Interview preparation

Section1

Git

Git is a distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It is widely used by developers and teams for tracking changes in source code, collaborating on projects, and managing version history.

Key Features of Git:

1. **Distributed Version Control**: Each developer has a full copy of the entire repository and history, ensuring robust collaboration and backup.
2. **Branching and Merging**: Git's branching model allows you to experiment with new features, isolate work, and merge changes easily.
3. **Performance**: Git is optimized for performance, making operations like branching, committing, and merging very fast.
4. **Data Integrity**: Git uses a cryptographic hash function (SHA-1) to identify commits and ensure the integrity of version history.
5. **Collaboration**: Teams can share changes via remote repositories like GitHub, GitLab, or Bitbucket.

Git is a fundamental tool for modern development workflows, enabling robust version control and collaboration.

Git Workflow: Git workflow involves making changes locally, staging them, committing, and then syncing with a remote repository using push/pull commands. Branching allows for isolated development and merging for integration.

Git Commands: Git commands are used for managing repositories, making commits, syncing with remotes, and handling branches (e.g., git add, git commit, git push).

Git Basic Commands: Common basic commands include git init (initialize repo), git clone (copy repo), git status (check changes), git add (stage changes), and git commit (save changes).

Pull: Fetches and merges changes from a remote.

Push: Sends committed changes to a remote.

Merge: Combines changes from different branches.

Rebase: Re-applies commits on top of another branch.

Fetch: Downloads updates without merging.

Cherry-pick: Applies specific commits from another branch.

Git Branching Strategies:

"A Git branching strategy is a way to organize how developers work on different parts of a project. For example, Git Flow is great for structured projects with planned releases, while GitHub Flow or Trunk-Based Development works well for fast-moving teams with continuous deployment. The goal is to manage code efficiently and avoid conflicts while ensuring smooth collaboration and delivery."

Here are some git commands

- git config --global user.name "Your Name" – Set your name for Git commits.
- git config --global user.email "you@example.com" – Set your email for Git commits.
- git init – Initialize a new Git repository.
- git clone <url> – Clone an existing repository to your local machine.
- git status – Show the status of changes in the repository.
- git add <file> – Stage a file for commit.
- git add . – Stage all changes for commit.
- git commit -m "message" – Save changes with a descriptive message.
- git push – Push changes to a remote repository.
- git pull – Fetch and merge changes from a remote repository.
- git branch – List all branches.
- git branch <name> – Create a new branch.
- git checkout <name> – Switch to a specific branch.
- git merge <branch> – Merge another branch into the current branch.
- git log – View the commit history.
- git log --oneline – View a simplified commit history.
- git reset <file> – Unstage a file without changing it.
- git reset --hard <commit> – Reset to a specific commit, discarding changes.

Difference Between Git and GitHub:

1. **Git**: A version control system that tracks changes in your codebase, enabling collaboration and history management. It is installed and used locally on your machine.
2. **GitHub**: A cloud-based platform for hosting Git repositories, providing additional features like collaboration tools, issue tracking, pull requests, and CI/CD integration.

In short, Git manages your code's version history, while GitHub helps share and collaborate on that code remotely

Server

A **server** is a computer or system that provides resources, data, services, or programs to other devices, called clients, over a network. Servers can handle multiple client requests simultaneously and play a critical role in hosting websites, applications, databases, and more.

Examples: **Web Server**: Delivers website content (e.g., Apache, Nginx).

Database Server: Manages and serves databases (e.g., MySQL, PostgreSQL).

File Server: Stores and shares files over a network.

JENKINS

What is Jenkins and What are Advantages: Jenkins is an open-source automation server that supports continuous integration and continuous delivery (CI/CD) for software projects. Its advantages include automating repetitive tasks, reducing errors, and enhancing collaboration and speed in development cycles.

Useful Plugins in Jenkins: Some useful Jenkins plugins include Git, Maven Integration, Pipeline, Docker, Slack Notifications, and Test Results Analyzer, enhancing automation, integration, and notifications.

Different Types of Pipeline Jobs: Jenkins pipeline jobs include **Declarative Pipeline** (simple and structured) and **Scripted Pipeline** (flexible and written in Groovy), along with **Multibranch Pipeline** (automated branch management) and **Pipeline as Code** (stored in version control).

Difference Between Declarative Pipeline and Scripted Pipeline: A **Declarative Pipeline** is more structured and easier to write, using a simplified syntax, while a **Scripted Pipeline** provides more flexibility, allowing complex logic with Groovy scripts.

DOCKER

Docker: Docker is a platform that enables developers to automate the deployment, scaling, and management of applications in lightweight, portable containers. It allows consistent environments across various systems.

