Docker

1 **Run a basic container:**
    - Pull the `nginx` image from Docker Hub and run a container exposing it on port 8080.
    - Verify it's running by visiting `http://localhost:8080`.
2. **List containers:**
    - Start a `hello-world` container and list all running containers.
    - Then, list all containers (including stopped ones).
3. **Inspect container details:**
    - Run a `busybox` container in detached mode and inspect its configuration.
4. **Stop and remove a container:**
    - Stop the running `nginx` container and remove it.
5. **Create and use a Dockerfile:**
    - Write a Dockerfile to create a custom image based on `ubuntu`, install `curl` in it, and run a container to execute `curl https://example.com`.
6. **Mount a volume:**
    - Run an `nginx` container and mount a local directory into `/usr/share/nginx/html` to serve custom HTML files.
7. **Pull and tag images:**
    - Pull the `alpine` image, tag it as `my-alpine:latest`, and push it to your Docker Hub repository.
8. **Build and run a custom image:**
    - Create a custom `Dockerfile` that installs Python and runs a simple Python script.
9. **Use environment variables:**
    - Run a MySQL container using the official image, passing environment variables for the root password and database name.
10. **Docker Compose for multiple containers:**
    - Write a `docker-compose.yml` file to run a WordPress site with a MySQL database.
11. **Networking:**
    - Create a custom Docker network and run two containers (`nginx` and `alpine`) that can communicate with each other on the same network.
12. **Persist data using volumes:**
    - Create a volume and attach it to a `mysql` container to persist the database data.
13. **Inspect logs:**
    - Run a `httpd` container and check the container logs for incoming requests.
14. **Scaling services:**
    - Use Docker Compose to scale a simple Python Flask application to 3 replicas.
15. **Use `.dockerignore`:**

- Create a `.dockerignore` file to exclude files like `.git` and `README.md` from your Docker build context.

16. **Optimize Docker images:**
    - Create a multi-stage Dockerfile to build and run a Node.js application while keeping the final image lightweight.
17. **Health checks:**
    - Add a health check to a Docker container running an `nginx` server.
18. **Docker Swarm:**
    - Initialize a Docker Swarm cluster and deploy a service with 3 replicas.
19. **Update services in Swarm:**
    - Deploy a service using Docker Swarm, then update its image to a newer version.
20. **Bind mounts:**
    - Use a bind mount to map a local configuration file into an `nginx` container and update the container settings dynamically.
21. **Build a custom network:**
    - Create a bridge network and deploy three containers (`frontend`, `backend`, and `database`) that communicate only within this network.
22. **Troubleshoot failing containers:**
    - Run a misconfigured container (e.g., a database container with incorrect environment variables) and debug why it fails.
23. **Use Docker labels:**
    - Add labels to a Docker image for versioning and application metadata.
24. **Push and pull private images:**
    - Log in to Docker Hub, tag an image, push it to your private repository, and pull it back.
25. **Security practices:**
    - Run a `nginx` container with minimal privileges using the `--user` flag.
26. **Run in detached mode:**
    - Run a `redis` container in detached mode and verify it is running.
27. **Inspect container resource usage:**
    - Run a `postgres` container and monitor its CPU and memory usage.
28. **Deploy a multi-container app:**
    - Use Docker Compose to deploy a React front-end application connected to a Node.js back-end with a MongoDB database.
29. **Version rollback:**
    - Deploy a Docker service with a specific image version, update it to a newer version, and then roll back to the previous one.
30. **Deploy using secrets:**
    - Deploy a containerized application in Swarm mode using Docker secrets to manage sensitive data like database passwords.

Kubernetes

1. **Create a cluster:**
    - Set up a Kubernetes cluster using Minikube or kind.
2. **Verify cluster status:**
    - Use `kubectl` to check the status of the cluster and nodes.
3. **Run a pod:**
    - Create a simple pod running the `nginx` image.
4. **List pods:**
    - Use `kubectl` commands to list all running pods in the default namespace.
5. **Inspect a pod:**
    - Inspect the details of the `nginx` pod you created.
6. **Expose a pod:**
    - Expose the `nginx` pod as a service on port 80.
7. **Create a namespace:**
    - Create a namespace called `dev` and verify it exists.
8. **Deploy an application:**
    - Create a deployment for an `httpd` server with 2 replicas.
9. **Check logs:**
    - View the logs of the `nginx` pod.
10. **Delete a pod:**
    - Delete the `nginx` pod and ensure it's removed from the cluster.
11. **Scale a deployment:**
    - Scale the `httpd` deployment to 5 replicas.
12. **Access a service:**
    - Access the `nginx` service using `kubectl port-forward`.
13. **Update a deployment:**
    - Update the `nginx` deployment to use a newer image version.
14. **Roll back a deployment:**
    - Roll back the `nginx` deployment to the previous image version.
15. **Get cluster information:**
    - Use `kubectl cluster-info` to get details about your cluster.
16. **Create a ConfigMap:**
    - Create a ConfigMap with environment variables for an application.
17. **Use a ConfigMap in a pod:**
    - Mount the ConfigMap as environment variables in a pod.
18. **Create a Secret:**
    - Store a database password in a Kubernetes Secret.
19. **Use a Secret in a pod:**
    - Mount the Secret as an environment variable in a pod.
20. **Check pod resource usage:**
    - Monitor CPU and memory usage of a specific pod using `kubectl top`.
21. **Add resource limits:**

- Define CPU and memory limits for a deployment.

