Docker

Basic

1. What is Docker and why is it used?

   Docker is a platform that allows developers to create, deploy, and run applications in containers. Containers are lightweight and portable, making it easier to manage and scale applications consistently across different environments

2. What is a Docker container?

   A Docker container is a lightweight, standalone, and executable package that includes everything needed to run a piece of software, including the code, runtime, system tools, libraries, and settings.

3. Explain the difference between Docker image and Docker container.

   A Docker image is a read-only template used to create containers. It includes the application and its dependencies. A Docker container is a runtime instance of an image, meaning it's the actual environment where the application runs.

4. What is Docker Hub?

   Docker Hub is a cloud-based repository where Docker users can find, share, and manage Docker images. It's similar to GitHub but for Docker images.

Intermediate

1. How do you create a Docker image?

   You create a Docker image by writing a Dockerfile, which contains instructions for building the image, and then using the `docker build` command to create the image from the Dockerfile.

2. What is a Dockerfile and what are its key components?

   A Dockerfile is a text file that contains a series of instructions on how to build a Docker image. Key components include:

   - **`FROM`: Specifies the base image.**
   - **`RUN`: Executes commands in the image.**
   - **`COPY` or `ADD`: Copies files into the image.**
   - **`CMD` or `ENTRYPOINT`: Defines the command to run when the container starts.**

3. How do you link two Docker containers?

   You can link two Docker containers using Docker networks. You can create a user-defined bridge network using `docker network create` and then run containers with `--network` to join them to the network, allowing them to communicate.

4. What is the purpose of the `docker-compose` command?

   The `docker-compose` command is used to define and run multi-container Docker applications. It uses a YAML file (`docker-compose.yml`) to configure the application's services, networks, and volumes, and the `docker-compose up` command to start all the services.

Advanced

1. **Explain how Docker handles networking.**
   - Docker provides several network drivers for networking containers:
     - `bridge`: Default network driver, suitable for standalone containers.
     - `host`: Shares the host's network stack.
     - `overlay`: For multi-host networking, used with Docker Swarm.
     - `macvlan`: Assigns a MAC address to containers, making them appear as physical devices on the network.
2. **How do you optimize Docker images to reduce size?**
   - To optimize Docker images:
     - Use multi-stage builds to keep only necessary artifacts.
     - Minimize the number of layers by combining commands.
     - Use a smaller base image (like `alpine`).
     - Clean up unnecessary files and caches.
3. **What are some best practices for securing Docker containers?**
   - Best practices include:
     - Use minimal base images.
     - Run containers as non-root users.
     - Use Docker Content Trust (DCT) for image signing.
     - Regularly scan images for vulnerabilities.
     - Limit container capabilities with security options like `--cap-drop`.
4. **Describe the process of deploying a Dockerized application to a cloud provider.**
   - Steps to deploy a Dockerized application to a cloud provider:
     - Build the Docker image.
     - Push the image to a Docker registry (like Docker Hub or AWS ECR).
     - Use the cloud provider's container service (like AWS ECS, Google Kubernetes Engine, or Azure Container Instances) to deploy the image.
     - Configure necessary networking, scaling, and monitoring.

Terraform

Basic

1. **What is Terraform and what problem does it solve?**
   - Terraform is an open-source infrastructure as code (IaC) tool that allows you to define and provision infrastructure using a high-level configuration language. It solves the problem of managing and automating infrastructure

provisioning across multiple cloud providers.

2. **What is a Terraform provider?**
   - A Terraform provider is a plugin that enables Terraform to manage resources on a specific platform (e.g., AWS, Azure, Google Cloud). Providers are responsible for understanding API interactions and exposing resources.
3. **What is the purpose of the terraform init command?**
   - The `terraform init` command initializes a Terraform working directory. It downloads the necessary provider plugins and sets up the backend configuration for storing the Terraform state.
   - **In simple terms, `terraform init` initializes a Terraform working directory by setting up necessary components that Terraform needs to manage your infrastructure.**

     By running `terraform init`, you ensure that your Terraform environment is properly set up and ready to apply changes to your infrastructure using `terraform apply`

Intermediate

1. **Explain the difference between terraform plan and terraform apply.**
   - `terraform plan`: Creates an execution plan, showing what actions Terraform will take to achieve the desired state defined in the configuration files.
   - `terraform apply`: Executes the actions proposed in the plan to create, update, or delete resources to reach the desired state.
2. **How does Terraform manage state?**
   - Terraform uses a state file (`terraform.tfstate`) to keep track of the resources it manages. This state file is critical for mapping the real-world infrastructure to the configuration. It can be stored locally or remotely (e.g., in AWS S3).
3. **What are Terraform modules and how are they used?**
   - Terraform modules are reusable packages of Terraform configurations that can be used to create consistent infrastructure components. They are used to organize and encapsulate code, making it easier to manage and share.

Advanced

1. **How can you use Terraform to manage multiple environments?**
   - You can manage multiple environments by:
     - Using separate state files for each environment.
     - Organizing configurations into environment-specific directories.
     - Using Terraform workspaces to switch between different states within a single configuration.
2. **Explain how to handle secrets in Terraform.**
   - Secrets can be handled in Terraform using:
     - Environment variables.
     - Encrypted storage solutions like HashiCorp Vault or AWS Secrets Manager.
     - Terraform providers that integrate with secret management systems.

    - Keeping secrets out of configuration files and state files by using data sources or external services.

3. **How do you use Terraform with CI/CD pipelines?**
    - Terraform can be integrated with CI/CD pipelines by:
        - Automating the execution of `terraform init`, `terraform plan`, and `terraform apply` commands.
        - Using version control triggers to run Terraform on code changes.
        - Storing the Terraform state remotely to maintain consistency across pipeline runs.
        - Incorporating approval steps for infrastructure changes.

4. **Discuss the concept of drift detection in Terraform.**
    - Drift detection in Terraform involves identifying changes that have occurred in the infrastructure outside of Terraform's control. This can be done by running `terraform plan` to compare the current state with the desired state defined in the configuration, and noting any discrepancies (drift) that need to be addressed.