CAREER BYTE CODE
REALTIME PROJECTS PLATFORM

91 COUNTRIES
subscriber 241k Learners
+32 471 40 89 08
CAREERBYTECODE.SUBSTACK.COM

Kubernetes Security Issues

Part 3

Troubleshooting

Pocket Guide

1. Kubernetes Pod Stuck in 'CrashLoopBackOff' Due to SecurityContext Policies

Problem Statement

A Kubernetes pod fails to start and enters a `CrashLoopBackOff` state. Checking logs with `kubectl describe pod <pod-name>` shows security-related errors such as:

Unset

```
Error: permission denied while trying to access the filesystem
```

This issue often occurs due to security restrictions enforced by the `SecurityContext` settings in the pod specification.

What Needs to Be Analyzed

- Run `kubectl describe pod <pod-name>` and check for security-related error messages.
- Inspect the `SecurityContext` settings in the pod spec (`kubectl get pod <pod-name> -o yaml`).
- Verify if the container is attempting to run as `root`, which might be restricted.
- Check Pod Security Policies (PSP) or Pod Security Admission (PSA) for enforced security settings.
- Examine the container image requirements (some images require `root` access).

How to Resolve Step by Step

1. **Check the Security Context in Pod Spec**

Unset

```
kubectl get pod <pod-name> -o yaml | grep -i securityContext -A 10
```

2. Look for:

Unset

```
securityContext:
  runAsUser: 1000
  runAsGroup: 1000
  allowPrivilegeEscalation: false
```

3. **Modify the SecurityContext to Allow Execution**
 If the application needs root privileges, update the deployment spec:

Unset

```
securityContext:
  runAsUser: 0
```

4. Apply the changes:

Unset

```
kubectl apply -f <deployment-file>.yaml
```

5. **Check Pod Security Policies (PSP) or Admission Controls**

Unset

```
kubectl get psp
kubectl describe psp <policy-name>
```

6. If PSP is restricting `runAsUser=0`, modify the policy or create a new one with relaxed rules.

7. **Verify if App Needs Root Privileges**
 If the application does not require root access, rebuild the image with appropriate permissions:

Unset

```
USER 1000
```

8. **Restart the Pod and Verify Logs**

Unset

```
kubectl delete pod <pod-name>
kubectl logs <pod-name> -f
```

Skills Required to Resolve This Issue

- Knowledge of Kubernetes SecurityContext settings
- Understanding of Pod Security Policies (PSP) or Pod Security Admission (PSA)
- Familiarity with containerized application permissions
- YAML configuration skills

Conclusion

Security restrictions on pod execution can cause `CrashLoopBackOff` errors. Understanding `SecurityContext`, PSP, and application requirements helps in resolving the issue effectively.

---

2. Kubernetes Network Policy Blocking Pod Communication

Problem Statement

Pods in the same namespace are unable to communicate due to network policy restrictions. Running `kubectl logs <pod-name>` may show errors like:

Unset

```
Connection timed out
Network is unreachable
```

This issue occurs when a `NetworkPolicy` is applied but does not allow the required traffic.

What Needs to Be Analyzed

- List existing network policies:

Unset

```
kubectl get networkpolicy -n <namespace>
```

- Check if the pod is affected by a network policy:

Unset

```
kubectl describe networkpolicy <policy-name> -n <namespace>
```

- Verify pod labels and match them with the policy.
- Test connectivity using `curl` or `ping` within the cluster.

How to Resolve Step by Step

1. **Check Current Network Policies**

Unset

```
kubectl get networkpolicy -n <namespace>
```

2. **Describe the Policy to See Its Rules**

Unset

```
kubectl describe networkpolicy <policy-name> -n <namespace>
```

3. Look for `podSelector`, `ingress`, and `egress` rules.

4. **Modify the Network Policy to Allow Traffic**
   If a policy is blocking required traffic, update it:

Unset

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-all
  namespace: my-namespace
spec:
  podSelector: {}
  ingress:
    - from:
        - namespaceSelector: {}
  egress:
    - to:
        - namespaceSelector: {}
```

5. **Apply the Updated Policy**

Unset

```
kubectl apply -f <policy-file>.yaml
```

6. **Test Network Connectivity**

Unset

```
kubectl exec -it <pod-name> -- curl <service-name>:<port>
```

Skills Required to Resolve This Issue

- Understanding of Kubernetes Network Policies
- YAML configuration skills
- Networking knowledge (firewalls, CIDR, ports)

Conclusion

Network policies in Kubernetes can restrict communication between pods. Properly configuring ingress and egress rules ensures smooth network connectivity.

3. Kubernetes Service Account Token Expired or Missing Permissions

Problem Statement

A Kubernetes workload fails due to missing or expired service account tokens. Logs may show:

Unset

```
Error: unauthorized
Error: failed to get token
```

This can happen due to incorrect RBAC (Role-Based Access Control) settings or expired tokens.

What Needs to Be Analyzed

- Check the pod’s service account:

Unset

```
kubectl get pod <pod-name> -o yaml | grep serviceAccount
```

- Verify service account token secrets:

Unset

```
kubectl get secrets -n <namespace> | grep <service-account-name>
```

- Check RBAC permissions:

Unset

```
kubectl get roles,rolebindings,clusterroles,clusterrolebindings -n
<namespace>
```

- Verify token expiration:

Unset

```
kubectl describe secret <secret-name> -n <namespace>
```

How to Resolve Step by Step

1. **Create a New Service Account (if needed)**

Unset

```
kubectl create serviceaccount my-service-account -n <namespace>
```

2. **Assign Necessary Permissions**

Unset

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

3. Apply it:

Unset

```
kubectl apply -f <role-file>.yaml
```

4. **Bind the Role to the Service Account**

Unset

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: my-role-binding
  namespace: my-namespace
subjects:
  - kind: ServiceAccount
    name: my-service-account
    namespace: my-namespace
```

```
roleRef:
  kind: Role
  name: my-role
  apiGroup: rbac.authorization.k8s.io
```

5. Apply it:

```
Unset
kubectl apply -f <rolebinding-file>.yaml
```

6. **Update Pod to Use the New Service Account**

```
Unset
serviceAccountName: my-service-account
```

7. **Restart the Pod**

```
Unset
kubectl delete pod <pod-name>
```

Skills Required to Resolve This Issue

- Kubernetes RBAC and service accounts
- YAML configuration skills
- Debugging authentication issues

Conclusion

Service account issues can lead to authentication failures. Ensuring correct RBAC roles and tokens helps resolve permission errors effectively.

4. Kubernetes Pod Cannot Pull Image Due to Unauthorized Error (ErrImagePull / ImagePullBackOff)

Problem Statement

A Kubernetes pod fails to start because it cannot pull the container image from a private registry. When running `kubectl describe pod <pod-name>`, you see errors like:

Unset

```
Failed to pull image "my-private-registry.com/app:latest":
unauthorized: authentication required

ErrImagePull

ImagePullBackOff
```

This usually happens when the image registry requires authentication, but the pod does not have access credentials.

What Needs to Be Analyzed

- Check pod status with:

Unset

```
kubectl get pod <pod-name> -o wide
```

- Describe the pod for more details:

Unset

```
kubectl describe pod <pod-name>
```

- Look for `ErrImagePull`, `ImagePullBackOff`, or `unauthorized` messages.
- Verify if the image is from a private registry:

Unset

```
spec:

  containers:
    - image: my-private-registry.com/app:latest
```

- Check if an `imagePullSecret` exists:

Unset

```
kubectl get secrets -n <namespace>
```

- Confirm the node has internet access and can reach the registry.