22. **Create a PersistentVolume:**
    - Create a PersistentVolume backed by a local storage directory.
23. **Use a PersistentVolumeClaim:**
    - Attach a PersistentVolume to a pod using a PersistentVolumeClaim.
24. **Deploy with Helm:**
    - Use Helm to install the `nginx` chart.
25. **Debug a pod:**
    - Use `kubectl exec` to troubleshoot issues inside a running pod.
26. **View pod events:**
    - Use `kubectl describe` to inspect events associated with a failing pod.
27. **Create a Job:**
    - Create a Kubernetes Job that runs a one-time script to process data.
28. **Create a CronJob:**
    - Schedule a task to run every 5 minutes using a Kubernetes CronJob.
29. **Work with labels:**
    - Label a pod and use `kubectl` to list pods with that specific label.
30. **Annotate a pod:**
    - Add an annotation to a running pod and verify it.
31. **Set up Node affinity:**
    - Deploy a pod that can only run on a specific node.
32. **Use taints and tolerations:**
    - Taint a node and deploy a pod that tolerates the taint.
33. **Create a custom health check:**
    - Add liveness and readiness probes to an `nginx` deployment.
34. **Autoscale a deployment:**
    - Use the Horizontal Pod Autoscaler (HPA) to scale the `httpd` deployment based on CPU usage.
35. **Enable RBAC:**
    - Create a Role and RoleBinding to grant permissions to a specific user in a namespace.
36. **Deploy a StatefulSet:**
    - Create a StatefulSet for a MySQL database with a PersistentVolumeClaim.
37. **Set up Ingress:**
    - Create an Ingress resource to route traffic to multiple services.
38. **Use a NetworkPolicy:**
    - Create a NetworkPolicy to restrict traffic between pods.
39. **Deploy with a custom scheduler:**
    - Configure and use a custom Kubernetes scheduler.
40. **Backup and restore etcd:**
    - Take a backup of the etcd database and restore it.
41. **Monitor your cluster:**
    - Set up Prometheus and Grafana for monitoring your cluster.
42. **Log aggregation:**
    - Deploy an EFK (Elasticsearch, Fluentd, Kibana) stack for centralized logging.
43. **Configure Pod Disruption Budget:**
    - Create a Pod Disruption Budget to maintain high availability for a deployment.

44. **Run a DaemonSet:**
    - Create a DaemonSet to deploy a log collector on every node.
45. **Set up kube-proxy metrics:**
    - Enable and monitor kube-proxy metrics for networking diagnostics.
46. **Use Admission Controllers:**
    - Set up a Mutating or Validating webhook to enforce policies.
47. **Create an Operator:**
    - Develop a basic Kubernetes Operator using the Operator SDK.
48. **Enable multi-cluster management:**
    - Set up a federation to manage multiple Kubernetes clusters.
49. **Use Kustomize:**
    - Create an environment-specific configuration for an application using Kustomize.
50. **Secure your cluster:**
    - Implement Pod Security Policies and configure node-level isolation.

GitHub/BitBucket

1. **Create a repository:**
    - Create a new repository named `my-first-repo`.
2. **Clone a repository:**
    - Clone a public repository to your local system using `git clone`.
3. **Add files to a repository:**
    - Create a file named `README.md`, add it to your local repository, and push it to GitHub.
4. **Commit changes:**
    - Modify the `README.md` file and commit the changes with an appropriate message.
5. **Push changes:**
    - Push the committed changes to the remote GitHub repository.
6. **Fork a repository:**
    - Fork a public repository and list it under your account.
7. **Create a branch:**
    - Create a new branch named `feature/update-readme`.
8. **Switch branches:**
    - Switch to the `feature/update-readme` branch and confirm your branch using `git branch`.
9. **Merge branches:**
    - Merge the `feature/update-readme` branch into the `main` branch.
10. **Delete a branch:**
    - Delete the `feature/update-readme` branch locally and on GitHub.
11. **Work with pull requests:**

- Create a pull request to merge changes from a feature branch to the `main` branch.

