Week - 0 (DevOps SRE Fundamentals)

- DevOps/SRE tools setup on AWS - 15mins
- DevOps Flow - 30 mins
- Redhat Linux - Introduction, and Common Commands 30mins
- Linux folder and file structure 30 mins
- AWS account setup 20 mins
- AWS basics / SRE system overview 30 mins
- DevOps projects and real-time scenarios 15 mins
- GIT(Beginner Content so that Git does not become a roadblock to understanding further topics) 20 mins

---

DevOps/SRE tools setup on AWS

What Is SRE?

- **Site reliability engineering (SRE)** is the practice of using software tools to automate IT infrastructure tasks such as system management and application monitoring. Organizations use SRE to ensure their software applications remain reliable amidst frequent updates from development teams. SRE especially improves the reliability of scalable software systems because managing a large system using software is more sustainable than manually managing hundreds of machines.

- Ensuring reliability is an important factor for any company, as it directly influences customer perception and feedback. Companies like Google and Facebook are widely regarded as reliable because they consistently maintain high availability, ensuring their services are consistently up and running. In contrast, platforms like IRCTC face challenges in reliability, experiencing downtimes that impact user experience.

- In the realm of DevOps, reliability is a core principle achieved through continuous monitoring, automated testing, and efficient deployment practices. By embracing DevOps methodologies, organizations can enhance their system reliability, minimize downtime, and ultimately cultivate positive

customer feedback. The seamless integration of development and operations in a DevOps culture allows for swift identification and resolution of potential issues, contributing to a more reliable and resilient infrastructure.

Why is site reliability engineering important?

- Improved collaboration
- Enhanced customer experience
- Improved operations planning

Key Concepts and Principles:

a. Reliability:

- Reliability is the primary focus of SRE. It is defined as the probability that a system will perform its intended function without failure over a specified period.

b. Service Level Objectives (SLOs):

- SLOs are specific, measurable targets set for the reliability of a service. They help define the acceptable level of downtime or error rates.

c. Error Budgets:

- Error budgets are the allowed amount of downtime or errors in a system within a given time frame. SREs use error budgets to balance reliability with the need for rapid development.

Role of an SRE:

SREs are responsible for ensuring the reliability, availability, and performance of services. They work closely with development teams to align reliability goals with business objectives.

SRE Practices and Methodologies:

a. Service Level Indicators (SLIs):

- SLIs are metrics used to measure the performance and reliability of a service. Examples include latency, throughput, and error rates.

b. Service Level Objectives (SLOs) and Service Level Agreements (SLAs):

- SLOs define the target reliability goals, while SLAs are formal agreements with customers or stakeholders regarding the consequences of failing to meet those goals.

c. Error Budgets:

- Error budgets quantify the acceptable level of unreliability, allowing teams to prioritize between reliability and new features.

d. Monitoring and Alerting:

- SREs implement robust monitoring and alerting systems to detect and respond to issues proactively.

e. Incident Response:

- SREs follow well-defined incident response processes to minimize downtime and impact during service disruptions.

f. Automation:

- Automation is a key SRE practice to increase efficiency and reduce human errors. It includes tasks like deployment, scaling, and recovery.

Collaboration and Communication:

a. Collaboration with Development Teams:

- SREs collaborate with development teams to ensure reliability is considered from the design phase to deployment.

b. Communication Channels:

- Effective communication is crucial, and SREs use various channels like incident postmortems, documentation, and regular meetings to share knowledge and improve processes.

Case Studies:

a. Google's SRE Model:

- Google's implementation of SRE has been influential in the industry. Their emphasis on automation, toil reduction, and error budgets has set standards for other organizations.

b. Other Industry Examples:

- Various companies, such as Netflix, LinkedIn, and Dropbox, have adopted SRE principles and practices, tailoring them to their specific needs.

DevOps

What Is DevOps?

- **DevOps** is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market. For companies their **TTM**(time to market) is very crucial in the world of competition, DevOps methodology helps in achieving fast TTM.

Traditional vs DevOps

Traditional

- **Development Team Working**: Business Requirements -> Code Development -> Store the Code on a centralised location -> Notify the Operations Team

- **Operations Team**: Download the Code from GitHub/Centralised location -> Perform Manual Testing -> Configuration of servers -> Deploy the Application on the Server

The traditional approach Involves two distinct teams: Development and Operations.
- Development Team: Responsible for application development.
- Operations Team: Responsible for testing and deploying the application.

How Traditionally We Work?