How to Resolve Step by Step

1. **Manually Authenticate to the Private Registry**
 Run this command on your local machine:

Unset

```
docker login my-private-registry.com
```

2.
If successful, create a Kubernetes secret for authentication:

Unset

```
kubectl create secret docker-registry my-registry-secret \

  --docker-server=my-private-registry.com \

  --docker-username=<your-username> \

  --docker-password=<your-password> \

  --docker-email=<your-email> \

  -n <namespace>
```

3.
Attach the Secret to the Pod's `imagePullSecrets`

Modify the pod spec or deployment YAML:

Unset

```
spec:

  imagePullSecrets:

    - name: my-registry-secret
```

4. **Verify the Secret is Associated with the Service Account**

Unset

```
kubectl get serviceaccount default -n <namespace> -o yaml
```

5. If missing, patch it:

Unset

```
kubectl patch serviceaccount default -n <namespace> -p
'{"imagePullSecrets": [{"name": "my-registry-secret"}]}'
```

6. **Restart the Pod and Check the Logs**

Unset

```
kubectl delete pod <pod-name>

kubectl get pod -o wide

kubectl describe pod <pod-name>
```

7. **Ensure Kubernetes Nodes Can Reach the Registry**
SSH into a node and test pulling the image manually:

Unset

```
docker pull my-private-registry.com/app:latest
```

Skills Required to Resolve This Issue

- Kubernetes secrets and authentication
- Docker registry authentication
- Debugging network connectivity
- YAML configuration skills

Conclusion

Private container registries require authentication. Ensuring the correct `imagePullSecrets` are set and testing connectivity helps resolve image pull errors.

---

5. Kubernetes Pod Fails Due to Read-Only File System Error

Problem Statement

A pod crashes with an error indicating a read-only file system, such as:

Unset

```
Permission denied: Read-only file system
```

This happens when a container attempts to write to a restricted directory due to security policies like read-only root file system enforcement.

What Needs to Be Analyzed

- Check the pod logs for specific error messages:

Unset

```
kubectl logs <pod-name>
```

- Inspect the pod's `SecurityContext`:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A 10 securityContext
```

- Look for:

Unset

```
securityContext:

  readOnlyRootFilesystem: true
```

- Check the mounted volume permissions:

Unset

```
kubectl describe pod <pod-name>
```

- Verify if the application is trying to write to `/tmp`, `/var`, or `/app` without permissions.

How to Resolve Step by Step

1. **Confirm the Security Policy**
   If `readOnlyRootFilesystem: true` is enforced, change the pod spec:

Unset

```
securityContext:

  readOnlyRootFilesystem: false
```

2.
   Apply changes:

Unset

```
kubectl apply -f <deployment-file>.yaml
```

3.

Use a Writable Volume for Temporary Files
If the app needs to write to a directory, mount a writable volume:

Unset

```
volumeMounts:

  - mountPath: /app/temp

    name: temp-storage

volumes:

  - name: temp-storage

    emptyDir: {}
```

4.

Ensure Correct File Permissions

Unset

```
kubectl exec -it <pod-name> -- ls -l /app
```

5.

If necessary, adjust ownership:

Unset

```
kubectl exec -it <pod-name> -- chown -R 1000:1000 /app
```

6.

Restart the Pod and Verify Logs

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name> -f
```

Skills Required to Resolve This Issue

- Kubernetes SecurityContext knowledge
- Volume management in Kubernetes
- File system debugging in containers

Conclusion

Security settings like read-only root file systems can prevent applications from writing to necessary directories. Using writable volumes and modifying security policies can resolve these issues.

6. Kubernetes Pod Stuck in 'CreateContainerConfigError' Due to Missing Secrets or ConfigMaps

Problem Statement

A Kubernetes pod fails to start and gets stuck in `CreateContainerConfigError`. Running `kubectl describe pod <pod-name>` shows errors like:

Unset

```
Error: secret "my-secret" not found

Error: ConfigMap "my-config" not found
```

This happens when a required Secret or ConfigMap is missing, incorrectly referenced, or not mounted properly.

What Needs to Be Analyzed

- Check pod events to identify the missing Secret or ConfigMap:

Unset

```
kubectl describe pod <pod-name>
```

- Verify if the Secret or ConfigMap exists in the correct namespace:

Unset

```
kubectl get secrets -n <namespace>

kubectl get configmap -n <namespace>
```

- Inspect the pod's YAML file to check how the Secret/ConfigMap is referenced:

Unset

```
kubectl get pod <pod-name> -o yaml
```

- Check if the volume mount paths are correctly configured.

How to Resolve Step by Step

1. **Ensure the Secret/ConfigMap Exists**
   If missing, create it:

Unset

```
kubectl create secret generic my-secret
--from-literal=DB_PASSWORD=supersecret -n <namespace>

kubectl create configmap my-config --from-literal=APP_ENV=production -n
<namespace>
```

2. **Check Names and Namespaces**
   Ensure the pod is referencing the correct name and namespace:

Unset

```
envFrom:

  - secretRef:

      name: my-secret
```

3. **Verify Mount Paths in Deployment YAML**

If using a volume mount:

Unset

```
volumeMounts:

  - name: config-volume

    mountPath: /etc/config

volumes:

  - name: config-volume

    configMap:

      name: my-config
```

4. **Reapply the Deployment and Restart Pods**

Unset

```
kubectl apply -f <deployment-file>.yaml

kubectl delete pod <pod-name>
```

5. **Check Logs for Errors**

Unset

```
kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Kubernetes Secrets and ConfigMaps
- YAML troubleshooting
- Debugging pod events and logs

Conclusion

Misconfigured Secrets and ConfigMaps can cause deployment failures. Ensuring proper names, namespaces, and mounting methods will resolve these errors.

---

7. Kubernetes Pod Cannot Connect to API Server Due to RBAC Denial (Forbidden Error)

Problem Statement

A pod running inside a cluster fails to communicate with the Kubernetes API server, showing errors like:

Unset

```
Error: Forbidden

User "system:serviceaccount:default:my-service-account" cannot get
resource "pods" in API group ""
```

This happens due to missing RBAC (Role-Based Access Control) permissions.

What Needs to Be Analyzed

- Check the service account used by the pod:

Unset

```
kubectl get pod <pod-name> -o yaml | grep serviceAccount
```

- Verify if the service account has the required role bindings:

Unset

```
kubectl get rolebindings,clusterrolebindings -n <namespace>
```

- Check if the necessary RBAC roles exist:

Unset

```
kubectl get roles,clusterroles -n <namespace>
```

How to Resolve Step by Step

1. Create a Role with Required Permissions

Unset

```
kind: Role

apiVersion: rbac.authorization.k8s.io/v1

metadata:

  name: pod-reader

  namespace: my-namespace

rules:

  - apiGroups: [""]

    resources: ["pods"]

    verbs: ["get", "list"]
```

2. Create a RoleBinding for the Service Account

Unset

```
kind: RoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

  name: read-pods-binding

  namespace: my-namespace

subjects:

  - kind: ServiceAccount
```

```
    name: my-service-account

    namespace: my-namespace

roleRef:

  kind: Role

  name: pod-reader

  apiGroup: rbac.authorization.k8s.io
```

3. Apply the RBAC Configuration

Unset

```
kubectl apply -f role.yaml

kubectl apply -f rolebinding.yaml
```

4. Update Pod to Use the Correct Service Account

Unset

```
serviceAccountName: my-service-account
```

5. Restart the Pod and Verify Access

Unset

```
kubectl delete pod <pod-name>
```

Skills Required to Resolve This Issue

- Kubernetes RBAC policies

- Debugging authentication issues
- YAML configuration

Conclusion

RBAC policies control access to cluster resources. Assigning the correct roles and role bindings ensures pods can interact with the Kubernetes API securely.

---

8. Kubernetes Ingress Not Routing Traffic Due to TLS Misconfiguration

Problem Statement

An Ingress resource is defined, but HTTPS requests fail with errors like:

Unset

```
SSL handshake failed

ERR_SSL_PROTOCOL_ERROR
```

This often happens when the TLS secret is missing or incorrectly configured.

What Needs to Be Analyzed

- Check if the TLS secret exists:

Unset

```
kubectl get secret <tls-secret-name> -n <namespace>
```

- Verify the Ingress resource for the correct TLS configuration:

Unset

```
kubectl describe ingress <ingress-name>
```

- Test if the certificate is correctly mounted:

Unset

```
kubectl exec -it <pod-name> -- ls /etc/ssl/certs
```

How to Resolve Step by Step

1. **Create a TLS Secret (if missing)**

Unset

```
kubectl create secret tls my-tls-secret \
  --cert=server.crt --key=server.key \
  -n <namespace>
```

2. **Ensure the Ingress Resource Uses the TLS Secret**

Unset

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  tls:
    - hosts:
        - example.com
      secretName: my-tls-secret
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            backend:
              service:
                name: my-service
                port:
                  number: 443
```

3. **Restart the Ingress Controller**

Unset

```
kubectl rollout restart deployment ingress-nginx-controller -n
ingress-nginx
```

4. **Verify TLS Configuration**
   Test the HTTPS response:

Unset

```
curl -v --insecure https://example.com
```

Skills Required to Resolve This Issue

- Kubernetes Ingress configuration
- TLS/SSL certificate management
- Debugging HTTPS issues

Conclusion

TLS misconfiguration in Kubernetes Ingress can prevent secure connections. Correctly setting up TLS secrets and Ingress rules ensures secure HTTPS communication.