12. **Resolve merge conflicts:**
    - Simulate a merge conflict by editing the same lines in two branches and resolving the conflict during the merge.
13. **Use GitHub Issues:**
    - Create an issue in your repository describing a bug or feature request.
14. **Close an issue:**
    - Commit a change and link it to the issue by using `Fixes #issue_number` in the commit message.
15. **Add collaborators:**
    - Add a collaborator to your repository with write access.
16. **Create a release:**
    - Tag a version of your repository and create a release for it.
17. **Work with labels:**
    - Add labels like `bug` or `enhancement` to an issue in your repository.
18. **Set up a GitHub Action:**
    - Add a workflow file to automate tests for your repository using GitHub Actions.
19. **Work with milestones:**
    - Create a milestone and assign issues to it.
20. **Use GitHub Discussions:**
    - Start a discussion in your repository to gather feedback on a new feature.
21. **Enable branch protection rules:**
    - Protect the `main` branch by requiring pull request reviews before merging.
22. **Create a CODEOWNERS file:**
    - Add a `CODEOWNERS` file to define who reviews pull requests for specific files.
23. **Use GitHub Pages:**
    - Deploy a static website using GitHub Pages from a repository.
24. **Automate releases:**
    - Use a GitHub Action to automatically create a release when a new tag is pushed.
25. **Use secrets:**
    - Add a secret in your repository settings and use it in a GitHub Actions workflow.
26. **Work with submodules:**
    - Add another repository as a submodule to your project and demonstrate updating it.
27. **Enable Dependabot:**
    - Enable Dependabot to automatically check for outdated dependencies in your repository.
28. **Set up a project board:**
    - Create a project board and add issues or pull requests to it.
29. **Monitor repository insights:**
    - Explore and analyze repository insights, such as traffic, contributions, and commits.
30. **Configure repository webhooks:**

- Add a webhook to your repository to trigger a custom action when a push event occurs.

Jenkins:

1. **Install Jenkins:**
   - Install Jenkins on your local system or a server.
2. **Create a new job:**
   - Create a freestyle project in Jenkins.
3. **Run a simple job:**
   - Configure the job to print "Hello, World!" in the build logs.
4. **Check Jenkins logs:**
   - View the build logs of the executed job.
5. **Install plugins:**
   - Install the "Git Plugin" using the Jenkins Plugin Manager.
6. **Configure system settings:**
   - Set up the Jenkins system email address in the global configuration.
7. **Integrate GitHub:**
   - Add a GitHub repository URL to a freestyle job.
8. **Clone and build a GitHub repository:**
   - Configure a job to clone a GitHub repository and display its contents in the build logs.
9. **Schedule a job:**
   - Use the Jenkins scheduler (CRON syntax) to run a job every 5 minutes.
10. **Add build parameters:**
    - Configure a job with a string parameter and echo its value in the build logs.
11. **Trigger a build remotely:**
    - Set up a job that can be triggered using a URL with a token.
12. **Set up notifications:**
    - Configure email notifications to send build results to your email address.
13. **Set up build retention:**
    - Configure a job to keep only the last 5 build logs.
14. **Use Post-Build Actions:**
    - Configure a job to archive artifacts after the build.
15. **Configure Jenkins slaves:**
    - Add a Jenkins slave node and run a job on the slave.
16. **Pipeline basics:**
    - Create a simple pipeline job that prints "Pipeline Execution Started."
17. **Use Jenkins credentials:**
    - Add SSH credentials to Jenkins and use them in a job.
18. **Integrate with Maven:**
    - Configure a Maven job in Jenkins to build a Java project.
19. **Execute shell commands:**
    - Add a shell script step in a freestyle job to create and display a file.
20. **Git polling:**

    - Configure a job to poll a GitHub repository for changes and trigger builds automatically.

21. **Create a declarative pipeline:**
    - Write a pipeline script to clone a GitHub repository and list its files.
22. **Parallel stages:**
    - Create a pipeline job with two parallel stages executing different shell commands.
23. **Parameterized builds:**
    - Configure a pipeline that accepts a string parameter and echoes its value in the logs.
24. **Blue Ocean:**
    - Install the Blue Ocean plugin and create a pipeline using its GUI.
25. **Archive artifacts:**
    - Create a job to build and archive a `.zip` or `.tar` file as an artifact.
26. **Deploy to a server:**
    - Create a pipeline job that deploys a built artifact to a remote server using SCP.
27. **Integrate Jenkins with Docker:**
    - Configure Jenkins to build and run a Docker container.
28. **Pipeline libraries:**
    - Create a shared library and use it in a pipeline.
29. **Set up a multibranch pipeline:**
    - Configure a multibranch pipeline to build all branches in a Git repository.
30. **Jenkinsfile from repository:**
    - Create a job that uses a `Jenkinsfile` stored in a GitHub repository.
31. **Post-build actions in pipelines:**
    - Add post-build actions in a pipeline to send notifications upon failure.
32. **Test reports:**
    - Integrate a pipeline with JUnit to publish test reports.
33. **Configure webhooks:**
    - Set up a GitHub webhook to trigger a Jenkins job upon repository updates.
34. **Environment variables:**
    - Use and display environment variables in a pipeline.
35. **Security settings:**
    - Configure matrix-based security for users and roles in Jenkins.
36. **Integrate Jenkins with Kubernetes:**
    - Configure a pipeline to deploy a Docker container to a Kubernetes cluster.
37. **Create a backup:**
    - Use the ThinBackup plugin to create a backup of Jenkins configurations.
38. **Integrate with Slack:**
    - Set up Slack notifications for job statuses.
39. **Conditional stages:**
    - Write a pipeline with conditional stages that run based on the branch name.
40. **Jenkins performance monitoring:**
    - Install and configure the Monitoring plugin to observe Jenkins resource usage.