The development team develops the application code and stores it in a centralized location on the Internet. They notify the operations team via email about the code completion. The operations team then downloads the code, conducts manual testing, and deploys the application to the server. This model keeps development and operations teams working separately.

Disadvantages of Traditional

- Manual process, leading to time-consuming tasks.
- Prone to errors.
- Longer Time to Market (TTM), making it unsuitable for fast-growing companies.

How DevOps Works?

DevOps is a methodology that integrates development and operations into a single team. The DevOps team collaboratively develops and deploys applications quickly and with fewer bugs. It also helps in finding out the bugs in very early stages.

DevOps utilizes various tools to automate the development and deployment processes, ensuring efficiency.

Advantages of DevOps

- Automated process, reducing time consumption.
- Less error-prone.
- Shorter Time to Market (TTM), suitable for fast-growing companies.

DevOps Tools

Git:
Description: Git is a distributed version control system that facilitates collaborative software development. It allows teams to track changes in source code, manage different versions, and coordinate contributions seamlessly.

Jenkins:
Description: Jenkins is an open-source automation server that supports continuous integration and continuous delivery (CI/CD). It automates the building, testing, and deployment of code, streamlining the development process and ensuring code quality.

Docker:
Description: Docker is a containerization platform that enables developers to package applications and their dependencies into portable containers. These

containers can run consistently across various environments, promoting a standardized and efficient deployment process.

Kubernetes:
Description: Kubernetes is a container orchestration tool that automates the deployment, scaling, and management of containerized applications. It provides a robust framework for managing complex, distributed systems, ensuring optimal resource utilization and high availability.

Ansible:
Description: Ansible is a powerful configuration management tool that automates application configuration, infrastructure provisioning, and software deployment. It uses simple, human-readable scripts to define tasks, making it easy to manage and scale IT infrastructure.

Terraform:
Description: Terraform is an infrastructure provisioning tool that allows users to define and automate the deployment of infrastructure as code. It supports various cloud providers and ensures consistent and reproducible infrastructure deployments.

Monitoring Tools (e.g., Grafana, Prometheus, Nagios):
Description: Monitoring tools play a critical role in tracking the performance and health of applications and infrastructure. Grafana, Prometheus, Nagios, and similar tools provide insights into system metrics, facilitate proactive issue identification, and contribute to maintaining a reliable and efficient IT environment.

What is Kernel?

The kernel is a computer program that is the core of a computer's operating system(it's the brain of OS), with complete control over everything in the system. It manages the following resources of the Linux system – *File management, Process management, I/O management. Memory management, Device management, etc.*

Cloud Computing: Revolutionizing IT Infrastructure

Cloud computing represents a paradigm shift in the way businesses and individuals access, store, and manage computing resources over the Internet. Instead of relying on local servers and data centers, cloud computing enables users to access a wide array of computing services and resources on-demand, often paying for only what they use. This transformative model brings unparalleled scalability, flexibility, and cost-efficiency to the world of IT, It's easy to setup and scale as per the requirement

Key Characteristics of Cloud Computing:

On-Demand Self-Service:

Users can provision and manage computing resources as needed, without requiring human intervention from service providers.

Broad Network Access:

Cloud services are accessible over the internet from various devices such as laptops, smartphones, and tablets.

Resource Pooling:

Cloud providers pool and allocate resources dynamically, allowing multiple users to share a common infrastructure while maintaining privacy and security.

Rapid Elasticity:

Computing resources can be scaled up or down quickly to meet changing demand, ensuring optimal performance and cost efficiency.

Measured Service:

The usage of cloud resources is monitored, controlled, and billed based on consumption, providing transparency and cost predictability.

Challenges of Traditional Data Centers:

Setting up and managing a traditional data center involves significant capital investment, hardware procurement, and ongoing maintenance efforts. For startups and smaller companies, establishing and maintaining an in-house data center can be financially and operationally daunting. Challenges include:

High Initial Costs:

Purchasing hardware, networking equipment, and infrastructure components can strain the budget of startups, hindering their ability to scale.

Operational Complexity:

Managing a data center requires expertise in hardware maintenance, security protocols, and scalability planning, which can divert focus from core business activities.

Scalability Constraints:

Expanding a traditional data center to accommodate growth often involves lengthy procurement cycles and additional infrastructure investments.

The AWS Advantage:

Amazon Web Services (AWS), a leading cloud service provider, offers a comprehensive suite of cloud computing services that address these challenges:

Infrastructure as a Service (IaaS):

AWS provides a range of computing resources, including virtual servers, storage, and networking, enabling users to scale their infrastructure without the need for physical hardware.

