1. **What is the difference between Kubernetes and OpenShift?**

Kubernetes

- **Open-source** container orchestration platform.
- Originally developed by **Google**, now maintained by the **Cloud Native Computing Foundation (CNCF)**.
- Highly flexible and customizable, but requires significant manual configuration for production use (like networking, logging, security, etc.).
- Lacks a built-in web console (but third-party dashboards like Lens can be added).
- No built-in CI/CD or image registry.
- You need to manage RBAC, ingress controllers, monitoring tools (like Prometheus/Grafana), etc., separately.

OpenShift (OpenShift Container Platform)

- **Enterprise Kubernetes distribution** developed by **Red Hat**.
- Built **on top of Kubernetes**, adds many enterprise-ready tools out of the box.
- Comes with:
    - A built-in **web console** for managing applications and resources.
    - **Integrated CI/CD** via Jenkins or Tekton pipelines.
    - **Built-in container registry** and image streams.
    - Enhanced **RBAC and security features** (e.g., Security Context Constraints - SCC).
    - **Red Hat support**, certified images, and operator lifecycle management.
- More opinionated, with stricter defaults — e.g., non-root containers by default for better security.

Feature	Kubernetes	OpenShift
Base	CNCF Kubernetes	Kubernetes with enterprise tools
Web UI	Not included (Dashboard optional)	Built-in, user-friendly web console
CI/CD	Manual integration	Built-in via Jenkins or Tekton
Image Registry	Not included	Built-in container image registry
Security Defaults	Flexible, less restrictive	Enforces strict security (e.g., non-root)
Enterprise Support	Community-based	Red Hat commercial support available
Installation Complexity	DIY, highly customizable	Streamlined with Red Hat tools

2. Can you explain the architecture of an OpenShift cluster? What are master and node components?

OpenShift Cluster Architecture Overview

- An OpenShift cluster consists of several **core components** running across different types of machines:
- **Control Plane (Master Nodes)**
- **Worker Nodes (Node Components)**
- **Infrastructure Nodes** (optional — for routers, monitoring, logging, etc.)

3. How do you manage deployments in OpenShift?

OpenShift supports strategies like Rolling or Recreate to control how pods are updated. You can also use health checks (readiness and liveness probes), ConfigMaps, Secrets, and ImageStreams to manage and automate deployments.

4. What's the process to scale applications in OpenShift?

To scale applications in OpenShift, you can increase or decrease the number of pod replicas using the oc scale command or the web console. For example, oc scale deployment/my-app --replicas=3 will scale your app to 3 pods. You can also set up **Horizontal Pod Autoscalers (HPA)** to automatically scale based on CPU or memory usage.

5. How do you troubleshoot a pod stuck in CrashLoopBackOff?

Check pod logs to see the error:
oc logs pod-name

- **Describe the pod** to get event details:
  oc describe pod pod-name
- **Check container exit code and reason** — often due to config errors, missing files, or failed startup scripts.
- **Verify environment variables, ConfigMaps, Secrets, and mounted volumes**.
- **Check readiness/liveness probes** — incorrect probes can cause restarts.

6. What's the difference between Routes and Services in OpenShift?

- A **Service** exposes a set of pods internally within the cluster using a stable DNS name and IP. It enables communication between pods.
- A **Route** exposes a Service **externally** to users outside the cluster by assigning a public URL. It lets users access your application over HTTP/HTTPS.

Service = internal access
Route = external access

7. How do you configure persistent storage in OpenShift?

To configure persistent storage in OpenShift:

1. **Create a PersistentVolume (PV)** – defines the actual storage (e.g., NFS, AWS EBS, Ceph).

2. **Create a PersistentVolumeClaim (PVC)** – the app requests storage by claiming a PV.
3. **Mount the PVC to a pod** in your Deployment or DeploymentConfig.

