KUBERNETES

INTERVIEW QUESTIONS & ANSWERS

+100

- Core concepts & architecture
- Networking intricacies & best practices
- Deployment strategies & advanced scaling
- Security deep dives (RBAC, Secrets, Policies)
- Real-world troubleshooting scenarios
- Ecosystem tools & GitOps principles

I. Kubernetes Fundamentals and Architecture

1. **What is Kubernetes and why is it used?**
    - **Answer:** Kubernetes (K8s) is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It solves the challenges of managing large, distributed applications across multiple servers by providing features like automated scheduling, self-healing, rolling updates, and service discovery.
2. **Explain the core components of Kubernetes architecture.**
    - **Answer:** Kubernetes architecture consists of two main parts:
        - **Control Plane (Master Node):**
            - **Kube-apiserver:** The front-end of the Kubernetes control plane, exposing the Kubernetes API. All communication with the cluster goes through the API server.
            - **etcd:** A highly available, distributed key-value store that stores all cluster data (configurations, state, metadata).
            - **Kube-scheduler:** Selects a node for newly created Pods based on resource requirements, constraints, and other policies.
            - **Kube-controller-manager:** Runs controller processes that regulate the cluster's state (e.g., Node Controller, Replication Controller, Endpoints Controller, Service Account & Token Controllers).
        - **Worker Nodes:**
            - **Kubelet:** An agent that runs on each node in the cluster, ensuring containers are running in a Pod. It communicates with the control plane.
            - **Kube-proxy:** A network proxy that runs on each node, maintaining network rules on nodes and enabling network communication to your Pods from inside or outside of the cluster.
            - **Container Runtime (e.g., containerd, CRI-O, Docker):** The software responsible for running containers.
3. **What is a Pod in Kubernetes? Why is it the smallest deployable unit?**

- **Answer:** A Pod is the smallest deployable unit in Kubernetes. It represents a single instance of a running process in your cluster. It's the smallest unit because containers within a Pod share the same network namespace, IP address, and storage, making them tightly coupled and allowing them to communicate via localhost.

4. **What is the relationship between Kubernetes and Docker?**
    - **Answer:** Docker is a containerization platform that enables you to package and run applications in isolated environments called containers. Kubernetes is a container orchestration platform that *manages* and *automates* the deployment, scaling, and operation of Docker containers (or any other OCI-compliant container runtime) across a cluster of machines. Docker builds the house (container), Kubernetes manages the neighborhood (cluster).
5. **Explain Namespaces in Kubernetes. When would you use them?**
    - **Answer:** Namespaces provide a mechanism for isolating groups of resources within a single Kubernetes cluster. They are essentially virtual clusters within a physical cluster. You would use them for:
        - **Resource isolation:** Separating resources for different teams, projects, or environments (e.g., dev, staging, prod).
        - **Access control:** Applying RBAC policies at a namespace level to restrict user access.
        - **Resource quotas:** Setting resource limits (CPU, memory) per namespace.
6. **What are Labels and Selectors in Kubernetes?**
    - **Answer:**
        - **Labels:** Key-value pairs that are attached to Kubernetes objects (e.g., Pods, Services, Deployments). They are used to organize, identify, and categorize objects.
        - **Selectors:** Used to filter objects based on their labels. Services use selectors to identify the Pods they should route traffic to, and Deployments use them to manage their associated Pods.
7. **What is a Service in Kubernetes? Why is it needed?**
    - **Answer:** A Service is an abstraction that defines a logical set of Pods and a policy by which to access them. It provides a stable network endpoint (a fixed IP address and DNS name) for a group of Pods, even as Pods are

created, deleted, or rescheduled. This is crucial because Pods are ephemeral and their IP addresses can change.

8. **Differentiate between ClusterIP, NodePort, and LoadBalancer Service types.**
    - **Answer:**
        - **ClusterIP:** Exposes the Service on an internal IP in the cluster. It's only reachable from within the cluster. Default type.
        - **NodePort:** Exposes the Service on each Node's IP at a static port (the NodePort). Makes the Service accessible from outside the cluster by NodeIP:NodePort.
        - **LoadBalancer:** Exposes the Service externally using a cloud provider's load balancer. This type automatically provisions a load balancer and routes external traffic to your Service. (Requires cloud provider integration).
        - **ExternalName:** Maps the Service to a DNS name, not a cluster IP or NodePort. Used for services external to the cluster.
9. **What is a Deployment in Kubernetes?**
    - **Answer:** A Deployment is a higher-level abstraction that manages the lifecycle of Pods and ReplicaSets. It provides declarative updates to Pods, ensuring that the desired number of replicas are running and facilitating features like rolling updates, rollbacks, and self-healing.
10. **Explain the difference between a ReplicaSet and a Deployment.**
    - **Answer:**
        - **ReplicaSet:** Ensures a specified number of identical Pod replicas are running at all times. It's a low-level controller primarily used by Deployments.
        - **Deployment:** A higher-level object that manages ReplicaSets. It provides declarative updates to Pods and supports features like rolling updates, rollbacks, and versioning, making it the preferred way to manage stateless applications.
11. **What is a StatefulSet and when would you use it?**
    - **Answer:** A StatefulSet is a workload API object used to manage stateful applications. Unlike Deployments, StatefulSets provide:
        - **Stable, unique network identifiers:** Each Pod gets a stable hostname.

        - **Stable, persistent storage:** PersistentVolumes are provisioned for each Pod.
        - **Ordered, graceful deployment and scaling:** Pods are created and terminated in a defined order.
        - **Ordered, graceful deletion:** Pods are deleted in reverse ordinal order.
    - You'd use StatefulSets for applications requiring persistent storage, stable network identities, or ordered scaling, such as databases (e.g., MySQL, PostgreSQL), message queues (e.g., Kafka), or other stateful services.
12. **What is a DaemonSet and when would you use it?**
    - **Answer:** A DaemonSet ensures that a copy of a Pod runs on *all* (or a subset of) nodes in a cluster. They are typically used for cluster-level functionalities like:
        - **Logging agents:** (e.g., Fluentd, Logstash)
        - **Monitoring agents:** (e.g., Prometheus Node Exporter)
        - **Storage daemon:** (e.g., Ceph)
13. **Explain ConfigMaps and Secrets. What's the difference?**
    - **Answer:**
        - **ConfigMaps:** Used to store non-sensitive configuration data as key-value pairs. They allow you to decouple configuration from application code, making it easier to manage and update.
        - **Secrets:** Similar to ConfigMaps but designed for storing sensitive data like passwords, API keys, and tokens. Kubernetes handles Secrets more securely by encoding them (Base64) and providing mechanisms for secure distribution and access.
    - **Difference:** Sensitivity of data and how they are handled securely.
14. **What is Ingress in Kubernetes? How does it differ from a LoadBalancer Service?**
    - **Answer:**
        - **Ingress:** An API object that manages external access to services within the cluster, typically HTTP/HTTPS traffic. It provides features like URL-based routing, name-based virtual hosting, and SSL/TLS

termination. An Ingress resource requires an Ingress Controller (e.g., Nginx Ingress Controller, Traefik) to function.