Pay-as-You-Go Pricing:

AWS follows a pay-as-you-go pricing model, allowing startups and businesses to pay only for the resources they consume, eliminating the need for upfront capital investment.

Global Reach and Availability:

With data centers strategically located worldwide, AWS ensures low-latency access and high availability, catering to a diverse range of users across the globe.

Managed Services:

AWS offers many managed services, simplifying complex tasks such as database management, machine learning, and analytics, enabling organizations to focus on innovation rather than infrastructure management.
AWS provides an accessible, scalable, and cost-effective alternative to traditional data centers. This shift allows startups and companies to leverage advanced computing resources without the burden of managing complex infrastructure, fostering innovation and growth in an increasingly digital landscape.

AWS EC2

Amazon EC2, part of Amazon Web Services (AWS), is a highly scalable and versatile cloud computing service that empowers businesses to run virtual servers in the cloud. It forms the backbone of many applications, offering resizable computing capacity to cater to diverse workloads and use cases. Companies like Netflix use EC2 Instances in their servers to fulfill their computing requirements.

Key Features and Concepts:

Virtual Servers (Instances):

EC2 instances are virtual servers in the cloud, allowing users to run applications and workloads. Users can choose from a wide range of instance types optimized for different use cases, such as computing power, memory, and storage.

Now That we have the infrastructure ready, So let's learn about OS

What is Linux OS?

Linux is an open-source operating system known for its free, secure, flexible, and stable nature. It is used in servers, desktops, and mobiles. In this training we will be using Redhat Enterprise Linux, which is built for enterprises to maintain their industry demands, it comes up with proper securities and common tools required for configuring servers.

Basic Linux Commands

- #pwd: Prints the current working directory.
- #ls: Lists files and directories.
- #cd: Changes the directory.
- #mkdir: Creates a directory.
- #touch: Creates a file.
- #cat: Prints the content of a file.
- #id: Prints user and group IDs.

In Linux, all files and directories are stored in the root directory (/). As In Windows *C:* drive is the main drive where all system files are stored, Similarly in Linux "/" is the root directory.

How to Install Linux OS?

1. **Bare Metal:** Install Linux OS directly on the hardware.
2. **Virtualization:** Install Linux OS on a virtual machine using tools like VirtualBox or Hyper-V.
3. **Cloud:** Install Linux OS on the cloud using cloud providers like AWS, Azure, or GCP.

4. **Container:** Install Linux OS on containers using tools like Docker or Kubernetes.

Hosting Apache HTTPD Web Server on Redhat Linux OS

a) Use the below command to install Apache web server on Linux OS:

#sudo yum install -y httpd

b) Use the below Command to edit the index.html file:

#sudo vi /var/www/html/index.html

c) Use the below command to start the Apache web server:

#sudo systemctl start httpd

d) Use the below command to stop the Apache web server:

#sudo systemctl stop httpd

Vl editor

- Use the below command to edit the file using vi editor:

#vi <file-name>

- Use the below keyword to switch to inserting mode(for inserting data)

i

- Use the below shortcut to save and close the file:

:wq

- Use the below shortcut to exit from the file without saving the file:

:q!

Redhat Linux - Introduction, and common commands

Red Hat Enterprise Linux is the world's leading enterprise Linux platform, certified on hundreds of clouds and with thousands of hardware and software vendors. Redhat comes up with intensive community support and regular patches.

Basic Linux Terminal Commands

#mkdir: Create a new directory.
Example: mkdir new_directory
#mkdir dir1 dir2 dir3: create multiple dir in current loc
#mkdir -p /a/b/c/: Create a dir with multiple levels of nested dir
#mkdir -m <mode> dir_name: Create dir with specific permissions

#ls: List files and directories in the current working directory
#ls -l: display detailed info in long format
#ls -a: list file and directories along with hidden files
#ls -S: sorts files by size, with the largest first

#pwd: Displays the current working directory.

#cd ~: move to a home directory
#cd <path>: move to a specified path

#rmdir dir_name: Remove an empty dir
#rmdir -r dir_name: Remove a dir and its contents recursively

#touch: Create an empty file.
Example: touch new_file.txt

#cp: Copy files or directories.
Example: cp file.txt /path/to/destination

#mv: Move or rename files or directories.

