1. Basics of DevOps

1. What is DevOps?

DevOps is a set of practices, cultural philosophies, and tools that integrate and automate the processes between software development and IT operations teams. The core goal is to shorten the software development lifecycle and provide continuous delivery with high software quality. It breaks down the silos between traditionally separate teams, allowing them to work together more efficiently.

2. Difference between Agile and DevOps.

Feature	Agile	DevOps
Focus	Software **development** methodology. Focuses on rapid iteration, customer feedback, and adapting to change.	A set of **practices and culture** that extends Agile principles to the entire software delivery process, from development to operations.
Scope	Limited to the development phase.	Encompasses the entire software delivery lifecycle (Plan, Code, Build, Test, Release, Deploy, Operate, Monitor).
Goal	Deliver working software frequently.	Deliver software with speed, quality, and stability.
Relation	DevOps is an extension and enabler of Agile. It makes Agile's rapid delivery possible at scale by automating the "Ops" side of things.	

Export to Sheets

3. What are the benefits of DevOps?

The main benefits of DevOps include:

- **Faster Time-to-Market:** Automation of the build, test, and deployment processes significantly reduces the time it takes to get new features and bug fixes to users.
- **Increased Efficiency:** Reduced manual effort and a streamlined workflow lead to more productive teams.
- **Improved Collaboration:** Breaking down silos between Dev and Ops teams fosters a culture of shared responsibility and communication.
- **Higher Quality and Stability:** Continuous testing and automated deployments reduce human error and lead to more reliable software.
- **Early Issue Detection:** Problems are identified and fixed early in the development cycle, reducing the cost and effort of remediation.

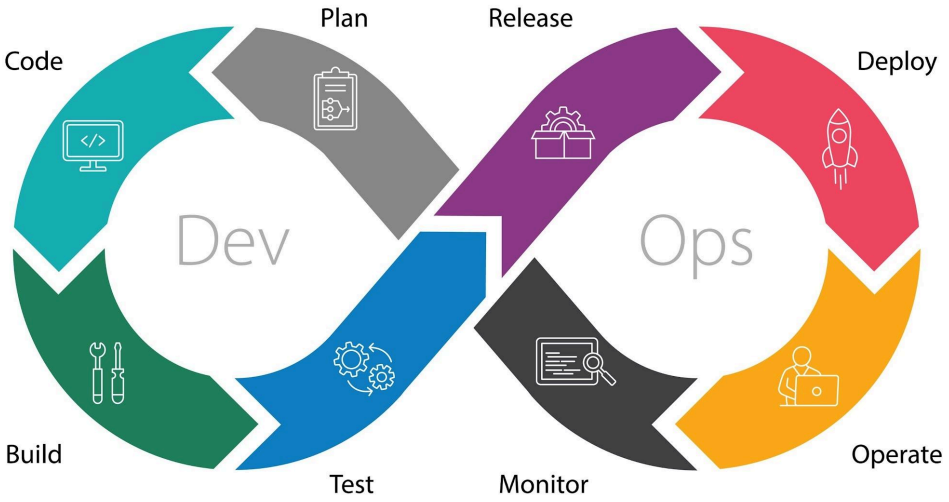
4. What are the key principles of DevOps?

The core principles can be remembered with the acronym **CALMS**:

- **C**ulture: A culture of collaboration, shared responsibility, and trust between teams.
- **A**utomation: Automating the entire pipeline, from code commit to deployment.
- **L**ean: Eliminating waste and unnecessary processes to increase efficiency.
- **M**easurement: Using metrics and data to monitor performance, identify bottlenecks, and make informed decisions.
- **S**haring: Encouraging knowledge sharing and feedback across all teams.

5. Explain the DevOps lifecycle.

The DevOps lifecycle is an iterative process that can be visualized as an **infinity loop** ♾️.



Licensed by Google

It's a continuous cycle of a few key phases:

1. **Plan:** Defining the project goals, tasks, and requirements.
2. **Code:** Writing the code and managing it with a version control system like Git.
3. **Build:** Compiling the code, running unit tests, and packaging it into a deployable artifact (like a Docker image).
4. **Test:** Running various automated tests (integration, security, etc.) on the artifact.
5. **Release:** Preparing the application for deployment.
6. **Deploy:** Pushing the application to production or a staging environment.
7. **Operate:** Managing the application in production, ensuring it's running smoothly.