- **LoadBalancer Service:** Provisions a cloud provider's load balancer to expose a Service. It works at the TCP/UDP layer.

- **Difference:** Ingress operates at Layer 7 (Application Layer) and offers more advanced routing rules based on hostname and path, while a LoadBalancer Service operates at Layer 4 (Transport Layer) and provides a simple way to expose a Service externally.

15. **How does Kubernetes handle storage orchestration?**
    - **Answer:** Kubernetes provides a flexible storage orchestration framework using:
        - **PersistentVolume (PV):** A cluster-wide resource representing a piece of networked storage provisioned by an administrator or dynamically provisioned.
        - **PersistentVolumeClaim (PVC):** A request for storage by a user, specifying the size and access mode. Kubernetes matches PVCs to available PVs.
        - **StorageClass:** Defines different classes of storage (e.g., "fast", "slow") and their provisioner. This allows for dynamic provisioning of PVs when a PVC requests a specific StorageClass.

II. Networking in Kubernetes

16. **Explain the Kubernetes networking model.**
    - **Answer:** Kubernetes enforces a "flat" networking model where:
        - All Pods can communicate with all other Pods without NAT.
        - All Nodes can communicate with all Pods without NAT.
        - The IP that a Pod sees itself as is the same IP that others see it as.
    - This model relies on a Container Network Interface (CNI) plugin (e.g., Calico, Flannel, Cilium) to implement the actual network fabric.
17. **How does Pod-to-Pod communication work within the same node?**
    - **Answer:** Pods on the same node communicate via the CNI bridge. The CNI plugin creates a virtual Ethernet pair for each Pod, with one end in the

Pod's network namespace and the other connected to a bridge on the node.

18. **How does Pod-to-Pod communication work across different nodes?**
    - **Answer:** Pods across different nodes communicate via the overlay network created by the CNI plugin. The CNI plugin encapsulates Pod traffic (e.g., using VXLAN, IPIP) and routes it between nodes.
19. **What are Network Policies in Kubernetes? When would you use them?**
    - **Answer:** Network Policies are Kubernetes resources that define rules for how Pods are allowed to communicate with each other and with other network endpoints. They provide network segmentation and security by restricting traffic flow. You'd use them for:
        - Isolating different application tiers (e.g., frontend, backend, database).
        - Restricting access to sensitive services.
        - Implementing micro-segmentation within your cluster.
20. **How does Kubernetes handle DNS for services and pods?**
    - **Answer:** Kubernetes uses an internal DNS service (typically CoreDNS) to provide service discovery.
        - **Pods:** Each Pod gets a DNS A record in the form pod-ip-address.namespace.pod.cluster.local.
        - **Services:** Each Service gets a DNS A record in the form service-name.namespace.svc.cluster.local. Headless services also get A records for their individual Pods.

III. Deployment and Scaling

21. **Describe the process of a rolling update in Kubernetes.**
    - **Answer:** A rolling update gradually updates application instances with a new version without downtime. When you update a Deployment, Kubernetes creates new Pods with the new image, waits for them to become healthy, and then slowly terminates old Pods. This ensures continuous availability during the update.
22. **How do you perform a rollback of a Deployment?**

- **Answer:** You can use kubectl rollout undo deployment <deployment-name>. You can also specify a specific revision to roll back to using --to-revision=<revision-number>.

23. **What is Horizontal Pod Autoscaling (HPA)? How does it work?**
    - **Answer:** HPA automatically scales the number of Pod replicas in a Deployment or ReplicaSet based on observed CPU utilization, memory utilization, or custom metrics. It periodically checks the metrics and adjusts the replicas field of the target resource.
24. **What is Vertical Pod Autoscaling (VPA)?**
    - **Answer:** VPA automatically adjusts the CPU and memory requests and limits for containers in a Pod based on historical usage. It helps optimize resource allocation and prevent over-provisioning or under-provisioning. (Note: VPA is still in beta and has implications for Pod rescheduling.)
25. **What is Cluster Autoscaler?**
    - **Answer:** Cluster Autoscaler automatically adjusts the number of nodes in your Kubernetes cluster based on resource requests and actual usage. If Pods cannot be scheduled due to insufficient resources, it adds new nodes. If nodes are underutilized, it removes them.
26. **How would you troubleshoot a failed Deployment?**
    - **Answer:**
        1. kubectl get deployments (check status)
        2. kubectl describe deployment <deployment-name> (check events, conditions, replica status)
        3. kubectl get replicasets (check associated ReplicaSets)
        4. kubectl get pods -l app=<label> (check Pod status)
        5. kubectl describe pod <pod-name> (check events, container status, volumes)
        6. kubectl logs <pod-name> (check application logs)
        7. kubectl exec -it <pod-name> -- /bin/sh (debug inside the container)
        8. Check kube-apiserver and kube-scheduler logs if no Pods are scheduling.
        9. Check kubelet logs on the node if Pods are stuck in a pending state.

27. **Explain Liveness and Readiness Probes. Why are they important?**
    - **Answer:**
        - **Liveness Probe:** Determines if a container is running and healthy. If the liveness probe fails, Kubernetes will restart the container. This prevents deadlocked containers from perpetually consuming resources.
        - **Readiness Probe:** Determines if a container is ready to serve traffic. If the readiness probe fails, Kubernetes will remove the Pod's IP from the Service endpoints, preventing traffic from being routed to an unready Pod. This ensures no traffic is sent to applications that are still starting up or experiencing issues.
    - **Importance:** They enable Kubernetes to self-heal and manage application availability and reliability.
28. **What is a CronJob in Kubernetes? Give an example use case.**
    - **Answer:** A CronJob creates Jobs on a repeating schedule. They are used for performing scheduled tasks.
    - **Example Use Case:** Running a daily database backup, generating weekly reports, or cleaning up old logs.

IV. Security and Access Control

29. **Explain Role-Based Access Control (RBAC) in Kubernetes.**
    - **Answer:** RBAC is a mechanism that allows you to define who (Subjects: Users, Groups, Service Accounts) can do what (Verbs: get, list, create, delete) to which resources (Resources: Pods, Deployments, Services) in which namespaces.
    - **Key components:**
        - **Role:** Defines permissions within a specific namespace.
        - **ClusterRole:** Defines permissions across the entire cluster.
        - **RoleBinding:** Grants the permissions defined in a Role to a Subject within a namespace.
        - **ClusterRoleBinding:** Grants the permissions defined in a ClusterRole to a Subject across the entire cluster.
30. **What are Service Accounts? How are they used?**

- **Answer:** Service Accounts provide an identity for processes that run in Pods. When a Pod makes API calls to the Kubernetes API server, it authenticates using the credentials of its Service Account. They are used for:
    - Allowing Pods to interact with the Kubernetes API.
    - Controlling access to resources within the cluster.
    - Integrating with external systems that need to authenticate to Kubernetes.