⚙ CI/CD Pipeline & Automation

1. **How have you implemented CI/CD pipelines with Jenkins and OpenShift?**

   I've implemented CI/CD pipelines in OpenShift using Jenkins by integrating it with OpenShift's native pipeline support. I used Jenkinsfiles (written in Groovy) to define the stages like checkout, build, test, and deploy.

- Pull code from GitHub/GitLab.
- Build Docker images using OpenShift's BuildConfig or Jenkins pipeline scripts.
- Push the image to the internal OpenShift registry.
- Deploy the image using oc rollout or update a DeploymentConfig.

2. **Can you explain how Git, Jenkins, and Ansible work together in a deployment pipeline?**

- **Git** stores the application code and configuration (like playbooks or deployment scripts).
- **Jenkins** acts as the automation server. It pulls code from Git (using a webhook or poll), builds the application, runs tests, and triggers deployment.
- **Ansible** is used by Jenkins to automate the deployment process—such as configuring servers, installing packages, or deploying the app to environments like staging or production.

3. **How would you automate the deployment of a containerized app into OpenShift?**

1) **Containerize the app**:
    a) Create a Docker image (or use Source-to-Image in OpenShift).
2) **Create a BuildConfig**:

    a) Set up a BuildConfig in OpenShift to automatically build your container image from source.
3) **Create a DeploymentConfig**:
    a) Define how your app should be deployed (number of replicas, update strategies, etc.).
4) **Set up Jenkins for CI/CD**:
    a) Configure Jenkins to pull code from Git and trigger a build.
    b) Use Jenkins to deploy the built image to OpenShift.
5) **Monitor and Automate Rollouts**:
    a) Use **ImageStreams** to track new versions of your container and trigger automatic deployments.
6) **Add Health Checks**:
    a) Implement readiness and liveness probes to ensure successful app deployments.

4. What strategies do you use to ensure zero-downtime deployments?

- **Use Rolling Deployments**:
- Ensure OpenShift gradually replaces old pods with new ones, ensuring availability during updates.
- **Configure Readiness and Liveness Probes**:
- Readiness probes ensure only healthy pods receive traffic.
- Liveness probes automatically restart unhealthy pods.
- **Deploy Multiple Replicas**:
- Run multiple replicas of your app to maintain availability during deployments.
- **Use Blue-Green Deployments**:
- Keep two environments (blue for production, green for the new version) and switch traffic to the new version once it's stable.
- **Use Canary Deployments**:
- Deploy the new version to a small subset of users first, and gradually roll out to the rest.
- **Handle Database Migrations Carefully**:
- Ensure backward-compatible database changes so both old and new versions can work with the schema.

💻 Linux & System Administration

1. **How do you perform kernel-level debugging in Linux?**

- **Check kernel logs**:
  Run dmesg to view kernel messages and errors.
- **Enable crash dump (Kdump)**:
  Set up kdump to collect data after a kernel crash for analysis.
- **Use KGDB (Kernel GDB)**:
  Connect a debugger like gdb to the running kernel for live debugging (usually from another machine).
- **Use Magic SysRq key**:
  Use commands like echo c > /proc/sysrq-trigger to trigger a controlled crash or gather info.
- **Use Kprobes or Ftrace**:
  Insert probes or trace kernel functions to monitor what's happening internally

2. **How do you secure a RedHat Linux system?**

Red Hat Linux Hardening Steps:

1. **Update system** – dnf update
2. **Enable firewall** – use firewalld
3. **Enforce SELinux** – getenforce
4. **Disable unused services** – systemctl disable
5. **Secure SSH** – disable root login, use keys
6. **Set strong passwords** – enforce password policy
7. **Enable audit logs** – configure auditd
8. **Limit user permissions** – use sudo wisely

3. **What are some important performance tuning parameters for Linux servers?**

- **Reduce swap usage**
  sysctl -w vm.swappiness=10
- **Increase open file limit**
  ulimit -n 65535
- **Change I/O scheduler (for SSDs)**
  echo noop > /sys/block/sda/queue/scheduler

- **Optimize network performance**
  sysctl -w net.core.somaxconn=1024
  sysctl -w net.ipv4.tcp_fin_timeout=15
- **Use performance tuning profile**
  tuned-adm profile performance

📊 Monitoring & Logging

1. **How do you use Grafana and Kibana for application monitoring?**

☑ Grafana (for metrics and dashboards):

1. **Connect to a data source.**
2. **Create dashboards** to visualize metrics such as CPU, memory, disk, or app performance.