8. **Monitor:** Collecting data on application performance, logging, and user feedback to inform the next planning cycle.

6. Difference between Continuous Integration (CI), Continuous Delivery (CD), and Continuous Deployment.

- **Continuous Integration (CI):** The practice of regularly merging all developers' code into a shared repository, usually several times a day. After each merge, an automated build and test process runs to detect integration issues early. The key here is a **working build**.
- **Continuous Delivery (CD):** An extension of CI where the code is not only built and tested but also prepared for a release to a production-like environment. The final step of deployment to production is still a **manual decision**.
- **Continuous Deployment:** The highest level of automation. Every change that passes the automated tests is **automatically deployed to production** without human intervention. This requires a very high level of confidence in your automated testing.

7. What are the most popular DevOps tools?

The toolchain is vast, but some of the most popular tools include:

- **Version Control:** Git, GitHub, GitLab, Bitbucket.
- **CI/CD:** Jenkins, GitLab CI/CD, GitHub Actions, CircleCI.
- **Configuration Management (IaC):** Ansible, Terraform, Puppet, Chef.
- **Containerization:** Docker.
- **Container Orchestration:** Kubernetes, Docker Swarm.
- **Monitoring & Logging:** Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana).
- **Cloud Providers:** AWS, Azure, GCP.

---

2. Version Control (Git/GitHub/GitLab)

8. What is Git and why is it used in DevOps?

Git is a distributed version control system (VCS). It's used to track changes in source code during software development. In DevOps, Git is the foundational tool for a few reasons:

- **Collaboration:** It allows multiple developers to work on the same codebase simultaneously without overwriting each other's work.
- **History & Rollback:** Git maintains a complete history of changes, making it easy to see who changed what and when, and to revert to a previous state if something goes wrong.
- **CI/CD Trigger:** The moment a developer commits code to a Git repository, it can trigger an automated CI/CD pipeline.

9. Difference between Git and GitHub/GitLab/Bitbucket.

- **Git** is the **version control system itself**—the software you install on your local machine to track code.
- **GitHub/GitLab/Bitbucket** are **web-based platforms that host Git repositories**. They provide a central remote location to store your code, as well as collaborative features like issue tracking, pull requests, and CI/CD pipelines. Think of Git as the engine and GitHub as the car dealership and service center.

10. What are Git branches?

Git branches are pointers to a specific commit. They allow developers to work on new features or bug fixes in an isolated environment without affecting the main codebase. When the work is complete, the branch is merged back into the main branch (e.g., `main` or `master`). This is a core part of collaborative development.

11. What is Git merge vs rebase?

- **Git `merge`:** Takes two or more branches and joins their histories together. It creates a new "merge commit" that has two parent commits. This keeps a clear, non-linear history of all merges.
- **Git `rebase`:** Moves a sequence of commits to a new base commit. Instead of a new merge commit, it rewrites the project history by replaying commits from one branch on top of another. This creates a clean, linear history but can be dangerous if you rebase a shared branch because it changes the commit IDs.

12. What is Git stash?

`Git stash` temporarily shelves changes you've made to your working copy, so you can work on something else and come back to them later. It's useful when you need to switch branches quickly without committing your half-finished work. For example, if you're on `feature-A` and a bug fix is urgently needed on `main`, you can `git stash`, switch to `main`, fix the bug, and then return to your `feature-A` branch and `git stash pop` your changes back.

13. What is Git workflow in DevOps?

A **Git workflow** is a set of rules or best practices for how a team uses Git. A common workflow in DevOps is the **GitFlow** or a more modern **trunk-based development**.

- **GitFlow:** Uses two main long-running branches: `main` (for releases) and `develop` (for ongoing development). Features are developed in separate `feature` branches, and releases are managed with `release` branches.
- **Trunk-based development:** Uses a single, main branch (the "trunk" or `main`) where all developers commit code directly (or through short-lived feature branches). This

requires a high degree of automation and frequent commits to avoid conflicts. It's a faster, more common approach in modern DevOps.

14. How to resolve Git conflicts?

A Git conflict occurs when two developers change the same lines of code in a file and Git can't decide which change to keep. To resolve a conflict:

1. **Git informs you** of the conflicting files when you try to merge or rebase.
2. **Open the conflicting file** in your editor. Git adds special markers (`<<<<<<<`, `=======`, `>>>>>>>`) to show you the conflicting sections.
3. **Manually edit the file** to choose which changes to keep, or to combine them.
4. **Remove the conflict markers.**
5. **`git add <file>`** to stage the changes.
6. **`git commit`** (for a merge) or `git rebase --continue` (for a rebase) to finalize the resolution.