31. **How do you secure access to the Kubernetes API server?**
    - **Answer:**
        - **Authentication:** Using client certificates, tokens, or integrated cloud provider authentication.
        - **Authorization (RBAC):** Defining granular permissions.
        - **Admission Controllers:** Enforcing policies before objects are created or modified.
        - **Network Policies:** Restricting network access to the API server.
        - **Encryption in transit:** Using TLS for all communication.
32. **What are Pod Security Standards (PSS) and why are they important?**
    - **Answer:** PSS define a set of security best practices for Pods in Kubernetes. They are divided into three levels:
        - **Privileged:** Unrestricted capabilities, used for highly privileged workloads.
        - **Baseline:** Minimally restrictive, prevents known privilege escalations.
        - **Restricted:** Highly restrictive, enforces hardening best practices.
    - They are important for improving the security posture of your cluster by preventing common security vulnerabilities in Pods.

V. Monitoring and Logging

33. **How do you monitor a Kubernetes cluster? What tools are commonly used?**

- **Answer:** Monitoring a Kubernetes cluster involves collecting metrics, logs, and events. Commonly used tools include:
    - **Metrics:** Prometheus (for scraping metrics) and Grafana (for visualization).
    - **Logging:** Fluentd/Fluent Bit, Logstash, Elasticsearch, Kibana (ELK stack).
    - **Tracing:** Jaeger, Zipkin.
    - **Built-in tools:** kubectl top, kubectl describe, Kubernetes Dashboard (though often not used in production).
    - **Cloud-native solutions:** Google Cloud Monitoring, AWS CloudWatch Container Insights, Azure Monitor for Containers.

34. **How do you collect logs from applications running in Kubernetes?**
    - **Answer:**
        - **Standard output/error:** Applications should write logs to stdout and stderr.
        - **Logging agents:** Deploy a logging agent (like Fluentd, Fluent Bit, or Logstash) as a DaemonSet on each node to collect logs from container runtimes and forward them to a centralized logging system (e.g., Elasticsearch, cloud logging services).
        - **Sidecar containers:** For applications that can't write to stdout/stderr, a sidecar container can be deployed in the same Pod to tail logs from a shared volume and forward them.
35. **What is the role of Prometheus in Kubernetes monitoring?**
    - **Answer:** Prometheus is a popular open-source monitoring system that scrapes metrics from configured targets (like Kubernetes components, nodes, and applications) and stores them in a time-series database. It's often used with Grafana for dashboards and alerts.

VI. Advanced Topics and Troubleshooting Scenarios

36. **Describe a scenario where you would use a Custom Resource Definition (CRD) and a Custom Controller/Operator.**

- **Answer:** You would use CRDs and Operators to extend Kubernetes's functionality for managing complex, stateful applications that have specific operational knowledge.
- **Scenario:** Managing a distributed database like Cassandra or MongoDB. A CRD would define the desired state of your database cluster (e.g., number of nodes, version, backup schedule). An Operator (Custom Controller) would then watch for changes to this CRD, understand the operational complexities of Cassandra, and automate tasks like provisioning new nodes, handling upgrades, performing backups, and recovering from failures.

37. **Explain the concept of Taints and Tolerations.**
    - **Answer:**
        - **Taints:** Applied to nodes to prevent Pods from being scheduled on them unless those Pods explicitly "tolerate" the taint. They mark a node as undesirable for scheduling.
        - **Tolerations:** Applied to Pods, allowing them to be scheduled on nodes that have matching taints. They allow (but don't require) a Pod to be scheduled on a tainted node.
    - **Use cases:** Dedicated nodes for specific workloads, preventing certain Pods from running on unhealthy nodes, isolating critical workloads.

38. **What are Node Affinity and Anti-Affinity?**
    - **Answer:**
        - **Node Affinity:** Forces Pods to be scheduled on nodes with specific labels. It's a "pull" mechanism where Pods "attract" nodes.
            - requiredDuringSchedulingIgnoredDuringExecution: Must meet the rule, but ignored if node labels change later.
            - preferredDuringSchedulingIgnoredDuringExecution: Kubernetes tries to meet the rule but doesn't guarantee it.
        - **Node Anti-Affinity:** Prevents Pods from being scheduled on nodes with specific labels, often to spread Pods across different nodes for high availability.
    - **Use cases:** Ensuring performance, compliance, or high availability.

39. **How would you debug a Pod that is stuck in a Pending state?**
    - **Answer:**

1. kubectl describe pod <pod-name>: Check the Events section for reasons like:
    - **Insufficient CPU/Memory:** The cluster doesn't have enough resources.
    - **Node Selector/Taints/Tolerations:** The Pod has a node selector or toleration that doesn't match any available nodes.
    - **Volume Issues:** PersistentVolumeClaim cannot be bound to a PersistentVolume.
    - **Networking Issues:** CNI plugin not working correctly on nodes.
2. kubectl get events --field-selector involvedObject.name=<pod-name>: More granular events.
3. Check kube-scheduler logs for scheduling decisions.
4. Check node resources: kubectl top nodes (if metrics server is running).

40. **How would you troubleshoot an application that is unreachable from outside the cluster?**
    - **Answer:**

1. **Check Service type:** Is it NodePort or LoadBalancer? If ClusterIP, it's not exposed externally.
2. **Verify Service endpoints:** kubectl describe service <service-name>. Ensure it has healthy Pods as endpoints.
3. **Check Pod status:** kubectl get pods -l app=<service-selector>. Are the Pods running and healthy (Readiness Probes passing)?
4. **Check Ingress (if used):** kubectl describe ingress <ingress-name>. Verify rules, backend services, and events.
5. **Check Ingress Controller:** Ensure the Ingress Controller Pods are running and healthy. Check their logs.
6. **Network Firewall/Security Groups:** For NodePort or LoadBalancer, ensure external firewalls allow traffic to the NodePorts or the LoadBalancer IP.
7. **DNS resolution:** If using a custom domain with Ingress, verify DNS records are correctly pointing to the Ingress Controller's external IP/hostname.
8. **kube-proxy:** Ensure kube-proxy is running on all nodes and check its logs.

41. **What is a Helm chart? Why is it useful?**
    - **Answer:** Helm is the package manager for Kubernetes. A Helm chart is a collection of files that describe a related set of Kubernetes resources. It's useful for:
        - **Packaging:** Bundling all Kubernetes resources for an application into a single, versionable unit.
        - **Deployment:** Easily deploying complex applications with a single command.
        - **Templating:** Parameterizing configurations for different environments.
        - **Management:** Managing releases, upgrades, and rollbacks of applications.
42. **How does Kubernetes handle self-healing?**
    - **Answer:** Kubernetes self-healing capabilities include:
        - **Restarting failed containers:** Liveness probes detect unhealthy containers and restart them.
        - **Rescheduling Pods on failed nodes:** If a node goes down, the controller manager detects it and reschedules its Pods to healthy nodes.
        - **Maintaining desired replicas:** ReplicaSets and Deployments ensure the specified number of Pod replicas are always running.
        - **Rolling back failed deployments:** If a new deployment fails, it can automatically roll back to the previous stable version.
43. **What is a Pod Disruption Budget (PDB)? When would you use it?**
    - **Answer:** A PDB is a Kubernetes API object that specifies the minimum number or percentage of Pods that must be available at all times during voluntary disruptions (e.g., node drain, cluster upgrade, scaling down). It ensures high availability for critical applications. You'd use it for stateful applications or mission-critical services that require a certain number of replicas to be running.
44. **Explain how kubectl exec works.**
    - **Answer:** kubectl exec allows you to execute commands inside a container running in a Pod. It establishes a secure, bidirectional stream

(similar to SSH) between your local machine and the container, enabling you to interact with the container's shell or run specific commands.