9. Kubernetes Pod Crashes Due to Seccomp Profile Restriction (Permission Denied Error)

Problem Statement

A pod fails to start or crashes due to `Permission Denied` errors when trying to execute certain system calls. Checking logs with `kubectl logs <pod-name>` shows:

Unset

```
operation not permitted
```

or

Unset

```
fatal error: syscall not allowed by seccomp
```

This happens when a Seccomp profile is applied, restricting the container from making certain system calls.

What Needs to Be Analyzed

- Check if the pod has a Seccomp profile enforced:

Unset

```
kubectl get pod <pod-name> -o yaml | grep seccomp
```

- Inspect the node’s security policies:

Unset

```
kubectl get nodes -o yaml | grep seccomp
```

- Review the default Seccomp profile applied in Kubernetes:

Unset

```
kubectl get psp -o yaml
```

- Identify which system calls are blocked by examining the logs.

How to Resolve Step by Step

1. **Check if a Seccomp Profile is Explicitly Set**
 In the pod spec, look for:

Unset

```
securityContext:

  seccompProfile:

    type: RuntimeDefault
```

2. Change it to:

Unset

```
securityContext:

  seccompProfile:

    type: Unconfined
```

3. **Create a Custom Seccomp Profile**
 If you need to allow specific system calls, define a custom profile:

Unset

```
{
  "defaultAction": "SCMP_ACT_ALLOW",
  "syscalls": [
    {
      "names": ["execve"],
      "action": "SCMP_ACT_ALLOW"
    }
  ]
}
```

4. Save it as `/var/lib/kubelet/seccomp/my-seccomp.json` on worker nodes.
5. **Apply the Custom Seccomp Profile to the Pod**

Unset

```
securityContext:
  seccompProfile:
    type: Localhost
    localhostProfile: my-seccomp.json
```

6. **Restart the Pod and Verify Logs**

Unset

```
kubectl delete pod <pod-name>
kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Seccomp security policies in Kubernetes
- YAML configuration for security contexts
- Debugging permission errors

Conclusion

Seccomp profiles restrict system calls for security reasons. Adjusting the security context or creating a custom Seccomp profile can resolve permission errors while maintaining security.

10. Kubernetes Network Policy Blocks Pod Communication (Connection Timeout)

Problem Statement

A pod cannot communicate with another pod, and requests fail with `Connection Timeout` or `Network is Unreachable` errors.
For example:

Unset

```
curl: (28) Connection timed out after 5000 milliseconds
```

This happens due to a restrictive Kubernetes Network Policy.

What Needs to Be Analyzed

- Check if a Network Policy exists:

Unset

```
kubectl get networkpolicy -n <namespace>
```

- Inspect the Network Policy rules:

Unset

```
kubectl describe networkpolicy <policy-name> -n <namespace>
```

- Test pod-to-pod communication using `ping` or `curl`:

Unset

```
kubectl exec -it <pod-name> -- curl <service-name>:<port>
```

- Verify if the pod’s labels match the Network Policy’s allowed selectors.

How to Resolve Step by Step

1. **List Existing Network Policies**

Unset

```
kubectl get networkpolicy -n <namespace>
```

2. **Modify the Network Policy to Allow Traffic**
   Example: Allow all traffic within the same namespace.

Unset

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all
  namespace: my-namespace
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - {}
  egress:
    - {}
```

3. Apply the policy:

Unset

```
kubectl apply -f network-policy.yaml
```

4. **Test Connectivity Again**

Unset

```
kubectl exec -it <pod-name> -- curl <service-name>:<port>
```

5. **If the Issue Persists, Check CNI Plugin Logs**
 If using Calico, check:

Unset

```
kubectl logs -n kube-system -l k8s-app=calico-node
```

6. If using Cilium, check:

Unset

```
kubectl logs -n kube-system -l k8s-app=cilium
```

Skills Required to Resolve This Issue

- Kubernetes Network Policies
- Debugging CNI (Container Network Interface) plugins
- Pod-to-pod communication testing

Conclusion

Restrictive Network Policies can unintentionally block communication between pods. Modifying the policy or creating an explicit allow rule can resolve these issues.

---

11. Kubernetes Pod Running as Root Blocked by SecurityContext (RunAsNonRoot Error)

Problem Statement

A pod fails to start due to security restrictions preventing it from running as the root user. The pod events may show:

```
Unset
Error: container has runAsNonRoot and image has non-numeric user
(root), cannot verify user is non-root
```

or

```
Unset
container has runAsNonRoot set to true but image is running as root
```

This happens because Kubernetes enforces the `runAsNonRoot` security policy, but the container tries to run as the root user.

What Needs to Be Analyzed

- Check the security settings in the pod spec:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 securityContext
```

- Verify the user ID inside the container:

Unset

```
kubectl exec -it <pod-name> -- id
```

- Inspect the Docker image to see if it runs as root:

Unset

```
docker inspect <image-name> | grep -i user
```

- Check if the Kubernetes Pod Security Admission (PSA) or Pod Security Policy (PSP) is enforcing this restriction.

How to Resolve Step by Step

1. **Modify the Pod Security Context to Allow Root (if necessary)**
   If security allows it, update the deployment YAML:

Unset

```
securityContext:

  runAsNonRoot: false
```

2. **Ensure the Image Uses a Non-Root User**
   If `runAsNonRoot: true` is required, modify the Dockerfile to use a non-root user:

Unset

```
RUN useradd -u 1001 appuser

USER appuser
```

3. Rebuild and push the new image:

Unset

```
docker build -t my-registry.com/app:latest .

docker push my-registry.com/app:latest
```

4. **Specify a Non-Root User in the Pod Spec**
 If modifying the image is not possible, specify a numeric non-root user:

Unset

```
securityContext:

  runAsUser: 1001
```

5. **Restart the Pod and Verify**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Kubernetes SecurityContext policies
- Dockerfile best practices
- Debugging pod permissions

Conclusion

Running containers as root is a security risk. Ensuring images use a non-root user or modifying SecurityContext settings can resolve permission issues while maintaining security.

12. Kubernetes Pod Cannot Bind to Privileged Port (<1024) Due to Security Policies

Problem Statement

A pod fails to start when trying to bind to a privileged port (e.g., `80`, `443`), showing errors like:

Unset

```
Error: listen tcp :80: bind: permission denied
```

This happens because Kubernetes enforces security restrictions preventing non-root users from binding to ports below `1024`.

What Needs to Be Analyzed

- Check the container logs for `bind: permission denied` errors:

Unset

```
kubectl logs <pod-name>
```

- Verify the application's configured port:

Unset

```
kubectl describe pod <pod-name>
```

- Check if the pod runs as a non-root user:

Unset

```
kubectl get pod <pod-name> -o yaml | grep runAsUser
```

How to Resolve Step by Step

1. **Change the Application to Use a Non-Privileged Port**
   Modify the app configuration to use a port above `1024`, such as `8080` or `8443`.
   Update the Deployment YAML:

Unset

```
ports:

  - containerPort: 8080
```

2. **Use an Init Container to Set Up Port Forwarding**
 If the application requires a privileged port, use `iptables` to redirect traffic:

Unset

```
initContainers:

  - name: init-iptables

    image: busybox

    command: ["sh", "-c", "iptables -t nat -A PREROUTING -p tcp --dport
80 -j REDIRECT --to-port 8080"]
```

3. **Grant the Container `CAP_NET_BIND_SERVICE` Capability**
 Modify the security context:

Unset