---

3. CI/CD

15. What is Jenkins?

Jenkins is an open-source automation server. It's a CI/CD tool used to automate the software development process. Jenkins can be used to build, test, and deploy applications, and it has a vast ecosystem of plugins that allow it to integrate with virtually any tool in the DevOps pipeline.

16. Difference between Jenkins, GitLab CI/CD, and GitHub Actions.

Feature	Jenkins	GitLab CI/CD	GitHub Actions
Type	Self-hosted automation server.	Integrated feature of the GitLab platform.	Integrated feature of the GitHub platform.
Setup	Requires manual installation, configuration, and maintenance.	YAML file (`.gitlab-ci.yml`) in your repository.	YAML file (`.github/workflows/`) in your repository.
Integration	Requires plugins for most integrations.	Tightly integrated with GitLab's features (repo, registry, etc.).	Tightly integrated with GitHub's features (issues, pull requests).

Use Case	Ideal for teams that need high customization, a wide range of plugins, and fine-grained control over their build environment.	Best for teams already using GitLab for their SCM.	Best for teams already using GitHub for their SCM.

Export to Sheets

17. What is a Jenkins pipeline?

A **Jenkins pipeline** is a suite of plugins that allows you to implement and integrate a continuous delivery pipeline into Jenkins. It's a text-based definition (using a `Jenkinsfile`) of your entire CI/CD process—from source code management to deployment. Pipelines are powerful because they are:

- **Code-based:** The pipeline is defined in a file, which can be version-controlled like any other code.
- **Durable:** A running pipeline can survive a Jenkins restart.
- **Reusable:** You can reuse the same pipeline for different projects.

18. What is Blue-Green Deployment?

Blue-Green deployment is a strategy that reduces downtime by running two identical production environments: "Blue" (the current live version) and "Green" (the new version).

1. The new version is deployed to the Green environment.
2. Once tested, the router/load balancer is switched to direct all incoming traffic from Blue to Green.
3. The Blue environment is kept as a backup for a quick rollback if needed, or eventually, it's decommissioned. This ensures a near-zero-downtime deployment.

19. What is Canary Deployment?

Canary deployment is a strategy where a new version of the application is rolled out to a small subset of users first.

1. The new version (the "canary") is deployed alongside the old one.
2. A small percentage of live traffic is routed to the new version.
3. If the new version performs well and no errors are reported, the traffic is gradually increased until all users are on the new version.
4. If issues arise, the traffic is immediately switched back to the old version. This approach allows for a controlled rollout, minimizing risk and impact.

20. What is rollback in CI/CD?

Rollback is the process of reverting an application to a previously stable, working version after a failed deployment. It's a critical part of a robust CI/CD pipeline, as it allows you to

quickly restore service and minimize the impact of a bad deployment. Rollbacks are often triggered manually or automatically after a deployment health check fails.

21. What are build pipelines vs release pipelines?

- **Build Pipeline:** Focuses on the "build" and "test" stages. It takes source code, compiles it, runs unit tests, and packages it into a deliverable artifact (e.g., a JAR file or Docker image). The output is a built and tested artifact.
- **Release Pipeline:** Takes the artifact produced by the build pipeline and deploys it to various environments (dev, staging, production). This pipeline is responsible for the "release," "deploy," and "operate" phases of the lifecycle. A single build artifact can be deployed multiple times through a release pipeline.

---

4. Containers & Orchestration

22. What is Docker?

Docker is a platform that uses operating-system-level virtualization to deliver software in packages called **containers**. A container is a lightweight, standalone, executable package of software that includes everything needed to run it: code, runtime, system tools, system libraries, and settings. Docker ensures that an application will run consistently across different environments.

23. Difference between Docker and Virtual Machines.

Feature	Docker Containers	Virtual Machines (VMs)
Architecture	Share the host OS kernel.	Each VM has its own guest OS.
Size	Lightweight (MBs).	Heavy (GBs).
Startup Time	Fast (seconds).	Slow (minutes).
Portability	Highly portable, can run on any host with Docker.	Less portable, requires a hypervisor and a compatible guest OS.
Resource Usage	More efficient, less overhead.	Resource-intensive, each VM has its own OS overhead.