45. **What is GitOps and how does it relate to Kubernetes?**
    - **Answer:** GitOps is an operational framework that uses Git as the single source of truth for declarative infrastructure and applications. In Kubernetes, this means:
        - All desired states of your cluster (Kubernetes manifests, Helm charts) are stored in Git.
        - Changes to the cluster are made by creating pull requests to the Git repository.
        - An automated agent (e.g., Flux, Argo CD) continuously observes the Git repository and applies any discrepancies to the cluster, ensuring the cluster's actual state matches the desired state in Git.
    - **Benefits:** Auditable deployments, faster disaster recovery, improved collaboration, and higher reliability.

VII. Real-World Scenarios and Troubleshooting

46. **Scenario: Your application's Pods are constantly restarting. How do you investigate?**
    - **Answer:**
        1. kubectl get pods: Check the RESTARTS column.
        2. kubectl describe pod <pod-name>: Look at the Events section for clues (e.g., OOMKilled, CrashLoopBackOff, failed liveness/readiness probes).
        3. kubectl logs <pod-name>: Check the application logs for errors or exceptions.
        4. kubectl logs <pod-name> -p: Check logs from the *previous* container instance if it's restarting.
        5. Check resource limits and requests in the Pod definition. Is the container running out of memory or CPU?
        6. Examine the application code for bugs or misconfigurations.

47. **Scenario: You need to deploy a new version of your application with zero downtime. How would you achieve this?**
    - **Answer:** Use a Kubernetes Deployment with a rollingUpdate strategy. Ensure your Pods have properly configured Liveness and Readiness Probes. The rollingUpdate strategy will gracefully replace old Pods with new ones, ensuring continuous availability. Consider using maxUnavailable and maxSurge to control the rollout speed and resource consumption during the update.
48. **Scenario: A new application deployment is failing, and the Pods are stuck in ImagePullBackOff. What's wrong?**
    - **Answer:** ImagePullBackOff indicates that Kubernetes is unable to pull the container image. Common causes:
        - **Incorrect image name or tag:** Double-check the image name and tag in the Pod/Deployment manifest.
        - **Private registry authentication:** If using a private registry, ensure imagePullSecrets are correctly configured in the Pod and linked to a Secret with valid credentials.
        - **Network issues:** Node cannot reach the image registry (firewall, DNS, proxy).
        - **Image doesn't exist:** The image might have been deleted from the registry.
49. **Scenario: You have a database running as a StatefulSet, and you need to scale it down. What considerations do you need to make?**
    - **Answer:** StatefulSets scale down gracefully by terminating Pods in reverse ordinal order (e.g., db-2, db-1, db-0). Before scaling down, ensure:
        - **Data integrity:** The database is designed for graceful shutdown and data consistency during scale-down (e.g., by performing graceful shutdowns of database instances).
        - **Data migration/rebalancing:** If the database sharded data, ensure data is rebalanced or migrated off the terminating instances.
        - **PersistentVolumeClaims:** Understand that scaling down a StatefulSet does *not* automatically delete the associated PVCs. You'll need to manually delete them if the data is no longer needed.

        - **Quorum:** Ensure you don't scale down below the minimum required replicas for your database to maintain quorum and availability.

50. **Scenario: You want to ensure that a critical application always has at least 3 Pods running, even during node maintenance. How would you configure this?**
    - **Answer:** Implement a **Pod Disruption Budget (PDB)** for your application with minAvailable: 3 (or minAvailable: 100% if you want all of them). This will tell Kubernetes that it should try to keep at least 3 Pods of that application running during voluntary disruptions like kubectl drain.
51. **Scenario: You notice high CPU utilization on one of your worker nodes. How would you investigate and resolve it?**
    - **Answer:**
        1. kubectl top nodes: Identify which node has high CPU.
        2. kubectl describe node <node-name>: Check resource usage and events.
        3. kubectl top pods --all-namespaces --sort-by='cpu': Identify which Pods on that node are consuming the most CPU.
        4. kubectl logs <pod-name>: Check logs of the high-CPU Pod for application issues.
        5. kubectl exec -it <pod-name> -- top: Go inside the container to see process-level CPU usage.
        6. **Resolution:**
            - **Scale Pods:** If the application can be scaled horizontally, increase the replica count of the high-CPU Pod's Deployment/StatefulSet.
            - **Resource Limits/Requests:** Adjust CPU limits and requests for the Pods to better fit the workload.
            - **Optimize Application:** Profile the application for performance bottlenecks.
            - **Node Autoscaling:** If the overall cluster is consistently high on CPU, consider configuring Cluster Autoscaler to add more nodes.

        - **Taints/Tolerations/Affinity:** Use these to ensure critical workloads get dedicated resources or are spread across nodes.

52. **Scenario: Your development team wants to deploy a new service, but they need to access a database running in a different namespace. How would you enable this communication securely?**
    - **Answer:**
        - **Service Discovery:** The database service can be accessed using its fully qualified domain name (FQDN): database-service-name.database-namespace.svc.cluster.local.
        - **Network Policies:** Implement Network Policies to explicitly allow ingress traffic from the application's namespace to the database service's Pods in the database namespace. This ensures only authorized communication is allowed.
        - **Service Accounts & RBAC:** If the application needs to interact with the Kubernetes API to discover the database (less common for direct database access), ensure the application's Service Account has the necessary RBAC permissions.

VIII. Ecosystem and CNCF Projects (for 2 years experience, some familiarity expected)

53. **What is the Cloud Native Computing Foundation (CNCF)?**
    - **Answer:** The CNCF is an open-source foundation that hosts and promotes cloud-native technologies, including Kubernetes, Prometheus, Envoy, Helm, and many others. Its goal is to make cloud-native computing ubiquitous.
54. **Briefly explain the purpose of some popular CNCF projects related to Kubernetes (e.g., Prometheus, Grafana, Helm, Istio, Envoy, Cilium).**
    - **Answer:**
        - **Prometheus:** Open-source monitoring and alerting toolkit designed for reliability and scalability.
        - **Grafana:** Open-source platform for analytics and interactive visualization. Often used with Prometheus to create dashboards.