Difference Between Virtualization and Containerization: Virtualization runs full operating systems on virtual machines, while containerization isolates applications at the OS level, sharing the host system's kernel for greater efficiency and speed.

What is Docker and Docker Flow: Docker is a tool for creating, deploying, and managing containers. Docker flow refers to the continuous process of building, testing, and deploying containerized applications using Docker commands and Docker Compose.

Dockerfile Instructions & Dockerfile: A Dockerfile is a script with instructions (such as FROM, RUN, COPY, CMD) to automate the process of building Docker images by specifying the OS, dependencies, and application setup.

Docker Basic and Docker Advanced Commands: Basic commands include docker build, docker run, docker ps, and docker images, while advanced commands cover docker-compose, docker network, docker volume, and docker exec.

Docker Compose and Docker Volumes: Docker Compose simplifies multi-container application setup using a docker-compose.yml file, while Docker Volumes manage persistent data storage for containers.

Docker Networking: Docker Networking allows containers to communicate with each other and with external systems through various network modes like **bridge**, **host**, **overlay**, and **none**.

KUBERNETES

Kubernetes: Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It provides features like self-healing, load balancing, and dynamic scaling, making it ideal for modern, cloud-native application development.

Kubernetes Architecture: Kubernetes follows a master-worker architecture, where the control plane manages cluster resources, and worker nodes run containerized workloads using kubelet and kube-proxy.

Kubernetes Objects: Objects include **Pods** (smallest deployable unit), **ReplicaSets** (maintain pod replicas), **Deployments** (manage updates), **Services** (expose pods), and **Ingress Controllers** (manage external HTTP/S access).

Kubernetes Autoscaling:

- **Horizontal Pod Autoscaler (HPA)**: Scales pods based on resource usage.
- **Cluster Autoscaler (CA)**: Scales worker nodes based on pending pods.

Kubernetes Deployment Strategies: Strategies include **Recreate Deployment** (stop old pods before starting new ones), **Rolling Deployment** (gradual updates), **Blue/Green Deployment** (parallel environments), and **Canary Deployment** (limited user testing).

Kubernetes YAML Files Walkthrough: YAML files define resources like **Pods**, **ReplicaSets**, **Deployments**, and **Services**, specifying configurations, metadata, and desired states for Kubernetes objects.

Kubernetes RBAC: Role-Based Access Control (RBAC) regulates access to Kubernetes resources through roles and bindings, ensuring secure and granular permission management.

Kubernetes Persistent Volumes (PV) and Persistent Volume Claim (PVC): PVs are storage resources provisioned by admins, while PVCs are user requests to use specific volumes, ensuring persistent data across pods.

Kubernetes Monitoring: Tools like Prometheus and Grafana monitor metrics like resource usage, pod health, and cluster status for performance optimization.

Kubernetes Liveness and Readiness: **Liveness probes** detect if a container is running, and **Readiness probes** check if it's ready to serve traffic, ensuring reliability and load management.

TERRAFORM

Terraform: Terraform is an open-source IaC tool by HashiCorp used for provisioning, managing, and automating infrastructure across various cloud providers.

What is IaC: Infrastructure as Code (IaC) is a method of managing infrastructure through code, allowing consistent and repeatable provisioning and configuration of resources.

What is Terraform and Terraform Lifecycle (Basic Workflow): Terraform automates infrastructure management with a lifecycle consisting of **init**, **plan**, **apply**, and **destroy** steps for resource creation and maintenance.

Terraform Basic and Advanced Commands: Basic commands include terraform init, plan, apply, and destroy. Advanced commands include terraform taint, refresh, state, and import.

What is Terraform State File and Where to Store: The Terraform state file tracks resource mappings, and it is typically stored locally or remotely in a backend like S3 or Terraform Cloud.

Terraform Backends (Local and Remote): Backends configure where Terraform stores state; **local** stores files on disk, while **remote** stores them in systems like S3, Azure Blob, or Terraform Cloud.

What is State Locking and How to Recover It: State locking prevents simultaneous state file modifications, and recovery involves manually unlocking or resolving lock files.

Terraform Provisioners (Local-Exec and Remote-Exec): Provisioners execute scripts during resource creation; **local-exec** runs scripts locally, and **remote-exec** executes commands on the remote resource.

Terraform Modules and How to Call Them: Modules are reusable infrastructure components, and you call them using the module block with a source reference.

Terraform Implicit vs Explicit Dependency: **Implicit dependencies** are inferred by Terraform based on resource references, while **explicit dependencies** are defined using depends_on.