Export to Sheets

24. What are Docker images and containers?

- **Docker Image:** A read-only template with instructions for creating a Docker container. It's like a blueprint or a class in object-oriented programming. It's a file that includes the application code, libraries, dependencies, and configuration.

- **Docker Container:** A running instance of a Docker image. It's a live, executable version of the image. You can have multiple containers running from the same image.

25. What is a Dockerfile?

A **`Dockerfile`** is a text file that contains a set of instructions used to build a Docker image. Each instruction creates a new layer on the image. It's a simple, human-readable script. Common instructions include `FROM` (base image), `RUN` (commands to execute), `COPY` (files from host to image), and `CMD` (default command to run).

26. What are Docker volumes and networks?

- **Docker Volumes:** A mechanism for persisting data generated by and used by Docker containers. They are the preferred way to store data because they are managed by Docker, are more performant than bind mounts, and can be shared between containers.
- **Docker Networks:** Provides connectivity between containers. Docker creates a default network for each container, but you can create custom networks to allow specific containers to communicate with each other in an isolated manner.

27. What is Docker Compose?

Docker Compose is a tool for defining and running multi-container Docker applications. You define your application's services, networks, and volumes in a single `docker-compose.yml` file. Then, with a single command (`docker-compose up`), you can create and start all the services. It simplifies the orchestration of multiple interconnected containers.

28. What is Kubernetes? Why use it?

Kubernetes (K8s) is an open-source container orchestration platform. It's used for automating the deployment, scaling, and management of containerized applications. You use it because it solves the challenges of running containers in production at scale:

- **Automation:** Automates the deployment and scaling of containers.
- **Self-Healing:** Automatically restarts failed containers, replaces them, or reschedules them to other nodes.
- **Load Balancing:** Distributes traffic across multiple instances of your application.
- **Service Discovery:** Allows containers to find and communicate with each other easily.
- **Portability:** Works on any cloud or on-premise infrastructure.

29. Difference between Docker Swarm and Kubernetes.

Feature	Docker Swarm	Kubernetes

Complexity	Simple, easy to set up and use.	Complex, has a steep learning curve.
Architecture	Tightly integrated with Docker CLI.	Standalone platform.
Community	Smaller, less active community.	Huge, active community and ecosystem.
Features	Basic orchestration features.	Rich set of features for scaling and management.
Use Case	Good for small to medium-sized projects or for teams that need simplicity.	Standard for large, complex, and enterprise-level container deployments.

Export to Sheets

30. Explain Kubernetes architecture (master, worker nodes).

A Kubernetes cluster consists of at least one **master node** and one or more **worker nodes**.

- **Master Node (Control Plane):** The brain of the cluster. It's responsible for managing the state of the cluster, scheduling workloads, and handling API requests. Key components include:
    - **API Server:** Exposes the Kubernetes API.
    - **etcd:** A key-value store that holds the cluster's configuration data.
    - **Scheduler:** Assigns new pods to worker nodes.
    - **Controller Manager:** Runs various controllers that manage the cluster's state.
- **Worker Node:** These are the machines (VMs or physical servers) that run your containerized applications. Key components include:
    - **Kubelet:** An agent that runs on each node and communicates with the master.
    - **Container Runtime:** The software that runs the containers (e.g., Docker, containerd).
    - **Kube-Proxy:** Manages network rules for services and pods.

31. What are Pods, Deployments, and Services in Kubernetes?

- **Pod:** The smallest and most fundamental unit in Kubernetes. A Pod is a single instance of an application that can contain one or more containers (e.g., your app container and a logging sidecar container). Containers within a Pod share the same network namespace and storage.
- **Deployment:** An object that manages a set of identical Pods. You define a desired state (e.g., "I want 3 replicas of this Pod"), and the Deployment ensures that the desired state is maintained. It handles rolling updates and rollbacks.
- **Service:** An abstraction that defines a logical set of Pods and a policy for accessing them. Services provide a stable IP address and DNS name for a set of Pods, so they

don't have to know the individual Pod IP addresses. It allows for load balancing and service discovery.

32. What is Helm in Kubernetes?

Helm is a package manager for Kubernetes. It's like `apt` or `brew` for Kubernetes applications. It uses a packaging format called **charts**, which are collections of files that describe a related set of Kubernetes resources. Helm simplifies the process of defining, installing, and managing complex Kubernetes applications.