        - **Helm:** The package manager for Kubernetes, used to define, install, and upgrade complex Kubernetes applications.
        - **Istio:** An open-source service mesh that provides traffic management, security, and observability for microservices.
        - **Envoy:** An open-source edge and service proxy designed for cloud-native applications. Used as a sidecar proxy in service meshes like Istio.
        - **Cilium:** A networking and security solution for Kubernetes that uses eBPF to provide high-performance networking, security policies, and observability.

55. **Have you worked with any CI/CD pipelines for Kubernetes? If so, describe your experience.**
    - **Answer:** (Focus on your actual experience here)
        - **Example:** "Yes, I've worked with GitLab CI/CD for deploying applications to Kubernetes. We used a pipeline that involved stages for building Docker images, pushing them to a registry, and then using Helm charts to deploy and update our applications in different environments (dev, staging, prod). We leveraged kubectl commands within the pipeline scripts for specific tasks like checking deployment status and performing rollbacks."
        - Mention tools like Jenkins, Argo CD, Flux CD, CircleCI, GitHub Actions, etc., if you have experience.
56. **What are some challenges you've faced working with Kubernetes in a production environment?**
    - **Answer:** (Be honest and show problem-solving skills)
        - **Networking complexities:** Troubleshooting NetworkPolicy issues or CNI plugin problems.
        - **Resource management:** Optimizing CPU/memory requests and limits, dealing with OOMKilled Pods.
        - **Stateful application management:** Handling persistent storage and data integrity for databases.
        - **Security:** Implementing RBAC, managing secrets, and understanding Pod Security Standards.

- **Debugging:** Identifying root causes of issues in a distributed system.
- **Learning curve:** The initial complexity of Kubernetes.
- **Upgrades:** Performing cluster upgrades with minimal disruption.

IX. Deeper Dive into Kubernetes Concepts

57. **Explain the lifecycle of a Pod.**
    - **Answer:** A Pod goes through several phases:
        - **Pending:** The Pod has been accepted by Kubernetes but one or more container images have not been created.
        - **Running:** All containers in the Pod have been created and at least one container is running, or is in the process of starting or restarting.
        - **Succeeded:** All containers in the Pod have terminated successfully, and will not be restarted.
        - **Failed:** All containers in the Pod have terminated, and at least one container has terminated in failure (e.g., non-zero exit code or by system).
        - **Unknown:** The state of the Pod could not be determined.
58. **What are Init Containers and when are they useful?**
    - **Answer:** Init Containers are special containers that run to completion before the main application containers in a Pod start. They are useful for:
        - **Setup/Pre-checks:** Performing setup tasks like database migrations, waiting for a service to be ready, or cloning a Git repository.
        - **Configuration:** Populating configuration files before the main application starts.
        - **Permissions:** Setting correct file permissions for volumes.
        - **Ensuring prerequisites:** Guaranteeing that external dependencies are met before the main application runs.
59. **Explain resource requests and limits in Kubernetes.**
    - **Answer:**

        - **Requests:** The minimum amount of resources (CPU and memory) a container requires. The scheduler uses requests to determine which node a Pod can run on.
        - **Limits:** The maximum amount of resources a container can consume. If a container exceeds its memory limit, it will be OOMKilled (Out Of Memory Killed). If it exceeds its CPU limit, it will be throttled.
    - **Importance:** Proper configuration of requests and limits is crucial for resource allocation, scheduling efficiency, and preventing resource contention.

60. **What is a ReadWriteMany PersistentVolume access mode, and what storage solutions support it?**
    - **Answer:** ReadWriteMany allows a volume to be mounted as read-write by many nodes simultaneously. This is typically supported by network file systems (NFS) or distributed file systems (e.g., CephFS, GlusterFS). Cloud providers often offer managed file storage services (e.g., AWS EFS, Azure Files, Google Filestore) that support this mode.
61. **How do you perform application-level load balancing within a Kubernetes cluster (without using cloud provider load balancers)?**
    - **Answer:**
        - **Service (ClusterIP):** The default Service type provides basic round-robin load balancing among healthy Pods.
        - **Ingress:** For HTTP/HTTPS traffic, an Ingress controller acts as a reverse proxy, distributing traffic based on hostnames and paths to different Services.
        - **Service Mesh (e.g., Istio, Linkerd):** Provides advanced traffic management features like intelligent routing, canary deployments, A/B testing, circuit breaking, and more granular load balancing policies at the application level.
62. **What is the difference between an emptyDir volume and a hostPath volume? When would you use each?**
    - **Answer:**
        - **emptyDir:** A volume that is created when a Pod is first assigned to a node and exists as long as that Pod is running on that node. It's

initially empty and data is lost when the Pod is removed from the node.

- **Use case:** Temporary storage for scratch space, caching, or inter-container communication within a Pod.

- **hostPath:** Mounts a file or directory from the host node's filesystem into a Pod. Data persists across Pod restarts but is tied to the specific node.
  - **Use case:** Accessing node-level data (e.g., logs, monitoring agents), or for applications that need to interact directly with the host filesystem (though generally discouraged for application data due to portability and persistence issues).

63. **How would you troubleshoot an issue where a Service is not routing traffic to any Pods?**
    - **Answer:**

1. kubectl describe service <service-name>: Check the Endpoints field. If it's empty, no Pods are matching the Service's selector.

2. kubectl get pods -l <service-selector>: Verify that Pods with the correct labels exist and are in a Running state with healthy readiness probes.

3. Check the targetPort in the Service definition and containerPort in the Pod definition to ensure they match.

4. Verify network connectivity between the Pods and the Service's cluster IP (e.g., kubectl exec into a Pod and try to curl the service IP/port).

5. Check kube-proxy logs on the node where the Service is being accessed for any errors.

64. **What is a headless Service? When is it used?**
    - **Answer:** A headless Service does not have a ClusterIP. Instead of acting as a load balancer, it provides direct access to the Pod IPs via DNS. When you query the DNS name of a headless service, it returns the IP addresses of the Pods selected by the service.
    - **Use cases:**
      - Stateful applications that need stable network identities for each replica (e.g., peer discovery in a distributed database like Cassandra).

- When you want to control load balancing yourself within your application.

65. **Explain tolerations and nodeSelector in Pod scheduling.**
    - **Answer:**
        - **nodeSelector:** A simple way to constrain Pods to nodes with specific labels. The Pod will *only* be scheduled on nodes that have *all* the specified labels. It's a hard requirement.
        - **tolerations:** Works with taints. A Pod with tolerations can be scheduled on a node that has a matching taint. Without the toleration, the Pod would not be scheduled on that tainted node. It's about *allowing* scheduling on tainted nodes, not forcing it.