☑ Kibana (for logs and search):

1. **Connect to Elasticsearch**, which stores logs.
2. **Search and filter logs** to debug issues or analyze trends

2. **What's your experience with Dynatrace? How do you integrate it with OpenShift?**

Dynatrace:

I have used **Dynatrace** for monitoring applications, infrastructure, and Kubernetes/OpenShift clusters. It provides **real-time observability**, automatic root-cause detection, and application performance insights.

Dynatrace + OpenShift Integration (Simple Steps):

1. **Create Dynatrace API Token**
   – Generate it from your Dynatrace account.
2. **Install Dynatrace Operator**
   – From OpenShift OperatorHub or using oc apply.
3. **Create Dynakube YAML**
   – Add your API token and environment details in the YAML.
4. **Apply Dynakube to OpenShift**
   – Run oc apply -f dynakube.yaml.
5. **Monitoring Starts Automatically**
   – Dynatrace OneAgent auto-installs and starts monitoring your apps and cluster.

3. What alerts and metrics do you set up in a production OpenShift cluster?

☐ **Monitor Node Health**:
Track CPU, memory, disk usage, and node status (Ready/Not Ready).

☐ **Track Pod Health**:
Set alerts for pod restarts or CrashLoopBackOff status.

☐ **Monitor Cluster Resources**:
Check available vs requested CPU/memory and scheduling issues.

☐ **Watch Networking Issues**:
Monitor network latency and DNS failures.

☐ **Check Persistent Storage**:
Alert for PVC binding failures or storage issues.

☐ **Track API Server Performance**:
Monitor request latency and error rates (5xx).

Database Integration

1. How have you integrated Oracle DB with containerized applications?

- **Pull Oracle DB Docker Image**

    - Run docker pull container-registry.oracle.com/database/enterprise:19.
- **Run Oracle DB in a Container**
    - Use the command:

- bash
- CopyEdit
- docker run -d -v /path/to/data:/opt/oracle/oradata oracle/database:19

- **Create Persistent Storage**
    - Use **host-mounted volumes** or **Kubernetes Persistent Volumes (PVs)** for data persistence.
- **Connect Application to Oracle DB**
    - Set database connection details (hostname, port, user, password) via environment variables in your app.
- **Use Kubernetes/OpenShift for Integration**
    - Create a **Kubernetes Service** for Oracle DB and connect your app to it.

2. How do you ensure high availability and backup for databases in OpenShift environments?

   High Availability

- **Use StatefulSets**
- Deploy database pods with **StatefulSets** for stable network identities and persistent storage.
- **Database Clustering**
- Set up **database clustering** for multi-master replication and failover.
- **Create a Kubernetes Service**
- Use a **Kubernetes Service** to load balance traffic and ensure clients connect to healthy database instances.
- **Enable Automatic Failover**
- Implement **automatic failover to** automatically promote a healthy replica when the primary fails

Backup Strategy:

1. **Schedule Backups**
    a. Use **OpenShift CronJobs** or **Kubernetes CronJobs** to automate database backups (e.g., daily backups).
2. **Use Persistent Volumes**
    a. Store database data in **Persistent Volumes (PVs)** to ensure data is safe and can be restored if needed.
3. **External Backup Solutions**
    a. Use cloud-based backups (e.g., **AWS RDS**, **Azure Database Backup**) for added security.
4. **Test Backups Regularly**
    a. Regularly restore backups to verify they are recoverable.