33. What is StatefulSet vs Deployment in Kubernetes?

Feature	Deployment	StatefulSet
Purpose	Used for stateless applications (e.g., web servers).	Used for stateful applications (e.g., databases, Kafka).
Naming	Pods have random, dynamic names (e.g., `web-1a2b3c`, `web-4d5e6f`).	Pods have stable, unique, and ordered names (e.g., `db-0`, `db-1`, `db-2`).
Scaling	Pods are scaled up and down in any order.	Pods are scaled up and down in a specific, ordered manner (e.g., `db-2` is created before `db-1` is deleted).
Storage	Pods don't require persistent storage or have a stable identity.	Pods are associated with stable, persistent storage.

Export to Sheets

---

5. Infrastructure as Code (IaC)

34. What is Infrastructure as Code (IaC)?

Infrastructure as Code (IaC) is the practice of managing and provisioning infrastructure (like servers, networks, databases, etc.) through code, rather than through manual processes. Instead of clicking through a cloud provider's UI, you write a script that defines what infrastructure you need. This makes infrastructure setup repeatable, version-controlled, and more reliable.

35. Difference between Terraform, Ansible, and CloudFormation.

Feature	Terraform	Ansible	CloudFormation
Type	Provisioning/Orchestration.	Configuration Management.	Provisioning/Orchestration.

Scope	**Cloud-agnostic**. Supports multiple cloud providers and services.	**Cloud-agnostic**. Focuses on host configuration.	**AWS-specific**. Only works on AWS.
Mechanism	**Declarative**; describes the desired end state.	**Declarative/Procedural**; can describe state but often used for a series of steps.	**Declarative**; describes the desired AWS state.
Use Case	Provisioning infrastructure (creating VMs, networks, databases).	Installing software, managing services, and configuring OS settings on existing machines.	Creating and managing AWS resources.

Export to Sheets

36. What is Terraform? How does it work?

Terraform is an open-source IaC tool created by HashiCorp. You define your desired infrastructure in configuration files written in HashiCorp Configuration Language (HCL).

How it works:

1. **Write:** You write `.tf` files that declare the resources you want to create (e.g., an AWS EC2 instance).
2. **`terraform init`:** Initializes a working directory containing Terraform configuration files.
3. **`terraform plan`:** Analyzes the configuration files and generates a plan of what actions it will take (create, update, or destroy) to achieve the desired state.
4. **`terraform apply`:** Executes the plan, creating the infrastructure as defined. Terraform also maintains a **state file** (`terraform.tfstate`) that maps your configuration to the real-world resources.

37. What is Ansible? How is it used in DevOps?

Ansible is an open-source tool used for configuration management, application deployment, and task automation. It's unique because it's **agentless**, meaning it doesn't require any special software to be installed on the machines it manages—it works over SSH.

In DevOps, Ansible is used for:

- **Configuration Management:** Ensuring all servers in an environment have the same software and configuration.
- **Application Deployment:** Pushing application code and starting services on servers.
- **Orchestration:** Automating complex, multi-step tasks across multiple servers.

38. Difference between Ansible, Puppet, and Chef.

Feature	Ansible	Puppet	Chef
Model	Push-based. The Ansible controller pushes configurations to nodes.	Pull-based. Agents on nodes pull configurations from a central master.	Pull-based. Agents pull recipes from a central server.
Architecture	Agentless. Uses SSH.	Agent-based. Requires an agent on each node.	Agent-based. Requires an agent on each node.
Language	YAML.	Puppet's own DSL (Declarative).	Ruby (Imperative).
Use Case	Simple ad-hoc tasks, application deployment, and configuration.	Best for long-term, large-scale infrastructure management.	Best for complex configuration scenarios and managing large server farms.

Export to Sheets

39. What are Terraform providers and modules?

- **Terraform Providers:** A plugin that allows Terraform to interact with a specific API, like a cloud provider (AWS, Azure, GCP), a SaaS provider (Cloudflare), or an on-premise system. A provider is responsible for understanding API interactions and exposing resources.
- **Terraform Modules:** A container for multiple resources that are used together. Modules allow you to create reusable components of your infrastructure. For example, you can create a module for a web server that includes an EC2 instance, a security group, and an EBS volume, and then reuse that module across different projects.