Terraform Different Types of Variables: Variables in Terraform include **input variables** (configure resources), **output variables** (return values), and **environment variables**.

Difference Between Count and For-Each: **Count** creates multiple instances of a resource numerically, while **for_each** iterates over a map or set for dynamic resource creation.

Terraform Datasource, Null Resource, and Dynamic Blocks: **Datasources** fetch external data, **null resources** define placeholders, and **dynamic blocks** create repeatable nested configurations.

Section 2

GCP COMPUTE ENGINE

GCP Compute Engine: Google Compute Engine is a service that provides virtual machines (VMs) on Google Cloud, enabling scalable and customizable compute resources for workloads.

What is Compute Engine Machine Types: Compute Engine machine types (e.g., General Purpose, Compute Optimized, Memory Optimized) define VM specifications. Use them based on workload needs like balancing cost and performance or handling intensive computing tasks.

When to Use Compute Engine: Use Compute Engine for highly customizable and scalable virtual machines suitable for a wide range of workloads, including web hosting, batch processing, and machine learning tasks.

Types of Compute Engine Machine Types:

- **General-Purpose**: For balanced performance (e.g., web apps).
- **Compute-Optimized**: For CPU-intensive tasks (e.g., scientific simulations).
- **Memory-Optimized**: For RAM-heavy workloads (e.g., databases).
- **Accelerator-Optimized**: For machine learning and GPU-intensive tasks.

High Availability and DR (Disaster Recovery): High availability ensures minimal downtime with features like load balancing and zonal redundancy, while disaster recovery involves strategies like backups and failover to recover from major disruptions.

Autoscaling Types (Horizontal and Vertical Autoscaling): Horizontal autoscaling adjusts the number of instances, while vertical autoscaling changes resource allocation (CPU/RAM) to match workload demands dynamically.

Instance Template and Instance Groups: Instance templates define configuration settings for VMs, while instance groups use these templates to deploy and manage multiple identical VMs.

Managed Instance Group (MIG) and Unmanaged Instance Group (UIG): MIG automatically manages identical instances and supports autoscaling; UIG provides manual control over diverse instances, useful for custom setups.

Managed Instance Group (MIG): Use MIG for deploying identical VMs with automatic scaling, load balancing, and self-healing, ideal for stateless, scalable workloads like web servers.

Unmanaged Instance Group (UIG): Use UIG when managing diverse or non-identical VMs manually, suitable for stateful workloads or custom configurations where automatic updates are not required.

GCP Networking

GCP Networking (VPC): Google Virtual Private Cloud (VPC) provides isolated, scalable, and configurable networking environments for GCP resources.

Types of VPC (Custom Mode & Auto Mode):

- **Custom Mode**: You define subnets manually, ideal for precise control.
- **Auto Mode**: Automatically creates one subnet per region, suitable for quick setups.

Subnets: Subnetworks define IP ranges within a VPC and allow regional resource segmentation for efficient management.

Firewalls: Firewalls control inbound and outbound traffic in a VPC, ensuring security by applying rules based on protocols, ports, and IP ranges.

CIDR: Classless Inter-Domain Routing (CIDR) specifies IP range allocation for subnets, enabling efficient IP address management.

Types of IP Addresses:

- **Internal IP**: For communication within the VPC.
- **External IP**: For internet access or external communication.

Internet Gateway: Provides connectivity between a VPC and the internet for resources with external IPs, enabling public access.

VPC Sharing and When to Use: VPC sharing allows multiple projects to use the same VPC for centralized control, ideal for enterprise setups with shared resources.

VPC Peering and When to Use: VPC peering connects two VPCs for private communication without public internet, used for inter-project or inter-org communication.

Load Balancer Types in GCP and When to Use Each:

- **HTTP(S)**: For web applications.
- **TCP/SSL Proxy**: For non-HTTP(S) workloads.
- **Internal**: For internal traffic within a VPC.
- **Network Load Balancer**: For low-latency, high-throughput traffic.

How Do We Connect On-Prem Servers to Google Cloud Servers: Use **Cloud VPN** for secure connections or **Interconnect** for high-speed, dedicated links.

GCP Identity-Aware Proxy (IAP): IAP secures application access by requiring authentication and authorization, ideal for protecting web apps and resources.

Google Cloud Storage

Google Cloud Storage: Google Cloud Storage is a scalable, durable, and cost-effective object storage service for unstructured data like files, images, or backups.