X. Scenario-Based Questions (for 2 years experience, focus on practical application)

66. **You have a legacy application that requires a specific kernel module to be loaded on the host. How would you deploy this application in Kubernetes?**
    - **Answer:** This is a tricky one as Kubernetes abstracts away the host OS.
        - **Option 1 (Not ideal for production):** Use a hostPath volume to mount the kernel module directory into the Pod, and potentially run an initContainer to load the module (if allowed by security policies). This ties the Pod to specific nodes.
        - **Option 2 (Better):** Re-architect the application to not require host kernel modules if possible, or deploy it on bare metal/VMs outside Kubernetes.
        - **Option 3 (Advanced/Specific):** If the kernel module is part of a standard distribution and can be enabled, use DaemonSets to ensure the module is loaded on all relevant nodes. For more complex scenarios, consider using a Kubernetes Operator that can manage node-level configurations.
        - **Most practical for a legacy app:** Deploy the application on dedicated nodes with the necessary kernel module pre-loaded, and use nodeSelector or nodeAffinity to schedule the Pods on those nodes.
67. **Your company is moving from a monolithic application to microservices on Kubernetes. What are some key design considerations for this migration?**

- **Answer:**
    - **Containerization:** Ensuring all services are properly containerized (Dockerfiles, optimized images).
    - **Service Decomposition:** Breaking down the monolith into manageable, independent microservices.
    - **API Design:** Defining clear API contracts between microservices.
    - **Service Discovery:** Leveraging Kubernetes Services for inter-service communication.
    - **Configuration Management:** Using ConfigMaps and Secrets for dynamic configuration.
    - **State Management:** Identifying stateful components and using StatefulSets or external databases.
    - **Networking:** Designing appropriate network policies for security.
    - **Logging and Monitoring:** Implementing centralized logging and monitoring for distributed services.
    - **CI/CD Pipeline:** Automating deployments with tools like Helm and GitOps.
    - **Scalability:** Designing services to be horizontally scalable.
    - **Resilience:** Implementing liveness/readiness probes, retries, circuit breakers (potentially with a service mesh).
    - **Security:** Implementing RBAC, Pod Security Standards, and image scanning.

68. **You need to restrict network access to a sensitive database Pod, allowing only specific application Pods to connect to it. How would you achieve this?**
    - **Answer:** Use Kubernetes **Network Policies**.

1. Define a NetworkPolicy in the database's namespace.
2. Use podSelector to target the database Pods.
3. Specify ingress rules with from clauses that use podSelector (and optionally namespaceSelector) to allow traffic *only* from the specific application Pods.
4. Ensure your CNI plugin supports Network Policies (e.g., Calico, Cilium).

69. **How would you handle application configuration that differs between development, staging, and production environments in Kubernetes?**

- **Answer:**
    - **ConfigMaps and Secrets:** Store environment-specific configuration in separate ConfigMaps and Secrets.
    - **Helm Charts:** Use Helm charts with values.yaml files. Create separate values.yaml files (e.g., values-dev.yaml, values-staging.yaml, values-prod.yaml) for each environment. When deploying with Helm, specify the appropriate values file (helm install -f values-prod.yaml ...).
    - **Kustomize:** A native Kubernetes configuration management tool that allows you to customize raw YAML files without templating. You can define a base manifest and then create overlays for each environment to patch specific values.
    - **Environment Variables:** Inject environment-specific values as environment variables into containers.

70. **A developer reports that their Pod cannot write to its mounted volume. What steps would you take to diagnose this?**
    - **Answer:**

1. **Check Pod events:** kubectl describe pod <pod-name> for any volume mount errors.

2. **Check PVC/PV status:** kubectl get pvc <pvc-name> and kubectl describe pvc <pvc-name>, then check the associated kubectl describe pv <pv-name>. Ensure the PVC is Bound to a PV and the PV is available.

3. **Check StorageClass:** If dynamic provisioning is used, ensure the StorageClass is correctly defined and the provisioner is working.

4. **Check underlying storage:** Verify the actual storage (e.g., NFS server, cloud disk) is healthy and accessible from the node.

5. **Check node logs:** On the node where the Pod is running, check kubelet logs for volume mounting issues.

6. **Permissions within the container:** kubectl exec -it <pod-name> -- ls -ld /path/to/mount. Check the permissions of the mounted directory inside the container. The application might not have write permissions. You might need to adjust the securityContext in the Pod definition (e.g., fsGroup, runAsUser).

7. **Selinux/AppArmor (on host):** If present, these security modules on the node might be preventing access.

XI. Advanced Kubernetes Concepts

71. **What is an Admission Controller in Kubernetes? Give an example.**
    - **Answer:** Admission controllers are plugins that intercept requests to the Kubernetes API server *after* authentication and authorization but *before* the object is persisted in etcd. They can mutate (change) or validate requests.
    - **Example:**
        - **LimitRanger:** Enforces resource limits on Pods within a namespace.
        - **ResourceQuota:** Ensures that namespaces don't exceed their allocated resource quotas.
        - **PodSecurityPolicy (deprecated, replaced by PSS):** Used to enforce security policies on Pods.
        - **MutatingAdmissionWebhook/ValidatingAdmissionWebhook:** Allows you to define custom admission controllers using webhooks.
72. **How do you manage certificates and TLS in Kubernetes?**
    - **Answer:**
        - **Kubernetes Secrets:** You can store TLS certificates as Kubernetes Secrets of type kubernetes.io/tls.
        - **Ingress:** Ingress resources can use these TLS Secrets to terminate SSL/TLS at the Ingress Controller.
        - **Cert-manager:** A popular open-source tool that automates the issuance and renewal of TLS certificates from various issuing sources (e.g., Let's Encrypt) within Kubernetes. It integrates with Ingress and creates/manages TLS Secrets.
        - **Service Mesh (e.g., Istio):** Service meshes can manage mTLS (mutual TLS) between services, providing strong identity and encryption.
73. **Explain Kubernetes Operators in more detail. What problems do they solve?**
    - **Answer:** Kubernetes Operators are a method of packaging, deploying, and managing a Kubernetes application. They extend the Kubernetes API

to manage complex stateful applications. An Operator is essentially a Custom Controller that understands the domain-specific knowledge of an application (like a database or a message queue) and automates its operational tasks.

- **Problems they solve:** Automating day-2 operations for complex applications, including:
  - Deployment and upgrades
  - Scaling
  - Backup and restore
  - Failure recovery
  - Cluster rebalancing
  - Applying security patches
  - Handling application-specific configurations

74. **What is kubeconfig and how is it used?**
    - **Answer:** kubeconfig is a YAML file that contains information about your Kubernetes clusters, users, and contexts. It allows kubectl (and other Kubernetes clients) to connect to and authenticate with different Kubernetes clusters. It typically resides at ~/.kube/config.
75. **How does Kubernetes handle garbage collection?**
    - **Answer:** Kubernetes performs garbage collection to clean up resources that are no longer needed. This includes:
      - **Finished Pods/Jobs:** Deleting Pods associated with completed Jobs.
      - **Orphaned Objects:** Deleting objects that are no longer referenced by their owners (e.g., a ReplicaSet deleting Pods that no longer match its selector).
      - **Unused images and containers:** kubelet regularly cleans up old images and containers on the nodes.
      - **kube-controller-manager:** Contains a garbage collector that cleans up various objects.
76. **What is a Multi-Cluster Kubernetes setup? Why would you need one?**

- **Answer:** A multi-cluster Kubernetes setup involves managing multiple independent Kubernetes clusters.
- **Reasons for using it:**
  - **High Availability/Disaster Recovery:** Distributing workloads across different geographical regions or cloud providers to ensure business continuity.
  - **Geographical Proximity/Latency:** Deploying applications closer to users in different regions.
  - **Regulatory Compliance/Data Sovereignty:** Meeting data residency requirements.
  - **Isolation:** Providing strong isolation for different teams or sensitive workloads.
  - **Hybrid Cloud:** Running workloads across on-premise and cloud environments.
  - **Resource Limits:** Overcoming the scaling limits of a single cluster.