```
securityContext:

  capabilities:

    add: ["NET_BIND_SERVICE"]
```

4. **Restart the Pod and Test Connectivity**

Unset

```
kubectl delete pod <pod-name>

curl http://<pod-ip>:8080
```

Skills Required to Resolve This Issue

- Linux capabilities (`NET_BIND_SERVICE`)
- Kubernetes security policies
- Debugging network port bindings

Conclusion

Binding to privileged ports requires running as root or using the `NET_BIND_SERVICE` capability. Redirecting traffic or modifying application ports can resolve this issue securely.

13. Kubernetes Pod Fails to Pull Image Due to Private Registry Authentication Issues

Problem Statement

A Kubernetes pod fails to start because it cannot pull an image from a private container registry. Running `kubectl describe pod <pod-name>` shows errors like:

Unset

```
Failed to pull image "my-private-registry.com/app:latest":
unauthorized: authentication required

Error: ImagePullBackOff
```

This occurs when Kubernetes is not configured to authenticate with the private registry.

What Needs to Be Analyzed

- Check pod events for authentication errors:

Unset

```
kubectl describe pod <pod-name>
```

- Verify if an image pull secret exists:

Unset

```
kubectl get secrets -n <namespace>
```

- Confirm that the pod is using the correct secret:

Unset

```
kubectl get pod <pod-name> -o yaml | grep imagePullSecrets
```

- Ensure the Docker registry credentials are correct.

How to Resolve Step by Step

1. **Create an Image Pull Secret**

Unset

```
kubectl create secret docker-registry my-registry-secret \
  --docker-server=my-private-registry.com \
  --docker-username=<username> \
  --docker-password=<password> \
  --docker-email=<email> -n <namespace>
```

2. **Attach the Secret to the Pod Spec**
 Modify the deployment YAML:

Unset

```
spec:
  imagePullSecrets:
    - name: my-registry-secret
```

3. **Verify Kubernetes Can Pull the Image**

Unset

```
kubectl run test-pull --image=my-private-registry.com/app:latest
--dry-run=client -o yaml | kubectl apply -f -
```

4. **Restart the Pod**

Unset

```
kubectl delete pod <pod-name>
```

Skills Required to Resolve This Issue

- Kubernetes secrets and authentication
- Docker registry management
- Debugging image pull failures

Conclusion

Private registries require authentication for image pulls. Creating and attaching an image pull secret ensures Kubernetes can access the registry securely.

---

14. Kubernetes Pod Terminated Due to OOMKilled (Out of Memory Issue)

Problem Statement

A pod unexpectedly terminates, and checking the pod status with `kubectl describe pod <pod-name>` shows:

Unset

```
State:    Terminated

Reason:   OOMKilled
```

This occurs when a container exceeds its allocated memory limit.

What Needs to Be Analyzed

- Check pod events for memory-related errors:

Unset
```
kubectl describe pod <pod-name>
```

- Review container resource limits:

Unset
```
kubectl get pod <pod-name> -o yaml | grep -A5 resources
```

- Analyze memory usage of the container:

Unset
```
kubectl top pod <pod-name>
```

- Check if the node itself is running out of memory:

Unset
```
kubectl top nodes
```

How to Resolve Step by Step

1. **Increase Memory Limits in Deployment YAML**

Unset
```
resources:

  limits:

    memory: "512Mi"
  requests:

    memory: "256Mi"
```

2. **Optimize Application Memory Usage**
   - Use a lightweight base image (e.g., `alpine`)
   - Optimize caching and memory-intensive operations
3. **Enable Kubernetes Eviction Policies**
   If the node runs out of memory, adjust eviction thresholds in the kubelet configuration:

Unset

```
--eviction-hard=memory.available<100Mi
```

4. **Restart the Pod and Monitor Memory Usage**

Unset

```
kubectl delete pod <pod-name>

kubectl top pod <pod-name>
```

Skills Required to Resolve This Issue

- Kubernetes resource management
- Debugging memory consumption
- Application performance tuning

Conclusion

Setting appropriate memory limits and optimizing application memory usage prevents OOMKilled errors, ensuring pod stability.

---

15. Kubernetes Pod Stuck in CrashLoopBackOff Due to Read-Only Filesystem

Problem Statement

A pod continuously restarts, showing a `CrashLoopBackOff` status. Checking logs with `kubectl logs <pod-name>` reveals errors like:

Unset

```
Error: Read-only file system

Permission denied: cannot write to /var/log/app.log
```

This happens when the pod attempts to write to a filesystem path that is mounted as read-only due to security settings.

What Needs to Be Analyzed

- Check the pod logs for `Read-only file system` errors:

Unset

```
kubectl logs <pod-name>
```

- Inspect the security settings in the pod spec:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 securityContext
```

- Verify if the container has a read-only root filesystem:

Unset

```
kubectl describe pod <pod-name> | grep -A5 ReadOnlyRootFilesystem
```

- Check if a volume is mounted at the location the application is trying to write to:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 volumeMounts
```

How to Resolve Step by Step

1. **Disable Read-Only Filesystem (If Allowed)**
   If security allows, update the deployment YAML:

Unset

```
securityContext:

  readOnlyRootFilesystem: false
```

2. **Use an EmptyDir Volume for Writable Paths**
   If the application requires writing logs or temporary files, mount a writable volume:

Unset

```
volumeMounts:

  - name: writable-dir

    mountPath: /var/log

volumes:

  - name: writable-dir

    emptyDir: {}
```

3. **Modify the Application to Use a Writable Path**
   Update the application to write logs to a mounted directory instead of the root filesystem.

4. **Restart the Pod and Verify Logs**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Kubernetes security policies (`readOnlyRootFilesystem`)
- Debugging file permission errors
- Configuring volume mounts in Kubernetes

Conclusion

Restricting write access to the root filesystem enhances security. Mounting writable volumes or updating application paths can resolve filesystem-related crashes.

16. Kubernetes Pod Fails Due to ServiceAccount Token Mounting Restrictions

Problem Statement

A pod fails to start, and checking logs or pod events shows errors related to missing permissions, such as:

Unset

```
Error: cannot access Kubernetes API: permission denied

ServiceAccount token not found
```

This occurs when the pod requires API access, but the ServiceAccount token is not mounted due to security restrictions.

What Needs to Be Analyzed

- Check if the ServiceAccount token is mounted:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 serviceAccount
```

- Verify if token automounting is disabled in the ServiceAccount:

Unset

```
kubectl get serviceaccount <sa-name> -o yaml
```

- Ensure the correct RBAC permissions are assigned:

Unset

```
kubectl get rolebinding -n <namespace>
```

How to Resolve Step by Step

1. **Enable ServiceAccount Token Mounting**
   If the pod needs the default token, modify the deployment:

Unset

```
automountServiceAccountToken: true
```

2. **Create a Custom ServiceAccount with Correct Permissions**

Unset

```
kubectl create serviceaccount custom-sa -n <namespace>
```

3. **Assign RBAC Permissions**

Unset

```
apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

  name: api-access

  namespace: my-namespace

subjects:

  - kind: ServiceAccount

    name: custom-sa

    namespace: my-namespace

roleRef:

  kind: Role

  name: pod-reader

  apiGroup: rbac.authorization.k8s.io
```

4. **Attach the ServiceAccount to the Pod**

Unset

```
serviceAccountName: custom-sa
```

5. **Restart the Pod and Verify API Access**

Unset

```
kubectl delete pod <pod-name>
```

Skills Required to Resolve This Issue

- Kubernetes RBAC and ServiceAccounts
- Debugging API authentication failures
- Managing ServiceAccount token security

Conclusion

Restricting ServiceAccount token mounting improves security but may cause API access issues. Creating a custom ServiceAccount with appropriate permissions resolves the problem.

---

17. Kubernetes Pod Fails Due to NetworkPolicy Blocking Traffic

Problem Statement

A pod is unable to communicate with another pod, service, or external system. Running network tests inside the pod (e.g., `curl`, `ping`) results in:

Unset

```
Timeout error

Connection refused

No route to host
```

This usually happens because a Kubernetes `NetworkPolicy` is restricting the traffic.