GCP Storage Types and Storage Classes:

- **Storage Types**: Block (persistent disks), file (Filestore), and object (Cloud Storage).
- **Storage Classes**:
  - **Standard**: Frequent access.
  - **Nearline**: Access once a month.
  - **Coldline**: Access once a year.
  - **Archive**: Rarely accessed data.

How to Create Buckets and Bucket-Level Permissions: Buckets are created via Console, CLI, or API by defining names, storage classes, and locations. Permissions are managed using **IAM roles** or **ACLs** for access control.

What is Object Lifecycle: Object lifecycle rules automate actions like transition to a different storage class or deletion based on conditions like age or version.

gsutil Commands:

- **Basic**: gsutil mb (create bucket), gsutil cp (copy), gsutil ls (list).
- **Advanced**: gsutil rsync (sync), gsutil rm (delete), gsutil versioning (manage versions).

IAM

IAM: Identity and Access Management (IAM) in GCP controls access to resources by assigning roles to users, groups, and service accounts.

Organization Resource Hierarchy: GCP hierarchy includes **Organization** (top level), **Folders** (group projects), **Projects** (resource containers), and **Resources** (specific services).

Primitive, Predefined, and Custom Roles:

- **Primitive Roles**: Broad access levels (Owner, Editor, Viewer).
- **Predefined Roles**: Granular, service-specific access.
- **Custom Roles**: User-defined roles for specific access needs.

What is a Service Account, How to Create, and When to Use:

- **What**: Service accounts are special accounts used by applications or services to authenticate with GCP APIs.
- **How**: Create via Console or CLI (gcloud iam service-accounts create).
- **When**: Use for programmatic access, like deploying applications or running automated scripts securely.

App Engine

App Engine: Google App Engine is a fully managed platform for building and deploying web applications and services without managing the underlying infrastructure.

What App Engine and When to Use: Use App Engine when you need a serverless platform that automatically scales and manages application resources for web apps, APIs, and microservices, without worrying about infrastructure.

Types of App Engines:

1. **Standard**: For applications that require rapid scaling, cost efficiency, and a limited set of supported languages (e.g., Python, Java, Go). Best for lightweight, stateless apps.
2. **Flexible**: For applications with custom environments or specific requirements, supporting a wider range of languages and frameworks. Use when you need more control over the runtime or need to support heavier workloads.

Cloud Run

Cloud Run: Cloud Run is a fully managed platform that allows you to run stateless containers in a serverless environment. Use Cloud Run when you want to deploy containerized applications that automatically scale based on traffic without managing infrastructure.

What is Cloud Functions and When to Use: Cloud Functions is a serverless compute service that executes single-purpose functions in response to events. Use Cloud Functions when you need to run lightweight, event-driven code without worrying about infrastructure management.

Types of Cloud Functions:

1. **HTTP Triggered Functions**: These functions are triggered by HTTP requests, allowing them to serve as web endpoints or APIs.
2. **Event-Driven Functions**: Triggered by events from other GCP services, such as changes in Cloud Storage (e.g., file uploads), Pub/Sub messages, Firestore updates, or Cloud Logging.
3. **Background Functions**: Executed in response to asynchronous events like messages in a Pub/Sub topic or database updates, often used for backend processing.

How to Trigger in Cloud Functions: Cloud Functions can be triggered by events such as HTTP requests, changes in Cloud Storage, Pub/Sub messages, Firebase events, or Google Cloud Logging. You define triggers during function deployment.

Section 3

GCP DevOps Interview questions

Basic Linux Commands and Shell Script: Linux commands like ls, cd, cp, mv, rm, and grep are essential for managing files and processes. Shell scripting allows automation of tasks using commands like echo, if-else, for, and while loops.

What is GCSR and How to Integrate with GitHub, Jenkins, and Cloud Build: **Google Cloud Source Repositories (GCSR)** is a fully-managed Git repository service. It can be integrated with **GitHub** for source code management, **Jenkins** for CI/CD pipelines, and **Cloud Build** for automated builds by connecting repositories to trigger these services.

What is GCR and Google Artifact Repository: **Google Container Registry (GCR)** stores and manages Docker container images, while **Google Artifact Registry** is a unified repository service for storing and managing various artifacts, such as container images, language packages, and more.

What is SonarQube and How to Integrate with Jenkins or Cloud Build: **SonarQube** is a tool for continuous inspection of code quality, providing insights into bugs, vulnerabilities, and code smells. It can be integrated with **Jenkins** or **Cloud Build** by using their respective plugins to run code quality checks as part of the CI/CD pipeline.

What is Code Coverage in SonarQube: **Code Coverage** in SonarQube refers to the percentage of source code that is covered by automated tests, helping to identify untested code and improving overall software quality.