77. **What is etcd in Kubernetes? What are its key characteristics?**
    - **Answer:** etcd is a distributed, consistent, and highly available key-value store used by Kubernetes as its primary datastore. It stores all cluster state, configuration data, and metadata (e.g., Pod definitions, Service definitions, ConfigMaps, Secrets).
    - **Key characteristics:**
      - **Consistency:** Uses the Raft consensus algorithm to ensure data consistency across all members.
      - **High Availability:** Designed to be highly available with multiple instances.
      - **Distributed:** Can run across multiple machines.
      - **Key-value store:** Simple API for storing and retrieving data.
      - **Watches:** Allows clients to watch for changes to keys, enabling Kubernetes components to react to state changes.

78. **How does the Kubernetes Scheduler work? What factors does it consider?**
    - **Answer:** The Kube-scheduler is responsible for assigning newly created Pods to available nodes. It considers various factors during the scheduling process:

- **Resource requirements:** CPU, memory, GPU, etc., requested by the Pod.
- **Node capacity:** Available resources on each node.
- **Node selectors, taints, tolerations:** Constraints defined on Pods and nodes.
- **Node affinity/anti-affinity:** Preferences for scheduling Pods on specific nodes or avoiding certain nodes.
- **Pod affinity/anti-affinity:** Preferences for co-locating or separating Pods from other Pods.
- **Quality of Service (QoS) classes:** Ensuring critical Pods get priority.
- **Volume requirements:** Availability of suitable PersistentVolumes.
- **Port conflicts:** Avoiding port conflicts on a node.

XII. Miscellaneous and Best Practices

79. **What are some best practices for writing Dockerfiles for Kubernetes applications?**
    - **Answer:**
        - **Use a minimal base image:** (e.g., alpine, distroless) to reduce image size and attack surface.
        - **Multi-stage builds:** Separate build dependencies from runtime dependencies to create smaller final images.
        - **Cache layers effectively:** Place frequently changing instructions later in the Dockerfile.
        - **Don't run as root:** Use USER instruction to run as a non-root user.
        - **Copy only necessary files:** Use .dockerignore to exclude irrelevant files.
        - **Expose ports:** Use EXPOSE instruction.
        - **CMD/ENTRYPOINT:** Define how the application starts.
        - **Environment variables:** Use for dynamic configuration.

- **Scan images for vulnerabilities:** Integrate image scanning into CI/CD.

80. **How do you secure your Kubernetes cluster at the image level?**
    - **Answer:**
        - **Image scanning:** Use tools (e.g., Trivy, Clair, Anchore) to scan container images for known vulnerabilities.
        - **Trusted registries:** Use private, trusted container registries.
        - **Image signing and verification:** Ensure images come from trusted sources and haven't been tampered with.
        - **Least privilege:** Build images with minimal necessary privileges.
81. **What is the significance of the targetPort and port fields in a Kubernetes Service?**
    - **Answer:**
        - **port:** The port that the Service itself exposes within the cluster. Other services or external clients will connect to this port.
        - **targetPort:** The port on the Pod(s) that the Service will forward traffic to. This is the port your application within the container is listening on.
    - **Significance:** port allows the Service to have a stable external port, while targetPort allows flexibility in how the application inside the Pod is configured, decoupling the service port from the actual container port.
82. **How do you effectively manage secrets in Kubernetes?**
    - **Answer:**
        - **Use Kubernetes Secrets:** For storing sensitive data.
        - **Avoid storing secrets in Git:** Never commit plain text secrets to version control.
        - **External Secret Management:** Integrate with external secret management systems (e.g., HashiCorp Vault, AWS Secrets Manager, Azure Key Vault, Google Secret Manager) using tools like ExternalSecrets Operator or Secrets Store CSI Driver.
        - **RBAC:** Restrict access to Secrets using RBAC policies.

        - **Encryption at rest and in transit:** Ensure etcd is encrypted and all communication is TLS-encrypted.
        - **Rotate secrets regularly.**
        - **Pod Security Standards:** Apply PSS to restrict how Pods can access secrets.

83. **What is kube-proxy and what are its modes of operation?**
    - **Answer:** kube-proxy is a network proxy that runs on each node in the cluster. It's responsible for implementing the Kubernetes Service abstraction by maintaining network rules on nodes and enabling network communication to Pods from inside or outside of the cluster.
    - **Modes of operation:**
        - **iptables (default):** Uses iptables rules to perform load balancing and network address translation (NAT). Efficient for a large number of services.
        - **ipvs:** Uses IP Virtual Server (IPVS) for load balancing, which offers better performance for high-traffic services.
        - **userspace (deprecated):** The oldest and slowest mode, which involves kube-proxy acting as a proxy in userspace.
84. **Explain the concept of Immutable fields in Kubernetes API objects.**
    - **Answer:** Certain fields in Kubernetes API objects are immutable after creation. This means once the object is created, you cannot change the value of these fields directly. If you need to change an immutable field, you typically have to delete and recreate the object (e.g., changing the selector on a Service, or the volumeMounts on a Pod). This is done to maintain consistency and prevent unexpected behavior.
85. **What is kubectl drain used for?**
    - **Answer:** kubectl drain is used to gracefully evict all Pods from a node, preparing it for maintenance (e.g., kernel upgrade, hardware replacement). It marks the node as unschedulable and then evicts Pods, respecting any Pod Disruption Budgets (PDBs).
86. **How would you troubleshoot network latency issues between Pods in a Kubernetes cluster?**
    - **Answer:**

1. **Identify affected Pods/Services:** Is it all traffic, or specific services?

2. **Check CNI plugin:** Verify the health and logs of your CNI plugin (e.g., Calico, Flannel). Are there any errors or warnings?

3. **Node network health:** Check network interface statistics, CPU utilization, and general network performance on the affected nodes.

4. **Underlying network infrastructure:** Is there any latency at the host or cloud provider network level?

5. **Service Mesh (if applicable):** If using a service mesh, check its components and configuration for any issues.

6. **iperf or netperf:** Deploy temporary Pods with network benchmarking tools to measure latency and bandwidth between them.

7. **tcpdump:** Use kubectl debug or kubectl exec to run tcpdump inside Pods to inspect network traffic.

8. **DNS resolution:** Is DNS lookup adding latency?

9. **Application issues:** Is the latency caused by the application itself (e.g., inefficient database queries, excessive logging)?