What Needs to Be Analyzed

- Check if a `NetworkPolicy` is applied in the namespace:

Unset

```
kubectl get networkpolicy -n <namespace>
```

- Inspect the specific policy rules:

Unset

```
kubectl describe networkpolicy <policy-name> -n <namespace>
```

- Test connectivity between pods:

Unset

```
kubectl exec -it <pod-name> -- curl <service-name>:<port>
```

- Ensure the pod has the correct labels to match the NetworkPolicy rules.

How to Resolve Step by Step

1. **Allow Ingress Traffic for the Pod**
   If the pod should receive traffic, modify the policy:

Unset

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-internal-traffic
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      app: my-app
  ingress:
    - from:
        - podSelector: {}
```

2. **Allow Egress Traffic for the Pod**
 If the pod should send outbound requests, modify the egress policy:

Unset

```
egress:

  - to:

      - ipBlock:

          cidr: 0.0.0.0/0
```

3. **Verify the Pod Labels Match the NetworkPolicy**

Unset

```
kubectl get pods --show-labels -n <namespace>
```

4. **Test Connectivity After Changes**

Unset

```
kubectl exec -it <pod-name> -- curl <service-name>:<port>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes `NetworkPolicy`
- Debugging network connectivity issues
- Managing pod security restrictions

Conclusion

`NetworkPolicy` is an essential security feature but can unintentionally block required traffic. Adjusting ingress and egress rules ensures secure and functional networking.

18. Kubernetes Pod Fails Due to PodSecurityAdmission (PSA) or PodSecurityPolicy (PSP) Restrictions

Problem Statement

A pod fails to start, and checking events with `kubectl describe pod <pod-name>` shows errors like:

Unset

```
PodSecurityPolicy denied the request: Privileged mode is not allowed
```

or

Unset

```
Pod is forbidden due to PodSecurity admission control restrictions
```

This happens when Kubernetes security policies prevent certain privileges.

What Needs to Be Analyzed

- Check if `PodSecurityAdmission` is enabled:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- If using `PodSecurityPolicy`, check allowed privileges:

Unset

```
kubectl get psp -o yaml
```

- Inspect the pod spec for restricted settings:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A10 securityContext
```

How to Resolve Step by Step

1. **Determine the Required Security Level**
   If PodSecurityAdmission is used, check if the namespace has a restrictive label:

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=privileged
```

2. **Modify the Pod SecurityContext to Comply**
   Remove privileged mode if not necessary:

Unset

```
securityContext:

  privileged: false
```

3. **Grant the Pod Required Privileges** (If Allowed)
   If necessary, create a `RoleBinding` to assign a less restrictive `PodSecurityPolicy`:

Unset

```
kubectl create rolebinding psp-access --clusterrole=privileged-psp
--serviceaccount=<namespace>:<serviceaccount> -n <namespace>
```

4. **Restart the Pod**

Unset

```
kubectl delete pod <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes `PodSecurityAdmission` and `PodSecurityPolicy`
- Debugging security context issues
- Managing RBAC and security policies

Conclusion

Pod security restrictions prevent unauthorized access but may block required permissions. Adjusting namespace security levels or modifying pod settings ensures compliance and functionality.

---

19. Kubernetes Pod Fails Due to Seccomp Profile Restrictions

Problem Statement

A pod fails to start, and checking `kubectl describe pod <pod-name>` shows errors like:

Unset

```
PodSecurityPolicy denied the request: seccomp profile required

Permission denied due to seccomp restrictions
```

This occurs when the Kubernetes security context enforces a `seccomp` profile that restricts certain system calls.

What Needs to Be Analyzed

- Check if `seccomp` is enabled in the cluster:

Unset

```
kubectl get psp -o yaml | grep seccomp
```

- Inspect the pod's security context for `seccompProfile`:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 seccompProfile
```

- Ensure the node supports the required seccomp profile:

Unset

```
ls /var/lib/kubelet/seccomp
```

- Look for pod logs or dmesg output for syscall violations:

Unset

```
journalctl -k | grep seccomp
```

How to Resolve Step by Step

1. **Check Available Seccomp Profiles**
   On the node, list available profiles:

Unset

```
ls /var/lib/kubelet/seccomp
```

2. **Update the Pod Spec to Use an Allowed Seccomp Profile**
   Modify the deployment YAML to use `runtime/default`:

Unset

```
securityContext:

  seccompProfile:

    type: RuntimeDefault
```

3. **Disable Seccomp (If Allowed)**
   If security policies permit, remove the `seccompProfile` section:

Unset

```
securityContext: {}
```

4. **Apply Changes and Restart the Pod**

Unset

```
kubectl delete pod <pod-name>
```

5. **Monitor Logs for Any Remaining Restrictions**

Unset

```
kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes `seccomp` security settings
- Debugging system call restrictions
- Managing pod security policies

Conclusion

`Seccomp` enhances security by limiting system calls but can block required functionality. Configuring the correct profile or disabling `seccomp` when necessary ensures smooth pod execution.

---

20. Kubernetes Pod Fails Due to AppArmor Profile Violations

Problem Statement

A pod fails to start, and checking `kubectl describe pod <pod-name>` shows errors like:

Unset

```
Operation not permitted due to AppArmor profile
```

or

Unset

```
AppArmor denied the request
```

This happens when the node enforces an `AppArmor` profile that blocks certain operations.

What Needs to Be Analyzed

- Check if `AppArmor` is enabled on the node:

Unset

```
aa-status
```

- Inspect the pod spec for `AppArmorProfile`:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 apparmor.security.beta.kubernetes.io
```

- Look for denied operations in the system logs:

Unset

```
dmesg | grep DENIED
```

How to Resolve Step by Step

1. **Check Available AppArmor Profiles**
   On the node, list available profiles:

Unset

```
sudo aa-status
```

2. **Update the Pod Spec to Use an Allowed Profile**
   Modify the deployment YAML:

Unset

```
metadata:

  annotations:

    container.apparmor.security.beta.kubernetes.io/my-container:
runtime/default
```

3. **Disable AppArmor (If Allowed)**
   If security policies permit, remove the annotation:

Unset

```
metadata:

  annotations: {}
```

4. **Apply Changes and Restart the Pod**

Unset

```
kubectl delete pod <pod-name>
```

5. **Monitor Logs for Any Remaining Issues**

Unset

```
kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes `AppArmor` security settings
- Debugging denied system operations
- Managing node security policies

Conclusion

`AppArmor` improves security by restricting system operations, but misconfigurations can block necessary processes. Adjusting profiles or disabling enforcement (if allowed) ensures the pod functions correctly.

---

21. Kubernetes Pod Fails Due to Service Mesh (Istio/Linkerd) Security Policies

Problem Statement

A pod is unable to communicate with other services in the cluster after deploying a service mesh like Istio or Linkerd. The logs may show errors such as:

Unset

```
upstream connect error or disconnect/reset before headers. reset
reason: connection termination

403 Forbidden: RBAC access denied
```

This occurs when the service mesh enforces strict mTLS, authorization policies, or traffic rules that block communication.

What Needs to Be Analyzed

- Check if the service mesh is enforcing mTLS:

```
kubectl get peerauthentication -A
```

- Inspect authorization policies that may be blocking requests:

```
kubectl get authorizationpolicy -A
```

- Review sidecar injection to ensure the pod has the correct proxy:

```
kubectl get pod <pod-name> -o
jsonpath='{.metadata.labels.istio-injection}'
```

- Test connectivity from within the pod:

```
kubectl exec -it <pod-name> -- curl <service-name>:<port>
```

How to Resolve Step by Step

1. **Enable mTLS for the Namespace or Service**
   If mTLS is required, create a `PeerAuthentication` policy:

```
apiVersion: security.istio.io/v1beta1

kind: PeerAuthentication

metadata:

  name: default

  namespace: my-namespace
```

```
spec:
  mtls:
    mode: STRICT
```

2.
Allow Traffic Using AuthorizationPolicy
If requests are being blocked, create an authorization policy:

Unset

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-all
  namespace: my-namespace
spec:
  action: ALLOW
  rules:
    - {}
```

3.
Check and Restart the Sidecar Proxy
If the sidecar is missing, restart the pod to trigger injection:

Unset

```
kubectl delete pod <pod-name>
```

4. **Verify the Fix by Retesting Connectivity**

Unset

```
kubectl exec -it <pod-name> -- curl <service-name>:<port>
```

Skills Required to Resolve This Issue

- Understanding Istio/Linkerd security settings
- Debugging mTLS and RBAC issues
- Configuring authorization policies

Conclusion

Service meshes enhance security but may block unintended traffic. Adjusting mTLS and authorization policies ensures proper communication.