What is Maven, Nexus, and JFrog:

- **Maven**: A build automation tool for Java projects, managing dependencies and automating build processes.
- **Nexus**: A repository manager used for storing, managing, and distributing artifacts (e.g., libraries, binaries).
- **JFrog**: A company that provides tools like **Artifactory** for managing software artifacts across multiple package types.

What is CI/CD: **CI/CD (Continuous Integration/Continuous Delivery)** is a set of practices that allow developers to frequently merge code changes (CI) and automatically deploy them to production (CD), improving development efficiency and reducing deployment risks.

What is Cloud Build and Why It Is Used: **Cloud Build** is a Google Cloud service for automating the build, test, and deployment processes in the CI/CD pipeline, allowing developers to build and deploy code faster and more efficiently in a serverless environment.

What is the Difference Between Cloud Build and Jenkins: **Cloud Build** is a serverless, fully-managed service provided by Google Cloud, whereas **Jenkins** is an open-source automation server requiring manual setup and management of infrastructure.

What is Trigger and Different Types of Triggers in Cloud Build: A **trigger** in **Cloud Build** automatically initiates a build process based on events like code pushes to a repository. Types of triggers include **GitHub triggers**, **Cloud Source Repository triggers**, and **Pub/Sub triggers**.

Types of Executions in Cloud Build (Conditional Executions, Parallel, and Serial):

- **Conditional Execution**: Allows specific steps to be executed based on conditions.
- **Parallel Execution**: Steps are executed concurrently to speed up the process.
- **Serial Execution**: Steps are executed sequentially, one after another.

Explain Each Step in cloud.yaml: The **cloud.yaml** file defines the build steps in Cloud Build, including the source repository, build steps (like docker build, npm install), and any dependencies or conditions for execution.

What is GKE and Why It Is Used: **Google Kubernetes Engine (GKE)** is a fully-managed service for running and managing Kubernetes clusters, enabling easy deployment and scaling of containerized applications on Google Cloud.

Standard and Autopilot Cluster When to Use:

- **Standard Cluster**: Offers more control over configuration, ideal for users needing custom setups.
- **Autopilot Cluster**: A fully managed solution that abstracts away most configurations and is ideal for users who prefer simplicity and automatic scaling.

Difference Between Kubernetes and GKE: **Kubernetes** is an open-source container orchestration tool, while **GKE** is a fully-managed service on Google Cloud for running Kubernetes clusters.

What is Prometheus and Grafana?: **Prometheus** is an open-source monitoring and alerting toolkit for Kubernetes and other systems, while **Grafana** is an open-source platform for visualizing and analyzing monitoring data, commonly integrated with Prometheus for visualization.

GKE Monitoring Using Prometheus and Grafana: Prometheus collects and stores metrics from Kubernetes clusters, and Grafana is used to visualize this data for monitoring the performance and health of applications.

What is Cloud Operations? Why Do We Need Cloud Operations?: **Cloud Operations** (formerly Stackdriver) provides monitoring, logging, and management services to ensure GCP resources are running efficiently and help detect issues quickly.

Cloud Operations Capabilities (Monitoring, Logging, Observability):

- **Monitoring**: Tracks the performance and health of GCP resources.
- **Logging**: Collects and analyzes logs from services and applications.
- **Observability**: Helps monitor and understand the health and behavior of applications.

What is Cloud Operations Suite (Stackdriver) in GCP and Use of It: The **Cloud Operations Suite** provides integrated tools for monitoring, logging, tracing, and debugging applications on GCP, helping developers ensure the health and performance of their services.

What is Cloud Monitoring and How to Configure in GCP Resources: **Cloud Monitoring** collects metrics from GCP resources, such as VM instances and databases. It is configured through the **Google Cloud Console** or the **Monitoring API** to track performance and set up dashboards and alerts.

What is Cloud Logging and How to Configure in GCP Resources: **Cloud Logging** captures logs from GCP resources and services, allowing you to analyze and view logs in the **Cloud Console** or through the **Logging API** to troubleshoot and monitor applications.

How to Set Alerting Mechanism in GCP: Alerts can be configured using **Cloud Monitoring** by creating **alert policies** that trigger based on specific conditions like high CPU usage or memory spikes.

What is Cloud Tracer, Cloud Debugger, and Cloud Profiler and What Is Their Use?:

- **Cloud Tracer**: Helps trace requests through services, identifying bottlenecks in the application flow.
- **Cloud Debugger**: Allows live debugging of applications running on GCP without affecting production traffic.
- **Cloud Profiler**: Provides performance profiling to optimize resource usage and improve application efficiency.