\*

- Proficiency in Linux, scripting, and cloud platforms.

- Experience with CI/CD, IaC, and containerization tools.

- Strong problem-solving and automation skills.

**### \*\*Tools Commonly Used in DevOps\*\***

- \*\*Version Control\*\*: Git, GitHub, GitLab

- \*\*CI/CD\*\*: Jenkins, GitLab CI, CircleCI

- \*\*Configuration Management\*\*: Ansible, Puppet, Chef

- \*\*Monitoring\*\*: Prometheus, Grafana, Nagios, Datadog

- \*\*Containerization\*\*: Docker, Kubernetes

- \*\*IaC\*\*: Terraform, CloudFormation

- \*\*Logging\*\*: ELK Stack, Splunk

**### \*\*Key Skills for DevOps Professionals\*\***

- Strong understanding of Linux/Unix systems.

- Proficiency in scripting and automation.

- Experience with cloud platforms (AWS, Azure, Google Cloud).

- Familiarity with CI/CD pipelines.

- Knowledge of networking and security practices.

- Problem-solving and troubleshooting skills.

Sure! Here’s a concise version for interviews:

**### \*\*History of Linux\*\***

- \*\*Origin**\*\*: Linux was created by Linus Torvalds, a Finnish computer science student, in 1991.** Inspired by the UNIX operating system, Torvalds wanted to create a free and open-source alternative.

- \*\*Initial Release\*\*: The first version of the Linux kernel (version 0.01) was released on September 17, 1991. Torvalds shared the source code online, inviting other developers to contribute, which led to rapid improvements.

**### \*\*Distribution of Linux\*\***

**- \*\*Linux Distributions (Distros)\*\*: Linux is not just a single operating system but a family of distributions (distros), each with its own unique features, package management systems, and user interfaces. Popular distributions include:**

**- \*\*Ubuntu\*\*: Known for its user-friendliness, often recommended for beginners.**

**- \*\*Debian\*\*: A stable and reliable distribution, known for its extensive package repository.**

**- \*\*Fedora\*\*: Sponsored by Red Hat, focuses on innovation and cutting-edge features.**

**- \*\*Red Hat Enterprise Linux (RHEL)\*\*: A commercial distribution widely used in enterprise environments.**

**- \*\*CentOS\*\*: A free and open-source distribution derived from RHEL.**

- \*\*Arch Linux\*\*: Known for its simplicity and customization, popular among advanced users.

- \*\*SUSE\*\*: Known for its enterprise-grade solutions, especially in the European market.

**- \*\*Usage\*\*:**

- \*\*Servers\*\*: Linux dominates the server market due to its stability, security, and low cost. Many web servers, databases, and cloud infrastructures run on Linux.

- \*\*Supercomputers\*\*: Linux is the operating system of choice for most of the world’s supercomputers due to its performance and scalability.

- \*\*Embedded Systems\*\*: Linux is widely used in embedded systems like routers, smart TVs, and automotive systems.

**### \*\*Basic Linux Architecture and Components\*\***

**Kernel:** The **kernel** is the core component of an operating system, including Linux. It acts as a bridge between the software (applications) and the hardware (CPU, memory, devices) of a computer.

The kernel is responsible for managing the system's resources and enabling communication between the hardware and software. It controls all the basic functions of the computer, such as memory management, process management, and device management.

**Key Responsibilities of the Kernel**

1. **Process Management**:
   * **Scheduling**: Manages the execution of processes (running programs), deciding which processes should run and in what order.
   * **Multitasking**: Allows multiple processes to run simultaneously by sharing CPU time.
   * **Inter-process Communication**: Facilitates communication between different processes.
2. **Memory Management**:
   * **Allocation**: Manages the allocation and deallocation of memory to processes.
   * **Virtual Memory**: Uses virtual memory to extend physical memory by swapping data to and from the disk.
3. **Device Management**:
   * **Drivers**: Provides drivers that enable the operating system to communicate with hardware devices like disks, printers, and network cards.
   * **Input/Output**: Manages input/output operations, ensuring that data is correctly read from or written to devices.

**Kernel Mode vs. User Mode**: Operates in kernel mode with full access to the system, while applications run in user mode with restricted access to protect the system from accidental or malicious action

**Types of Kernels**

1. **Monolithic Kernel**:
   * **Single Code Base**: All kernel services run in a single address space, including device drivers, memory management, and process management.
   * **Examples**: Linux, Unix.
2. **Microkernel**:
   * **Minimalist Design**: Only essential services like process management and communication run in kernel mode; other services run in user space.
   * **Examples**: Minix, QNX.
3. **Hybrid Kernel**:
   * **Combination**: Combines aspects of monolithic and microkernels, running most services in kernel mode but modularizing them.
   * **Examples**: Windows NT, macOS.