---

22. Kubernetes Pod Fails Due to Node-Level Security Policies (SELinux, AppArmor, Seccomp)

Problem Statement

A pod fails to start or access necessary resources, and logs show errors like:

Unset

```
Permission denied

Operation not permitted
```

This occurs when node-level security policies (SELinux, AppArmor, or Seccomp) restrict the pod's access.

What Needs to Be Analyzed

- Check if SELinux is enforcing policies:

Unset

```
sestatus
```

- Inspect the audit logs for denied operations:

Unset

```
sudo ausearch -m AVC -ts recent
```

- Verify AppArmor restrictions:

Unset

```
aa-status
```

- Check the pod's `securityContext` for restrictive settings:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 securityContext
```

How to Resolve Step by Step

1. **Allow the Pod to Run in an SELinux-Compatible Context**
   If SELinux is blocking access, set the correct SELinux type:

Unset

```
securityContext:

  seLinuxOptions:

    level: "s0:c123,c456"
```

2.
   Modify Seccomp Profile if Necessary
   If `seccomp` is blocking execution, update the securityContext:

Unset

```
securityContext:

  seccompProfile:

    type: Unconfined
```

3. **Adjust AppArmor Profile (If Required)**
   Modify the pod spec to use a more permissive profile:

Unset

```
metadata:

  annotations:

    container.apparmor.security.beta.kubernetes.io/my-container: unconfined
```

4. **Restart the Pod and Verify Permissions**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding SELinux, AppArmor, and Seccomp
- Debugging permission issues at the node level
- Configuring Kubernetes security contexts

Conclusion

Node-level security policies protect against threats but can block necessary operations. Adjusting SELinux, AppArmor, and Seccomp settings ensures proper functionality while maintaining security.

23. Kubernetes Pod Fails Due to Insufficient Pod Security Standards (PSS) Permissions

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` returns an error like:

Unset

```
Pod security policy violation: PodSecurity "restricted" level forbids
the request
```

This happens when Kubernetes `Pod Security Standards (PSS)` restrict pod permissions.

What Needs to Be Analyzed

- Check which PSS mode is enforced in the namespace:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Inspect pod security settings:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A10 securityContext
```

- Verify if the required permissions are allowed by PSS:

Unset

```
kubectl auth can-i create pods
--as=system:serviceaccount:<namespace>:<serviceaccount>
```

How to Resolve Step by Step

1. **Identify the Current Security Level**
 If PSS is too restrictive, change the namespace label:

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=privileged --overwrite
```

2. **Adjust Pod SecurityContext to Match the Enforced Level**
 Modify the deployment YAML:

Unset

```
securityContext:

  privileged: false

  runAsUser: 1000

  allowPrivilegeEscalation: false

  readOnlyRootFilesystem: true
```

3. **Grant a ServiceAccount the Required Role (If Necessary)**

Unset

```
kubectl create rolebinding allow-pss-exempt \

  --clusterrole=pss-exempt \

  --serviceaccount=<namespace>:<serviceaccount> \

  --namespace=<namespace>
```

4. **Restart the Pod and Verify**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes Pod Security Standards (PSS)
- Managing namespace security labels
- Configuring RBAC and security policies

Conclusion

PSS helps enforce security best practices, but misconfigurations can block pods. Adjusting namespace policies or pod security contexts resolves the issue while maintaining security compliance.

24. Kubernetes Pod Fails Due to Read-Only Filesystem Policy

Problem Statement

A pod tries to write to a filesystem location and fails with errors like:

Unset

```
Read-only file system

Permission denied: cannot write to /tmp
```

This occurs when the pod runs in a read-only filesystem for security purposes.

What Needs to Be Analyzed

- Check if the pod has `readOnlyRootFilesystem` enabled:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 readOnlyRootFilesystem
```

- Inspect container logs for filesystem-related errors:

Unset

```
kubectl logs <pod-name>
```

- Verify if the application requires write access to specific directories.

How to Resolve Step by Step

1. **Check the Application's Required Writable Directories**
   Identify which directories need write access (e.g., `/tmp`, `/var/log`).

2. **Mount an EmptyDir Volume for Writable Paths**
   Update the pod spec to allow temporary writes:

Unset

```
volumes:
  - name: tmp-volume
    emptyDir: {}
containers:
  - name: my-container
    volumeMounts:
      - mountPath: /tmp
        name: tmp-volume
```

3.
   Disable Read-Only Filesystem (If Necessary)
   If security policies allow, modify the security context:

Unset

```
securityContext:
  readOnlyRootFilesystem: false
```

4. **Restart the Pod and Test File Writes**

Unset

```
kubectl delete pod <pod-name>

kubectl exec -it <pod-name> -- touch /tmp/testfile
```

Skills Required to Resolve This Issue

- Understanding Kubernetes security contexts
- Debugging application filesystem errors
- Configuring pod volumes

Conclusion

Running pods with a read-only filesystem enhances security, but some applications require writable paths. Using `emptyDir` volumes or adjusting security contexts ensures proper functionality.

---

25. Kubernetes Pod Fails Due to NetworkPolicy Blocking Traffic

Problem Statement

A pod is unable to communicate with other services within the cluster or externally. Checking logs or using `kubectl describe pod <pod-name>` may show:

Unset

```
Timeout while connecting to service

Connection refused
```

```
Network is unreachable
```

This occurs when a Kubernetes `NetworkPolicy` is blocking traffic to or from the pod.

What Needs to Be Analyzed

- Check if any `NetworkPolicy` is applied to the namespace:

```
Unset
kubectl get networkpolicy -n <namespace>
```

- Inspect the details of the applied `NetworkPolicy`:

```
Unset
kubectl describe networkpolicy <policy-name> -n <namespace>
```

- Verify the pod's labels match the allowed rules in the policy:

```
Unset
kubectl get pod <pod-name> --show-labels
```

- Test connectivity using `curl` or `netcat`:

```
Unset
kubectl exec -it <pod-name> -- curl <service-name>:<port>
```

How to Resolve Step by Step

1. **Allow Incoming Traffic (Ingress) If Required**
   If ingress traffic is blocked, create or modify a policy:

Unset

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-ingress
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      app: my-app
  policyTypes:
    - Ingress
  ingress:
    - from:
        - podSelector: {}
```

2. **Allow Outgoing Traffic (Egress) If Required**
 If egress traffic is blocked, add:

Unset

```
egress:
  - to:
      - namespaceSelector: {}
    ports:
      - protocol: TCP
```

```
          port: 443
```

3. **Delete Any Overly Restrictive Policy (If Needed)**
 If no restrictive policies are necessary, remove them:

Unset

```
kubectl delete networkpolicy <policy-name> -n <namespace>
```

4. **Restart the Pod and Verify Connectivity**

Unset

```
kubectl delete pod <pod-name>

kubectl exec -it <pod-name> -- curl <service-name>:<port>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes `NetworkPolicy`
- Debugging connectivity issues using `curl` and `netcat`
- Configuring pod-level network security

Conclusion

`NetworkPolicy` improves cluster security but can accidentally block required traffic. Configuring proper ingress/egress rules ensures secure yet functional communication.

---

26. Kubernetes Pod Fails Due to PodSecurityAdmission (PSA) Restrictions

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows:

Unset

```
PodSecurity "restricted" level forbids the request
```

This happens when Kubernetes `Pod Security Admission (PSA)` enforces stricter security policies.