Example: mv file.txt new_location/file.txt

#rm: Remove/delete files or directories.
Example: rm file.txt
#rm -rf: Remove directories and it's contents forcefully

#cat: Display the contents of a file.
Example: cat file.txt
#cat > file_name: Create or append data to a file using user input

#head file_name: Display the first 10 lines of a file
#tail file_name: Display the last 10 lines of a file

#vim: Text editors for creating or modifying files.
Example: vim file.txt

- *To save the file content press - esc + : + w*
- *To come out from the file - esc + : + q*
- *To save and exit the file - esc + : + wq*
- *To insert the data in the file - i*

#cp source dest: copy files and directories from one directory to another directory
Example: cp ./a.py /root/

#mv source_file dest_dir: Move a file from one location to another
#mv old_filename new_filename: Rename a file
#mv source_dir dest_dir: Move a directory
#mv old_dir new_dir: Rename a directory

#grep: Search for patterns in files.
Example: grep pattern file.txt

Linux folder and file structure

- In the Linux/Unix operating system, a unique and unifying principle is that everything is considered a file. This includes not only regular files such as images, videos, programs, and text files but also directories, which serve as repositories for various file types. Additionally, even devices like mice, keyboards, and printers are represented as files in the Linux system.

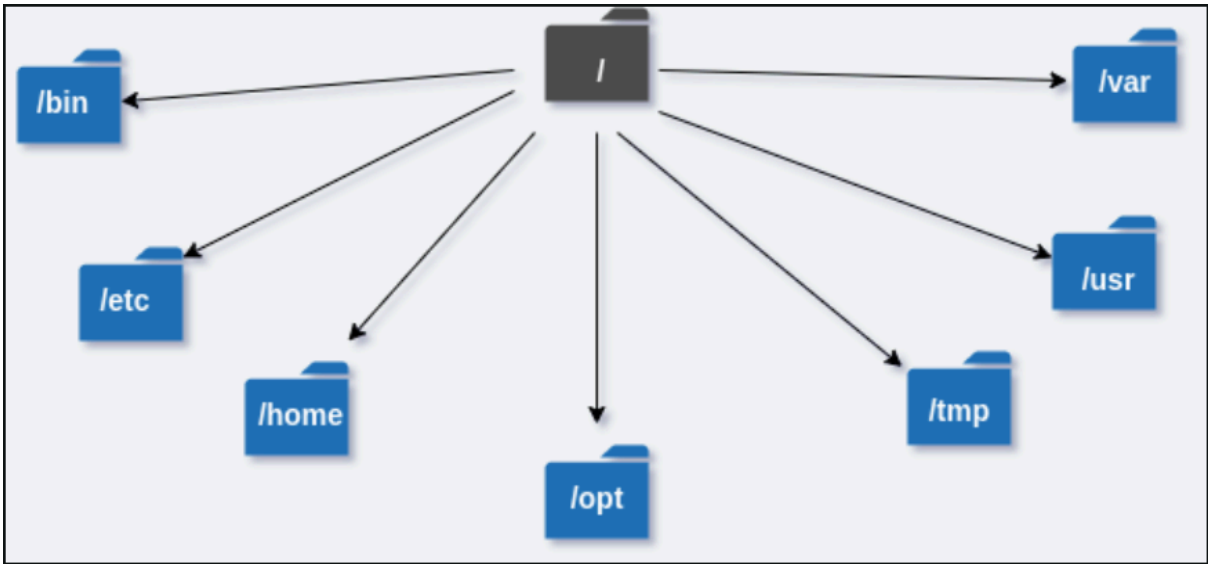
  Types of files present in the Linux file system:

  - **General Files:**

Description: Also known as ordinary files, these encompass a wide range of file types, from images and videos to programs and simple text files. They can exist in either ASCII or Binary format, constituting the most commonly used files in the Linux system.

- **Directory Files:**
  Description: Directory files act as containers for other file types and may even include subdirectories. These files play a pivotal role in organizing and structuring the Linux file system, facilitating the hierarchical organization of data.

- **Device Files:**
  Description: In contrast to Windows-like operating systems, where devices are often represented as drive letters (e.g., F:, G:, H:), Linux uniquely represents devices as files. For instance, hard drives and partitions are represented by paths such as /dev/sda1, /dev/sda2, and so forth. This file-based representation simplifies the handling and interaction with various devices within the Linux environment.



Directories

- **/bin (Binary Programs):**
  Description: The /bin directory houses essential binary or executable programs that are fundamental to the system's functionality. These programs, including basic commands like ls, cp, and mv, are crucial for

system bootstrapping and maintenance, allowing users and system processes to interact with the operating system.

