Kubernetes Cluster Component Security

Kubelet

Critical component of K8s node agent. It runs on every node. It ensures that containers are running in a pod as expected. The Kubelet communicates with the API Server to receive pod specifications and report the status of the node and its workloads. It also manages the lifecycle of pods, including starting, stopping, and restarting containers as needed.

Key Concepts:

1. **Pod Lifecycle Management:** ensures that the pods are healthy and running, uses liveness and readiness probes to monitor container health.
2. **Node Registration:** register the node with k8s cluster and provide node information update. Includes node capacity, resource usage.
3. **Resource management:** Monitor resources used by pods and nodes. Enforce resource limits, quality of service (QoS) policies.
4. **Container Runtime Interface (CRI):** interacts with CRI to manage container operations.

Security Best Practices:

1. **Restrict Access:** Deny all, then give access as required. Unauthorized operation prevents access, Firewall rules, network restrictions.
2. **Use TLS for Communication:** TLS for communication b/w Kubelet and API Server. Use secure certificates and keys.
3. **Authentication and authorization:** RBAC controls on Kubelet operations.
4. **Audit Logs:** enable them and monitor regularly, Setup alerts for unusual patterns and unauthorized access.
5. **Regular Updates and Patching:** node OS with security patches, container runtime on latest secure version.

Lab Exercise:

Step-by-Step Instructions (do the troubleshooting yourself)

Step 1: Enable Secure Communication

1. **Generate Certificates**
   ◦ Use a tool like openssl to generate server certificates for the Kubelet.

2. **Configure the Kubelet to Use Certificates**
   ◦ Edit the Kubelet configuration file (usually located in /var/lib/kubelet/config.yaml) to include the paths to the certificate and key files.

```
       serverTLSBootstrap: true
       tlsCertFile: "/var/lib/kubelet/pki/kubelet.crt"
       tlsPrivateKeyFile:
"/var/lib/kubelet/pki/kubelet.key"
```

Step 2: Implement Authentication and Authorization

1. **Configure Authentication**
   ◦ Ensure that the Kubelet is configured to require authentication tokens.

```
       authentication:
         x509:
           clientCAFile: "/var/lib/kubelet/pki/ca.crt"
```

2. **Configure Authorization**
   ◦ Enable authorization mode in the Kubelet configuration.

```
       authorization:
         mode: Webhook
```

Step 3: Enable Audit Logging

1. **Enable Audit Logs**

◦ Configure the Kubelet to log audit events by editing the configuration file.

```
featureGates:
  KubeletAudit: true
auditPolicyPath: "/etc/kubernetes/audit-policy.yaml"
```

2. Define an Audit Policy

◦ Create a policy file to specify what events to log.

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
- level: Metadata
  resources:
  - group: ""
    resources: ["pods", "nodes"]
```

3. Apply the Audit Policy

◦ Place the audit policy file in the specified path and restart the Kubelet.

Step 4: Regular Maintenance and Updates

1. Keep the Kubelet Updated

◦ Regularly update the Kubelet and the underlying node OS to the latest versions.

2. Monitor Kubelet Logs

◦ Continuously monitor the Kubelet logs for any signs of issues or security incidents.

Conclusion

Above exercise is just for techies, you can try it out and sort out the errors or perform debugging and troubleshooting yourself it will not come in exam, as it is Multiple Choise Exam.

By following these steps, you have configured and secured the Kubernetes Kubelet. You have enabled secure communication, implemented authentication and authorization, set up audit logging, and ensured regular maintenance. These practices help protect the Kubelet from unauthorized access and provide insights into node and pod activities for security and compliance.