What Needs to Be Analyzed

- Check if PSA is enabled in the namespace:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Inspect the pod spec to see if it violates PSA rules:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 securityContext
```

- Verify if the namespace is enforcing `restricted`, `baseline`, or `privileged` level:

Unset

```
kubectl label namespace <namespace>
```

How to Resolve Step by Step

1. **Check the Current PSA Level**
 If PSA is too strict, update the namespace label:

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=privileged --overwrite
```

2.
Modify the Pod's SecurityContext to Match the PSA Level

Update the deployment YAML to comply with `baseline` security:

Unset

```
securityContext:

  runAsNonRoot: true

  runAsUser: 1000

  allowPrivilegeEscalation: false
```

3. **Grant an Exemption for a Specific ServiceAccount (If Needed)**

Unset

```
kubectl create rolebinding psa-exempt \

  --clusterrole=psa-exempt \

  --serviceaccount=<namespace>:<serviceaccount> \

  --namespace=<namespace>
```

4. **Restart the Pod and Verify Security Compliance**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes `Pod Security Admission (PSA)`
- Managing namespace security policies
- Configuring pod security contexts

Conclusion

PSA ensures cluster-wide security, but restrictive policies can block necessary workloads. Adjusting namespace labels and pod security contexts resolves compliance issues.

---

27. Kubernetes Pod Fails Due to ImagePullBackOff or ErrImagePull (Private Registry Authentication Issues)

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows:

Unset

```
Failed to pull image "<private-registry>/my-app:latest": unauthorized: authentication required

Back-off pulling image "<private-registry>/my-app:latest"
```

This happens when Kubernetes cannot authenticate with a private container registry.

What Needs to Be Analyzed

- Check if the image is from a private registry:

Unset

```
kubectl get pod <pod-name> -o yaml | grep image
```

- Verify if a `Secret` is being used for registry authentication:

Unset

```
kubectl get secret <secret-name> -n <namespace> -o yaml
```

- Inspect the `imagePullSecrets` field in the deployment manifest:

Unset

```
kubectl get deployment <deployment-name> -o yaml | grep -A5
imagePullSecrets
```

- Try manually authenticating to the registry:

Unset

```
docker login <private-registry>
```

How to Resolve Step by Step

1. **Create a Kubernetes Secret for Registry Authentication**

Unset

```
kubectl create secret docker-registry my-registry-secret \

  --docker-server=<private-registry> \

  --docker-username=<your-username> \

  --docker-password=<your-password> \

  --docker-email=<your-email>
```

2. **Attach the Secret to the Pod Spec**
 Modify the `imagePullSecrets` field in the pod or deployment YAML:

Unset

```
spec:

  imagePullSecrets:

    - name: my-registry-secret
```

3. **Test If the Pod Can Pull the Image**
   Restart the pod and check its status:

Unset

```
kubectl delete pod <pod-name>

kubectl get pods
```

4. **Ensure the Correct ServiceAccount Is Used (If Needed)**

Unset

```
kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name":
"my-registry-secret"}]}'
```

Skills Required to Resolve This Issue

- Understanding container image authentication
- Managing Kubernetes secrets
- Debugging registry-related authentication failures

Conclusion

Private container registries require authentication to pull images. Setting up an `imagePullSecret` and attaching it to the pod ensures successful image pulls.

---

28. Kubernetes Pod Fails Due to OOMKilled (Out of Memory Error)

Problem Statement

A pod crashes unexpectedly, and checking its status using `kubectl describe pod <pod-name>` shows:

Unset

```
State:        Terminated
Reason:       OOMKilled
```

This happens when a pod exceeds its allocated memory limits.

What Needs to Be Analyzed

- Check the pod's memory usage:

Unset

```
kubectl top pod <pod-name>
```

- Inspect resource requests and limits:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 resources
```

- View pod logs before termination to identify memory spikes:

Unset

```
kubectl logs <pod-name> --previous
```

How to Resolve Step by Step

1. **Increase Memory Requests and Limits**
   Update the pod spec to allocate more memory:

Unset

```
resources:
  requests:
    memory: "512Mi"

  limits:

    memory: "1Gi"
```

2. **Enable Resource Auto-Scaling (If Needed)**

```
Unset
kubectl autoscale deployment <deployment-name> --min=1 --max=5
--cpu-percent=80
```

3. **Monitor Memory Usage in Real Time**
   Use `kubectl top` or `kubectl logs` to identify memory-intensive operations.

4. **Optimize the Application to Reduce Memory Consumption**

   ○ Identify memory leaks in the application.
   ○ Reduce unnecessary background processes.
   ○ Adjust garbage collection settings in JVM or Python applications.

Skills Required to Resolve This Issue

- Understanding Kubernetes resource management
- Debugging memory-intensive applications
- Optimizing containerized workloads

Conclusion

Pods can crash due to excessive memory usage. Adjusting resource requests and limits, optimizing application memory usage, and enabling auto-scaling can prevent OOMKilled errors.

---

29. Kubernetes Pod Fails Due to Forbidden Mounting of HostPath Volumes

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows an error like:

Unset

```
Error: Forbidden: HostPath volume is not allowed due to security
restrictions
```

This occurs when Kubernetes prevents a pod from mounting a `hostPath` volume due to security policies or PodSecurity Standards (PSS).

What Needs to Be Analyzed

- Check if the pod uses `hostPath` in its volume definitions:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 hostPath
```

- Verify if PodSecurity Standards (PSS) or PodSecurity Admission (PSA) are restricting hostPath usage:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Check if the cluster has a `PodSecurityPolicy (PSP)` restricting `hostPath`:

Unset

```
kubectl get psp -o yaml | grep -A10 hostPath
```

How to Resolve Step by Step

1. **Use an Alternative Volume Type (Recommended Approach)**
   If `hostPath` is not required, use `emptyDir` or `PersistentVolumeClaim (PVC)`:

Unset

```
volumes:

  - name: data-volume

    emptyDir: {}
```

2. **Modify Namespace Security Policies (If Allowed)**
   If the namespace is restricted, change its security label:

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=privileged --overwrite
```

3. **Grant Specific Permissions for `hostPath` Usage**
   If `PodSecurityPolicy` (PSP) is enabled, modify the policy to allow specific paths:

Unset

```
allowedHostPaths:

  - pathPrefix: "/var/log"

    readOnly: false
```

4. **Restart the Pod and Verify Volume Mounting**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes security policies (PSS, PSA, PSP)
- Configuring alternative Kubernetes volume types
- Managing namespace security labels

Conclusion

hostPath is restricted in many Kubernetes environments due to security concerns. Using alternatives like emptyDir or PVC is recommended, while modifying security policies should be done cautiously.

30. Kubernetes Pod Fails Due to Privileged Mode Restrictions

Problem Statement

A pod fails to start, and running kubectl describe pod <pod-name> shows:

Unset
```
Error: PodSecurity "restricted" level forbids privileged containers
```

This happens when a Kubernetes security policy prevents a pod from running in privileged mode.

What Needs to Be Analyzed

- Check if the pod is running in privileged mode:

Unset
```
kubectl get pod <pod-name> -o yaml | grep -A5 privileged
```

- Verify if Pod Security Admission (PSA) is enforcing a restricted policy:

Unset
```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Check if the cluster has a PodSecurityPolicy (PSP) that blocks privileged containers:

Unset

```
kubectl get psp -o yaml | grep -A10 privileged
```

How to Resolve Step by Step

1. **Remove Privileged Mode from the Pod Spec (Recommended Approach)**
   Edit the deployment YAML to disable privileged mode:

Unset

```
securityContext:

  privileged: false
```

2. **Adjust Namespace Security Labels (If Necessary)**
   If the pod requires privileged mode, update the namespace security level:

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=baseline --overwrite
```

3. **Grant Exemptions to Specific ServiceAccounts (If Required)**

Unset

```
kubectl create rolebinding allow-privileged \

  --clusterrole=privileged \

  --serviceaccount=<namespace>:<serviceaccount> \

  --namespace=<namespace>
```

4. **Restart the Pod and Verify Security Compliance**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes security contexts
- Configuring `Pod Security Admission (PSA)` and `PodSecurityPolicy (PSP)`
- Managing RBAC and namespace security settings

Conclusion

Running privileged containers is a security risk, and Kubernetes enforces restrictions by default. Removing privileged mode or granting controlled exemptions ensures security while maintaining functionality.

---

31. Kubernetes Pod Fails Due to ServiceAccount Token Issues

Problem Statement

A pod fails to start or loses access to the Kubernetes API, and running `kubectl describe pod <pod-name>` shows:

Unset