- **/etc (System Configuration Files):**
  Description: The /etc directory contains system-wide configuration files that govern the behavior of various applications and services. This includes settings for user accounts, network configurations, and software packages. System administrators often modify files in this directory to customize the behavior of the operating system and installed applications.

- **/home (Home Directory):**
  Description: The /home directory serves as the default location for user home directories. Each user is typically assigned a subdirectory within /home, where they store personal files, documents, and configuration settings. This directory is the default current directory when a user logs in.

- **/opt (Optional Software):**
  Description: The /opt directory is reserved for optional or third-party software installations. It provides a designated space for software packages that are not part of the core operating system but are installed separately. This segregation helps maintain a clean and organized file system structure.

- **/tmp (Temporary Space):**
  Description: The /tmp directory is a temporary storage location designed for short-term file storage. Files in this directory are typically cleared upon system reboot. Developers and system processes use /tmp to store temporary data, facilitating various tasks and operations.

- **/usr (User Programs):**
  Description: The /usr directory contains user-related programs, libraries, and documentation. It houses a substantial portion of the system's installed software, including applications and utilities used by both system administrators and regular users.

- **/var (Variable Files and Log Files):**
  Description: The /var directory holds variable files, including system logs (located in /var/log). These log files capture information about system events, user activities, and application behaviors. By separating variable data from essential system files, /var helps ensure stability and prevent unnecessary system congestion.

Git

Git is a version control system used to track changes in a local development environment. It is a go-to tool for developers to manage their codes. Without Git managing a broad code base would be very difficult.

Why Git?

Git is essential for storing code and tracking code changes.

Basic Git Commands

- #git init: Initializes a Git repository.
- #git status: Checks the status of the Git repository.
- #git add: Adds files to the staging area.

VCS - A version control system, or VCS, tracks the history of changes as people and teams collaborate on projects together. As developers make changes to the project, any earlier version of the project can be recovered at any time.

Developers can review project history to find out:

- Which changes were made?
- Who made the changes?
- When were the changes made?
- Why were changes needed?

Repository -

A repository, or Git project, encompasses the entire collection of files and folders associated with a project, along with each file's revision history. The file history appears as snapshots in time called commits. The commits can be organized into multiple lines of development called branches. Because Git is a DVCS, repositories are self-contained units and anyone who has a copy of the repository can access the entire codebase and its history. Using the command line or other ease-of-use interfaces, a Git repository also allows for: interaction with the history, cloning the repository, creating branches, committing, merging, comparing changes across versions of code, and more.

Through platforms like GitHub, Git also provides more opportunities for project transparency and collaboration. Public repositories help teams work together to build the best possible final product.

How GitHub works?

GitHub hosts Git repositories and provides developers with tools to ship better code through command line features, issues (threaded discussions), pull requests, code reviews, or the use of a collection of free and for-purchase apps in the GitHub Marketplace. With collaboration layers like the GitHub flow, a community of 100 million developers, and an ecosystem with hundreds of integrations, GitHub changes the way software is built

AWS Account Setup

- Create a free AWS account from here - https://aws.amazon.com/
- While creating an AWS account, It will ask for credit/debit card details with the PAN number.
- AWS will deduct 2 rupees from the account which they will return after successful account creation
- Once an AWS account is created you will be able to run and manage more than 200 services of AWS

DevOps real-time scenarios

Scenario 1: **Rolling Updates with Zero Downtime**
Situation: Deploy a new version of an application without downtime.
Solution: We can use Kubernetes here to perform rolling updates, gradually replacing old pods with new ones, ensuring high availability.

Scenario 2: **Handling a Traffic Spik**e
Situation: Sudden increase in traffic causing performance issues.
Solution: Implement auto-scaling in Kubernetes to automatically adjust the number of running instances based on load.

Scenario 3: **Incident Response**
Situation: An unexpected outage occurs in the production environment.
Solution:
Use monitoring tools to quickly identify the issue.
Roll back to the last stable version using CI/CD tools.
Conduct a post-mortem analysis to prevent future incidents.

Scenario 4: **Security Patch Deployment**
Situation: A critical security vulnerability is discovered.
Solution:
Apply the security patch to the application.
Use the CI/CD pipeline to test and deploy the patched version.
Ensure continuous monitoring to detect any further issues

So these are a few scenarios that we would be learning in detail in this DevOps training.