Troubleshooting & Production Support

1. Describe a time you had to troubleshoot an outage in production.

- **Gather Information**
    - Check **monitoring tools** (e.g., Grafana, Kibana) for resource usage (CPU, memory).
    - Review **logs** (application, database, system) for errors.
- **Check System Health**
    - Inspect the **status of pods** (e.g., Kubernetes/OpenShift) for failures like **CrashLoopBackOff**.
- **Identify the Cause**
    - Check if the issue is related to **recent deployments** or changes (e.g., code or infrastructure).
    - Verify **database connectivity** and performance.
- **Rollback to Stable Version**
    - If the issue is caused by new changes, **rollback** to the last stable deployment.
- **Resolve and Test**
    - Fix the issue (e.g., code bug, resource limits) and **test** in a staging environment.
    - Redeploy the fixed version and monitor for stability.

- **Implement Preventive Measures**
    - Set up **better monitoring**, **alerts**, and **rollback strategies** to avoid future outages.

2. What logs or tools would you check if an OpenShift node becomes unreachable?

- **Check Node Status**
- Run oc get nodes to see if the node is in Ready or NotReady state.
- **Check Kubelet Logs**
- SSH into the node and run: journalctl -u kubelet
- **Check System Logs**
- Review system logs for errors using: journalctl –xe
- **Test Network Connectivity**
- Use ping or telnet to test connectivity between the node and control plane.
- **Check Docker/Container Runtime Logs**
- Run:journalctl -u docker
- **Check OpenShift Master Logs**
- On the master node, run: journalctl -u origin-master
- **Check for Resource Exhaustion**
- Check memory, CPU, and disk usage: top, df -h, free -m

3. How do you handle pod-to-pod communication failures in a cluster?

- **Check Pod Status**
- Run: oc get pods -o wide
- **Check Network Policies**
- Run: oc get networkpolicy
- **Check Pod Logs**
- Run: oc logs <pod-name>
- **Check DNS Resolution**
- Run: nslookup <pod-name>.<namespace>.svc.cluster.local
- **Check Service Configuration**
- Run: oc get svc <service-name> -o yaml
- **Check Network Connectivity**
- Use ping or curl inside the pod.
- **Check CNI Configuration**
- Verify CNI plugin logs for errors.

- **Check Firewall/Security Groups**
- Ensure no firewall or security group issues.
- **Restart Affected Pods**
- Run: oc delete pod <pod-name>

🔐 Security & Access Control

1. How does Role-Based Access Control (RBAC) work in OpenShift?

- **Define a Role or ClusterRole**
- Role: Permissions in a specific namespace.
- ClusterRole: Permissions across the entire cluster.
- **Create the Role/ClusterRole**
- Example for a Role:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: <role-name>
  namespace: <namespace>
rules:
- resources: ["pods"]
  verbs: ["get", "list"]
```

1. **Create RoleBinding or ClusterRoleBinding**
    a. RoleBinding: Bind Role to a user/service account in a specific namespace.
    b. ClusterRoleBinding: Bind ClusterRole to a user/service account across the cluster.
2. **Grant Permissions**
    a. Specify allowed **verbs** (e.g., get, create, delete) for resources (e.g., pods, deployments).
3. **Access Control**
    a. OpenShift checks the **RoleBinding** or **ClusterRoleBinding** to allow/disallow actions based on the role.

2. How do you secure communication between services in OpenShift?

- **Use Network Policies**
- Define Network Policies to control traffic flow between services.
- **Use TLS Encryption**
- Enable TLS encryption for secure service communication.
- **Use Service Accounts with Least Privilege**
- Assign minimal permissions to service accounts for access control.
- **Use OpenShift's Internal DNS**
- Use OpenShift's internal DNS for secure communication between services.
- **Use OpenShift Service Mesh (Optional)**
- Implement a Service Mesh (like Istio) for automatic encryption and policy enforcement.
- **Audit and Monitor Traffic**
- Use audit logs and monitoring tools to track and secure service interactions.