40. What is idempotency in Ansible?

Idempotency means that a task or operation can be performed multiple times without changing the result beyond the initial execution. In Ansible, this is a key principle. For example, if you have a task that says "ensure this file exists," running that task multiple times won't change the state of the file after the first run. This makes Ansible safe to run repeatedly, which is crucial for continuous delivery.

---

6. Cloud Platforms

41. What is cloud computing?

Cloud computing is the on-demand delivery of IT resources and applications over the internet, with pay-as-you-go pricing. Instead of owning and maintaining physical data centers and servers, you can access computing power, storage, and databases from a third-party provider like AWS, Azure, or GCP.

42. Difference between IaaS, PaaS, SaaS.

Acronym	Full Name	Description	Example
IaaS	Infrastructure as a Service	Provides fundamental computing resources (virtual machines, storage, networks). You manage the OS and applications.	AWS EC2, Azure VMs.
PaaS	Platform as a Service	Provides a platform and environment for developers to build, deploy, and manage applications without the complexity of managing the underlying infrastructure.	AWS Elastic Beanstalk, Heroku.
SaaS	Software as a Service	A complete, ready-to-use software application delivered over the internet. You just use it.	Gmail, Salesforce, Dropbox.

Export to Sheets

43. Difference between AWS, Azure, and GCP in DevOps.

Feature	AWS (Amazon Web Services)	Azure (Microsoft)	GCP (Google Cloud Platform)
Market Share	**Leader**. Most mature and extensive service offerings.	**Second place**. Strong in enterprise and hybrid cloud environments.	**Third place**. Known for innovation and data analytics.
DevOps Tools	Broad suite of native tools (CodePipeline, CodeDeploy, CodeBuild, EKS, ECS).	Strong integrations with Microsoft's ecosystem (Azure DevOps, AKS).	Excellent for containerization and machine learning (GKE).
Philosophy	"Toolbox" approach; provides many services you piece together.	"Integrated" approach; strong ties to Microsoft development tools.	"Innovation" approach; often focuses on cutting-edge tech.

Export to Sheets

44. What is AWS EC2, S3, and Lambda?

- **AWS EC2 (Elastic Compute Cloud):** Provides resizable compute capacity in the cloud. It's essentially a virtual machine (VM) that you can launch and use. You have full control over the operating system and software. It's a foundational IaaS offering.

- **AWS S3 (Simple Storage Service):** An object storage service that provides highly scalable and durable storage. You can store any type of data, from text files to videos, and retrieve them via an API. It's often used for static website hosting, backups, and data lakes.
- **AWS Lambda:** A serverless computing service. It lets you run code without provisioning or managing servers. You simply upload your code, and Lambda takes care of everything required to run and scale it with high availability.

45. What is auto-scaling in AWS?

Auto-scaling is a feature that automatically adjusts the number of EC2 instances in an application to meet traffic demands. You define a desired capacity, and auto-scaling will add or remove instances based on metrics like CPU utilization or network traffic. This ensures your application can handle traffic spikes without being over-provisioned during off-peak hours, saving costs.

46. What is Load Balancer in cloud?

A **Load Balancer** is a device or service that distributes incoming network traffic across multiple servers. It ensures that no single server is overloaded, improving application performance, availability, and reliability. If one server fails, the load balancer stops sending traffic to it and redirects it to the healthy ones. It's a critical component for high-availability applications.

47. What is serverless computing?

Serverless computing is a cloud execution model where the cloud provider dynamically manages the allocation of machine resources. You don't have to worry about provisioning, scaling, or managing servers. You just write your code, and the provider runs it in response to events (e.g., an HTTP request, a file upload). You are only billed for the time your code is running. AWS Lambda is a prime example.

---

7. Monitoring & Logging

48. What is application monitoring in DevOps?

Application monitoring in DevOps is the practice of collecting metrics, logs, and traces from your applications and infrastructure to understand their health, performance, and behavior. It's about getting real-time insights into what your systems are doing, so you can detect, diagnose, and resolve issues before they impact users.

49. Difference between Prometheus and Grafana.

- **Prometheus:** A **time-series database and monitoring system**. It's used for collecting and storing metrics from your applications and infrastructure. It uses a **pull model**, where it scrapes metrics from configured endpoints.
- **Grafana:** A **data visualization and dashboarding tool**. It doesn't collect data itself. Instead, it connects to data sources (like Prometheus) and allows you to build powerful, customizable dashboards to visualize your metrics in graphs, charts, and other formats.