87. **What are PreStop hooks and PostStart hooks in a container lifecycle?**
    - **Answer:**
        - **PreStop hook:** Executed immediately before a container is terminated due to an API request or a management event (e.g., graceful shutdown, resource contention). It's a blocking call, meaning the container will not be terminated until the hook completes. Useful for graceful shutdowns, flushing logs, or completing in-flight requests.
        - **PostStart hook:** Executed immediately after a container is created. It's a non-blocking call. Useful for initial setup tasks that need to run after the container starts but before the main application logic takes over (e.g., registering with a service registry).
88. **How does Kubernetes handle resource limits and requests on a Pod with multiple containers?**
    - **Answer:** Resource requests and limits are defined *per container* within a Pod. The scheduler sums up the requests of all containers in a Pod to find a node with sufficient aggregate resources. Limits are also applied per container. If one container exceeds its limit, only that specific container is affected (throttled or killed), not the entire Pod.

89. **What is a mutating webhook and a validating webhook?**
    - **Answer:** These are types of Admission Webhooks that allow you to extend the Kubernetes API with custom logic.
        - **MutatingAdmissionWebhook:** Can modify or "mutate" the objects before they are stored in etcd. For example, injecting sidecar containers, adding labels, or setting default values.
        - **ValidatingAdmissionWebhook:** Can only validate requests and either allow or deny them. It cannot change the objects. For example, enforcing specific security policies or preventing certain configurations.
90. **Explain the concept of VolumeClaimTemplates in StatefulSets.**
    - **Answer:** VolumeClaimTemplates are used in StatefulSets to automatically provision PersistentVolumeClaims for each replica of the StatefulSet. Each Pod managed by the StatefulSet will get its own unique PVC based on the template. This ensures that each stateful Pod has its own dedicated, persistent storage.
91. **What is the difference between a Job and a CronJob?**
    - **Answer:**
        - **Job:** Creates one or more Pods and ensures that a specified number of them successfully terminate. Jobs are used for batch processing or one-off tasks.
        - **CronJob:** Manages Jobs on a repeating, time-based schedule (like cron on a Linux system). It creates Job objects at specified intervals.
92. **How can you ensure high availability of the Kubernetes control plane?**
    - **Answer:**
        - **Multiple Master Nodes:** Run multiple kube-apiserver instances behind a load balancer.
        - **Distributed etcd:** Run etcd in a highly available, clustered configuration (e.g., 3 or 5 members).
        - **Redundant Controllers/Schedulers:** kube-controller-manager and kube-scheduler can be run as multiple instances, with only one active at a time (leader election).
        - **Backup and Restore:** Regularly back up etcd data.

        - **Cloud Provider Managed Services:** Utilize managed Kubernetes services (EKS, GKE, AKS) where the cloud provider manages control plane HA.

93. **What is kubeadm?**
    - **Answer:** kubeadm is a tool that helps you bootstrap a minimum viable Kubernetes cluster. It handles the provisioning of control plane components, certificates, and setting up worker nodes to join the cluster. It's not a complete cluster lifecycle manager but a good starting point for self-managed clusters.
94. **Describe a situation where you would use a ServiceAccount and a ClusterRoleBinding together.**
    - **Answer:**
        - **Scenario:** You have an application running in a Pod that needs to list all Nodes in the cluster for monitoring purposes.
        - **Solution:**
            1. Create a ServiceAccount for your application Pod.
            2. Create a ClusterRole that defines permissions to get and list the nodes resource.
            3. Create a ClusterRoleBinding that binds this ClusterRole to your ServiceAccount.
            - By using a ClusterRoleBinding, you grant the ServiceAccount permissions across the entire cluster, allowing it to see all nodes, regardless of the namespace the application Pod runs in.
95. **What is kubectl diff?**
    - **Answer:** kubectl diff shows a diff between the current live configuration of a Kubernetes object and a potential new configuration from a local file. It's a useful tool for previewing changes before applying them, helping to prevent unintended modifications.
96. **How would you secure a Kubernetes cluster from a network perspective (beyond Network Policies)?**
    - **Answer:**

- **Firewalls/Security Groups:** Configure network firewalls (e.g., cloud provider security groups) to restrict ingress/egress traffic to/from nodes and control plane components.
- **Private Endpoints for API Server:** Access the Kubernetes API server only through private network endpoints if available.
- **VPC Peering/VPN:** Securely connect Kubernetes clusters to other private networks.
- **Egress Network Policies:** Control outbound traffic from Pods.
- **Service Mesh:** Implement mTLS and granular traffic control.
- **IDS/IPS:** Deploy intrusion detection/prevention systems at the network edge.

97. **What is the containerRuntimeInterface (CRI)?**
    - **Answer:** CRI is a plugin interface that enables Kubernetes to use different container runtimes (like containerd, CRI-O, Docker's dockershim - now deprecated) to run containers. It provides a gRPC API for the kubelet to interact with the container runtime. This abstraction allows Kubernetes to be independent of the specific container runtime used.
98. **How would you implement blue/green deployments in Kubernetes?**
    - **Answer:**

1. **Two Deployments:** Have two separate Deployments: "blue" (current version) and "green" (new version), each with its own set of Pods.
2. **Service:** A single Kubernetes Service points to the "blue" deployment.
3. **Switch Traffic:** To switch to "green," update the Service's selector to point to the "green" deployment's labels.
4. **Rollback:** If issues arise with "green," simply revert the Service's selector back to "blue."

- This approach offers zero downtime and an immediate rollback capability, but it doubles resource consumption during the switch.

99. **How would you implement canary deployments in Kubernetes?**
    - **Answer:**

1. **Two Deployments:** One for the stable version (main) and another for the canary (new version).

2. **Ingress/Service Mesh:** Use an Ingress controller or a service mesh (e.g., Istio, Linkerd) to manage traffic routing.

3. **Gradual Traffic Shifting:** Configure the Ingress/Service Mesh to gradually shift a small percentage of traffic (e.g., 5-10%) to the canary deployment.

4. **Monitoring:** Closely monitor metrics (errors, latency, performance) for the canary version.

5. **Increase Traffic/Rollout:** If the canary performs well, gradually increase the traffic to the canary until 100%. Then, the old stable deployment can be scaled down and removed.

6. **Rollback:** If issues are detected, immediately revert the traffic split to 0% to the canary.

100. **What is the kubectl debug command? When would you use it?**

- **Answer:** kubectl debug is a beta command in kubectl (as of recent versions) that allows you to create an ephemeral debug container in an existing Pod or create a new debug Pod from an existing Pod or node.
- **Use cases:**
    - **Troubleshooting running Pods:** Attach a debugger or run diagnostic tools without restarting the main application container.
    - **Inspecting a crashlooping container:** Get a shell into a container that's constantly restarting.
    - **Debugging nodes:** Create a debug Pod on a specific node to inspect its filesystem or processes.
    - **Running network diagnostics:** Use tools like tcpdump inside a debug container.