**Importance of the Kernel**

* **Performance**: The efficiency of the kernel directly affects the performance of the entire operating system.
* **Stability**: A well-designed kernel contributes to the stability and reliability of the system.
* **Security**: The kernel’s security mechanisms protect the system from unauthorized access and attacks.

**Summary**

The kernel is the fundamental part of an operating system that manages hardware resources and provides essential services to the software running on the computer. It plays a crucial role in ensuring that the system operates efficiently, securely, and reliably.

**Shell:** A shell is a command-line interface (CLI) that allows users to interact with the operating system by typing commands. It serves as an intermediary between the user and the operating system's kernel, enabling the execution of commands, scripts, and programs.

**Types of Shells**

1. **Bash (Bourne Again Shell)**:
   * **Most Common**: The default shell on many Linux distributions. Known for its scripting capabilities and command-line features.
   * **Features**: Command history, tab completion, and shell scripting support.
2. **Sh (Bourne Shell)**:
   * **Original Shell**: A simple and widely supported shell, often used in scripting because of its portability.
3. **Zsh (Z Shell)**:
   * **Extended Features**: Offers features like advanced autocompletion, theme customization, and better scripting capabilities compared to Bash.
4. **Csh (C Shell)**:
   * **C-like Syntax**: Syntax resembles the C programming language, and it includes features like aliases and scripting capabilities.
5. **Ksh (Korn Shell)**:
   * **Advanced Features**: Combines features of both the Bourne Shell and C Shell, often used for scripting.

**Key Functions of a Shell:**

1. Command Execution:

* Run Programs: Users can type commands to run programs, manage files, and perform system tasks.
* Example: Typing ls lists files in the current directory.

1. Shell Scripting:

* Automation: Users can write scripts to automate repetitive tasks. These scripts are sequences of commands stored in a file.
* Example: A script to back up files could include commands to copy files to a backup directory.

1. File and Directory Management:

* Navigation: Users can navigate the file system using commands like cd to change directories and mkdir to create directories.
* Example: cd /home/user navigates to the user's home directory.

1. Environment Management:

* Variables: Shells allow users to define environment variables that affect the behavior of the system and applications.
* Example: Setting PATH determines where the system looks for executable files.

1. Text Processing:

* Filters and Pipelines: Shells support text processing commands like grep, sed, and awk, which can be combined in pipelines to process data.
* Example: grep 'error' logfile.txt | less searches for the word "error" in a file and displays the results page by page.

1. User Interaction:

* Prompt: The shell provides a prompt where users enter commands, and it typically displays the current directory.
* Example: The prompt might look like user@hostname:~$, indicating the user is in the home directory.

**Summary**

The shell is a powerful tool that allows users to interact with the operating system through commands. Whether for daily tasks, system administration, or programming, the shell provides a versatile and efficient way to control and automate your computer's functions.

A **file system** is a method and structure that an operating system uses to manage files on a disk or partition. It organizes data, manages space, and keeps track of files and directories, allowing users to store, retrieve, and manipulate data.

**Commands To Run PowerShell:**

**1. Open PowerShell**

* Basic Access:
  + Press Windows + S, type "PowerShell", and press Enter.
* Administrative Access:
  + Press Windows + X, select "Windows Terminal (Admin)" or "Windows PowerShell (Admin)".

**2. Understand the PowerShell Environment**

* PowerShell Prompt: The command-line interface where you enter commands.
* Execution Policy: Controls the running of scripts. Check it with:

🡪**Get-ExecutionPolicy --** command in PowerShell shows the current **execution policy** for your PowerShell environment. The execution policy determines the level of restriction placed on running scripts and configuring the PowerShell environment.

**Execution Policy Levels**

Here are the common execution policy levels in PowerShell:

1. **Restricted**: No scripts can be run. This is the default setting on Windows client computers.
2. **AllSigned**: all scripts to be digitally signed by a trusted publisher before they can be run, regardless of their source (whether they are created locally or downloaded from the internet).
3. **RemoteSigned**: policy requires that all scripts and configuration files downloaded from the internet must be signed by a trusted publisher before they can be run. However, scripts and configuration files created locally on your machine do not need to be signed.
4. **Unrestricted**: No restrictions; all scripts can be run. However, there is a warning prompt for scripts downloaded from the internet.
5. **Bypass**: No restrictions; scripts run without any warnings or prompts.
6. **Undefined**: No policy is set in the current scope. This means that the policy is inherited from a higher scope.