50. What is ELK Stack (Elasticsearch, Logstash, Kibana)?

The **ELK Stack** is a collection of three open-source tools used for centralized logging:

- **Elasticsearch:** A distributed, RESTful search and analytics engine. It stores and indexes log data.
- **Logstash:** A data collection pipeline that ingests data from various sources, processes it, and sends it to Elasticsearch.
- **Kibana:** A data visualization and exploration tool. It provides a web interface for searching, visualizing, and analyzing the log data stored in Elasticsearch.

51. What is centralized logging?

Centralized logging is the practice of collecting log data from all applications, servers, and infrastructure components into a single, central system. Instead of having logs scattered across multiple machines, they are all in one place, making it easier to search, analyze, and monitor them. This is crucial for troubleshooting issues in a microservices or distributed environment.

52. What is alerting in monitoring tools?

Alerting is a feature in monitoring tools that automatically notifies a team when a specific metric crosses a predefined threshold or when an event occurs. For example, an alert can be triggered if CPU utilization on a server goes above 90% or if an application starts returning a high number of errors. Alerting is critical for proactive incident management.

53. What is APM (Application Performance Monitoring)?

APM is a discipline and a set of tools used to monitor and manage the performance and availability of software applications. APM tools provide a deep, end-to-end view of an application's behavior. They track things like:

- Response times
- Transaction traces
- Error rates
- Dependencies between services APM helps developers and operations teams find and fix performance bottlenecks in complex applications.

---

8. Security in DevOps (DevSecOps)

54. What is DevSecOps?

DevSecOps is the integration of security practices into the entire DevOps lifecycle. It means "shifting security left," or making security a shared responsibility from the start of the development process (planning and coding) rather than a last-minute audit. The goal is to build security in, not bolt it on.

55. How to secure Docker containers?

You can secure Docker containers by:

- **Using minimal base images:** Using a small, purpose-built base image (like Alpine Linux) reduces the attack surface.
- **Scanning images for vulnerabilities:** Using tools like Trivy or Clair to scan images for known vulnerabilities before they are used.
- **Running containers as non-root users:** This prevents a compromised container from gaining root access to the host machine.
- **Using an immutable image strategy:** Don't patch running containers; instead, build a new image with the fix and redeploy.
- **Regularly updating images:** Keep your images and their dependencies up to date to get the latest security patches.

56. What is vulnerability scanning?

Vulnerability scanning is the automated process of scanning applications, systems, or container images for known security vulnerabilities. This is a key part of DevSecOps, as these scans can be integrated into the CI/CD pipeline to automatically check for issues before code is deployed to production.

57. What is secret management? (Vault, AWS Secrets Manager)

Secret management is the practice of securely storing and managing sensitive information like API keys, passwords, database credentials, and tokens. Hardcoding these secrets in source code is a major security risk. Tools like **HashiCorp Vault** or **AWS Secrets Manager** provide a centralized, secure location to store and manage secrets, and they provide an API that applications can use to retrieve them dynamically at runtime.

58. What is OWASP Top 10?

The **OWASP Top 10** is a standard awareness document that lists the ten most critical web application security risks. It's a starting point for developers and security professionals to understand the most common attack vectors. Examples include Injection (SQL, Command), Broken Authentication, and Sensitive Data Exposure.

59. How to secure CI/CD pipelines?

Securing a CI/CD pipeline involves:

- **Securing source code:** Using a private, secure Git repository, and scanning code for vulnerabilities.
- **Access control:** Enforcing least-privilege access for all users and tools in the pipeline.
- **Secret management:** Never hardcode secrets in your pipeline scripts; use a secret management tool.
- **Vulnerability scanning:** Integrating tools to scan containers, dependencies, and code for vulnerabilities at every stage.
- **Network security:** Ensuring the build agents are on a secure network.

---

9. Advanced DevOps

60. What is Site Reliability Engineering (SRE)?

Site Reliability Engineering (SRE) is a discipline that applies aspects of software engineering to infrastructure and operations problems. The main goal is to create highly reliable and scalable software systems. SRE teams use automation and metrics to manage infrastructure and reduce the amount of manual work (toil).

