1. **Question:** You've configured node affinity and anti-affinity rules for a workload, but pods are not being scheduled as expected. How would you troubleshoot these scheduling issues? What are best practices for using affinity and anti-affinity to manage workload placement?

Answer: Troubleshooting node affinity/anti-affinity issues:

1. **Describe the pod:** Use kubectl describe pod <pod-name> to see why the pod is not being scheduled. The output will often indicate which affinity/anti-affinity rules are not being satisfied.
2. **Check node labels:** Ensure that nodes have the correct labels that match the pod's affinity/anti-affinity rules.
3. **Verify resource availability:** Make sure that the nodes that match the rules also have sufficient resources (CPU, memory, etc.) to run the pod.

Best practices:

- Use requiredDuringSchedulingIgnoredDuringExecution for hard requirements and preferredDuringSchedulingIgnoredDuringExecution for soft preferences.
- Avoid overly complex rules that can be difficult to manage and debug.

5. Question: Application pods in your cluster are generating large log files, causing storage issues. How would you manage log rotation and retention for pods? What Kubernetes features or external tools can assist in log management?

Answer: Log rotation and retention can be managed using:

- **Kubernetes features:**
  - Using a logging sidecar container with a tool like logrotate.
  - Configuring a Persistent Volume to store logs and using Kubernetes' volume management features for rotation.
- **External tools:**
  - Fluentd, Elasticsearch, and Kibana (EFK stack) for centralized log collection, processing, and storage.
  - Cloud provider logging services (e.g., AWS CloudWatch Logs, Azure Monitor Logs).

These tools allow you to collect logs from multiple pods, rotate them based on size or time, and retain them according to your policies.

6. Question: Your team is migrating to Helm for application deployment, but some charts require custom values. How would you manage and deploy custom values for Helm charts? What strategies can you use to handle versioning and rollbacks for Helm releases?

Answer: Custom values for Helm charts can be managed using:

- **Values files:** Create YAML files with custom values and use the -f flag during Helm install/upgrade.
- **--set flag:** Override individual values using the --set flag during Helm install/upgrade.

Versioning and rollbacks:

- Helm tracks releases and their versions.
- Use helm rollback to revert to a previous release.

7. Question: A StatefulSet for your database application is failing to create pods. How would you debug why the StatefulSet is failing? What are the specific considerations for storage and network configurations in StatefulSets?

Answer: Debugging StatefulSet failures:

1. **Check StatefulSet status:** Use kubectl describe statefulset <statefulset-name> to examine the StatefulSet's status and any associated events.
2. **Examine pod logs:** Check the logs of the pods that are failing to start.
3. **Verify Persistent Volume Claims (PVCs):** Ensure that the PVCs required by the

StatefulSet can be provisioned and bound.

Considerations for StatefulSets:

- **Storage:** StatefulSets require stable storage using PVCs.
- **Networking:** Each pod in a StatefulSet has a unique, stable network identity.

8. Question: Your cluster uses a mix of third-party and custom container images. How would you ensure that only secure and compliant images are used in the cluster? What tools (e.g., image scanners, policies) can assist with this process?

Answer: Ensuring secure and compliant images:

- **Image scanning:** Use tools like Clair, Anchore Engine, or Twistlock to scan images for vulnerabilities.
- **Admission controllers:** Implement admission controllers like Gatekeeper or Kyverno to enforce policies on image usage (e.g., requiring images from trusted registries).

9. Question: You deployed a Terraform module that creates a Kubernetes cluster along with necessary resources for an application. However, one of your Pods is stuck in a "CrashLoopBackOff" state. How would you troubleshoot this issue using Terraform and Kubernetes?

Answer: Troubleshooting CrashLoopBackOff:

1. **Check pod logs:** Use kubectl logs <pod-name> to examine the pod's logs and identify the cause of the crash.
2. **Describe the pod:** Use kubectl describe pod <pod-name> to see events related to the pod and any error messages.
3. **Verify resource limits:** Ensure the pod has sufficient resources (CPU, memory) and that it's not exceeding any limits.
4. **Check Terraform state:** Examine the Terraform state to ensure that all resources were created successfully.

10. Question: Your organization wants to ensure no single team can overuse cluster resources. How would you implement resource quotas and limits using Terraform in a multi-namespace Kubernetes cluster?

Answer: Resource quotas and limits can be implemented using:

- **Resource Quotas:** Define resource quotas in each namespace to limit the total amount of resources (CPU, memory, pods, etc.) that can be used.
- **Limit Ranges:** Set default resource requests and limits for containers within a namespace.

Terraform can be used to automate the creation and management of these resources across multiple namespaces.