61. Difference between SRE and DevOps.

Feature	DevOps	SRE
Philosophy	A culture of collaboration and shared responsibility.	A prescriptive framework for implementing DevOps.
Focus	Bridging the gap between Dev and Ops to increase speed.	**Reliability** as the primary metric. Managing reliability with code.
Tools	A broad toolchain for automation.	Focuses on specific tools and practices for reliability (monitoring, alerting, error budgets).
Relation	DevOps is the "what" (the goal), and SRE is the "how" (one way to achieve that goal).	

Export to Sheets

62. What is Chaos Engineering?

Chaos Engineering is the practice of intentionally introducing failures into a system to test its resilience. It's a proactive approach to finding weaknesses before they cause an outage. For example, a Chaos Engineering experiment might involve terminating a random instance in a production cluster to see if the system can self-heal without any human intervention.

63. What is GitOps?

GitOps is a way of implementing Continuous Deployment for cloud-native applications. It uses Git as the single source of truth for declarative infrastructure and applications. The core idea is:

1. Your entire system state (infrastructure and application configuration) is described in code within a Git repository.
2. A software agent (a controller) inside the cluster continuously observes the Git repository and the actual cluster state.
3. If the two states don't match, the controller automatically takes action (e.g., deploys a new version) to sync them. This makes deployments, rollbacks, and management a pull-based process driven by Git.

64. What is Policy as Code?

Policy as Code is the practice of writing security and compliance policies in a machine-readable format. Instead of having manual security reviews, you can enforce policies automatically using tools. For example, you can write a policy that says, "no one can deploy a container image that has not been scanned for vulnerabilities," and this policy can be enforced by a tool like Open Policy Agent (OPA) during a CI/CD pipeline run.

65. What is service mesh in Kubernetes (Istio, Linkerd)?

A **service mesh** is a dedicated infrastructure layer for handling service-to-service communication. It's a way to control how different parts of an application share data with one another. It's especially useful in complex microservices architectures. A service mesh adds features like:

- **Traffic management:** Controlling how traffic flows between services (e.g., canary deployments).
- **Observability:** Providing metrics, logs, and traces for all service communication.
- **Security:** Encrypting all service-to-service traffic (mTLS). Popular service meshes include **Istio** and **Linkerd**.

66. What is observability vs monitoring?

- **Monitoring:** The process of collecting predefined metrics and logs from your systems to understand their health. It's about answering the question, "Is my system working as expected?"
- **Observability:** A property of a system that allows you to understand its internal state from its external outputs. It's about answering the question, "Why is my system behaving this way?" Observability requires collecting not just metrics and logs, but also **traces** (end-to-end transaction flows) to understand complex, distributed systems.

67. What are SLAs, SLOs, SLIs?

These are key terms in SRE:

- **SLI (Service Level Indicator):** A quantitative measure of a service's performance. For example, "request latency" or "error rate." It's the metric you're measuring.
- **SLO (Service Level Objective):** A target value for a specific SLI. For example, "99.9% of all requests must have a latency of less than 300ms." It's the goal you are trying to meet.
- **SLA (Service Level Agreement):** A formal agreement between a service provider and a customer that specifies what will happen if the SLO is not met. It often includes penalties. It's the business promise.

68. What is zero-downtime deployment?

Zero-downtime deployment is the ability to deploy a new version of an application without any interruption to the service. This is achieved through strategies like Blue-Green and Canary deployments. It's a crucial goal in modern DevOps, as it ensures that users never experience an outage during an update.

69. What is infrastructure drift?

Infrastructure drift occurs when the actual state of your infrastructure differs from the desired state defined in your IaC code. This can happen when a developer or system administrator manually makes a change to a server (e.g., installs a new package or changes a configuration file) without updating the IaC code. Drift is a major problem because it makes your infrastructure unpredictable and difficult to manage.

70. How do you scale DevOps in large organizations?

Scaling DevOps in a large organization requires a shift from a few teams using DevOps to an enterprise-wide cultural transformation. Key strategies include:

- **Creating a Center of Excellence (CoE):** A central team that provides guidance, best practices, and support to other teams.
- **Standardizing toolchains:** Providing a recommended set of tools and platforms to avoid fragmentation.
- **Enabling self-service:** Empowering development teams to manage their own CI/CD pipelines and infrastructure through self-service portals.
- **Promoting a culture of shared responsibility:** Encouraging a "you build it, you run it" mentality across teams.
- **Implementing Policy as Code:** Enforcing security and compliance at scale through automated policies.