```
Failed to mount service account token

Unauthorized access to the API server
```

This happens when a pod's `ServiceAccount` token is missing, expired, or improperly configured.

What Needs to Be Analyzed

- Check if the pod is using a `ServiceAccount`:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A3 serviceAccount
```

- Verify if the `ServiceAccount` exists:

Unset

```
kubectl get serviceaccount <sa-name> -n <namespace>
```

- Inspect the token secret for the `ServiceAccount`:

Unset

```
kubectl get secret -n <namespace> | grep <sa-name>
```

- Check for missing automount settings:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A2
automountServiceAccountToken
```

How to Resolve Step by Step

1. **Ensure a Valid ServiceAccount Is Used**
   If the `ServiceAccount` is missing, create one:

Unset

```
kubectl create serviceaccount my-serviceaccount -n <namespace>
```

2. **Explicitly Attach the ServiceAccount to the Pod Spec**

Unset

```
spec:
  serviceAccountName: my-serviceaccount
```

3. **Enable Token Mounting (If Disabled)**
   If `automountServiceAccountToken: false` is set, remove it or explicitly set it to `true`:

Unset

```
spec:
  automountServiceAccountToken: true
```

4. **Grant RBAC Permissions If Needed**

Unset

```
kubectl create rolebinding my-sa-rolebinding \
  --clusterrole=view \
  --serviceaccount=<namespace>:my-serviceaccount \
  --namespace=<namespace>
```

5. **Restart the Pod and Verify API Access**

Unset

```
kubectl delete pod <pod-name>
kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes `ServiceAccount` and RBAC

- Debugging token-based authentication failures
- Configuring pod security settings

Conclusion

A missing or misconfigured `ServiceAccount` token can break pod-to-API communication. Ensuring proper token mounting and RBAC permissions restores access securely.

---

32. Kubernetes Pod Fails Due to Seccomp Profile Restrictions

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows:

Unset

```
PodSecurity "restricted" level forbids seccompProfile "unconfined"

Error: seccompProfile is not allowed
```

This happens when a pod's security settings conflict with cluster-wide security policies.

What Needs to Be Analyzed

- Check if `seccompProfile` is defined in the pod spec:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A3 seccompProfile
```

- Verify namespace security settings (`PSA`, `PSS`, or `PSP`):

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify cluster-wide security policy restrictions:

Unset

```
kubectl get psp -o yaml | grep -A5 seccomp
```

How to Resolve Step by Step

1. **Set the Seccomp Profile to a Secure Mode**
   Update the pod spec to use `RuntimeDefault`:

Unset

```
securityContext:
  seccompProfile:
    type: RuntimeDefault
```

2. **Adjust Namespace Security Policies (If Necessary)**
   Modify the namespace label to `baseline` or `privileged`:

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=baseline --overwrite
```

3. **Grant RBAC Exemptions for Seccomp Profile Usage**

Unset

```
kubectl create rolebinding allow-seccomp \
  --clusterrole=seccomp-privileged \
  --serviceaccount=<namespace>:<serviceaccount> \
  --namespace=<namespace>
```

4. **Restart the Pod and Verify Security Compliance**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes security contexts and `seccompProfile`
- Managing namespace security policies (`PSA`, `PSS`, `PSP`)
- Configuring pod-level security settings

Conclusion

Kubernetes enforces `seccomp` restrictions for security. Using `RuntimeDefault` or adjusting security policies ensures pods comply with security standards while running smoothly.

---

33. Kubernetes Pod Fails Due to ReadOnlyRootFilesystem Restriction

Problem Statement

A pod crashes or fails to start, and running `kubectl describe pod <pod-name>` shows an error like:

Unset

```
Error: Read-only file system

Permission denied: cannot write to /tmp
```

This occurs when a security policy enforces a read-only root filesystem, preventing the container from writing to certain paths.

What Needs to Be Analyzed

- Check if the pod is configured with `readOnlyRootFilesystem: true`:

```
Unset
kubectl get pod <pod-name> -o yaml | grep -A5 readOnlyRootFilesystem
```

- Verify if the application attempts to write to a restricted directory:

```
Unset
kubectl logs <pod-name>
```

- Check Kubernetes security policies (`PSA`, `PSS`, `PSP`) that enforce `readOnlyRootFilesystem`:

```
Unset
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

How to Resolve Step by Step

1. **Allow Write Access by Using an Ephemeral Volume**
   Modify the pod spec to mount a `emptyDir` volume at the required path:

```
Unset
volumeMounts:

  - mountPath: /tmp

    name: temp-storage

volumes:

  - name: temp-storage

    emptyDir: {}
```

2. **Update Application Configuration to Use Writable Paths**
   - Change the application's default write location to `/tmp`, `/run`, or another writable directory.
   - Example for an application writing logs:

Unset

```
export LOG_PATH=/tmp/app-logs
```

3. **Disable `readOnlyRootFilesystem` (If Security Policies Allow)**

Unset

```
securityContext:

  readOnlyRootFilesystem: false
```

4. **Modify Namespace Security Label (If Necessary)**

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=baseline --overwrite
```

5. **Restart the Pod and Verify Logs**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes security contexts

- Debugging file system permission issues
- Configuring volumes for persistent and temporary storage

Conclusion

Enforcing a read-only root filesystem improves security, but applications must be adapted to use writable directories or ephemeral storage solutions like `emptyDir`.

---

34. Kubernetes Pod Fails Due to Restricted Sysctl Settings

Problem Statement

A pod fails to start or logs errors like:

Unset

```
Error: Sysctl "net.ipv4.tcp_syncookies" is not allowed
```

This occurs when Kubernetes blocks certain system-level configurations (`sysctl`) due to security policies.

What Needs to Be Analyzed

- Check if the pod is requesting `sysctl` settings:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 sysctls
```

- Verify which `sysctl` settings are allowed by the cluster:

Unset

```
kubectl get psp -o yaml | grep -A10 allowedUnsafeSysctls
```

- Identify if `PodSecurityPolicy` or `PodSecurityAdmission` is blocking `sysctl`:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

How to Resolve Step by Step

1. **Use Safe `sysctl` Settings**
   Modify the pod spec to use only `safe sysctl` parameters:

Unset

```
securityContext:

  sysctls:

    - name: net.core.somaxconn

      value: "1024"
```

2. **Enable Specific `sysctl` Settings via PodSecurityPolicy (If Necessary)**
   If `PSP` is enabled, add allowed `sysctl` settings:

Unset

```
allowedUnsafeSysctls:

  - "net.ipv4.tcp_syncookies"

  - "net.core.somaxconn"
```

3. **Modify Namespace Security Policies (If Allowed)**

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=baseline --overwrite
```

4. **Restart the Pod and Verify Sysctl Settings**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes security policies (`PSA`, `PSP`)
- Configuring safe vs. unsafe `sysctl` settings
- Debugging system-level configurations in Kubernetes

Conclusion

Kubernetes restricts `sysctl` settings for security. Using safe configurations, modifying security policies, or allowing necessary `sysctl` settings can resolve related issues.

---

35. Kubernetes Pod Fails Due to AppArmor Profile Restrictions

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows an error like:

Unset

```
Error: AppArmor profile "unconfined" is not allowed
```

This occurs when Kubernetes security policies enforce mandatory `AppArmor` profiles, preventing the pod from running with an unconfined security setting.

What Needs to Be Analyzed

- Check if the pod specifies an `AppArmor` profile:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A3
apparmor.security.beta.kubernetes.io
```

- Verify which profiles are enforced in the cluster:

Unset

```
kubectl get psp -o yaml | grep -A5 appArmor
```

- Identify if `PodSecurityAdmission` (PSA) restricts `AppArmor`:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

How to Resolve Step by Step

1. **Assign a Default AppArmor Profile (Recommended Approach)**
   Update the pod spec with a valid `AppArmor` profile:

Unset

```
metadata:

  annotations:

    container.apparmor.security.beta.kubernetes.io/my-container:
runtime/default
```

2. **Modify Namespace Security Policies (If Necessary)**
   If security policies restrict `AppArmor`, adjust the namespace settings:

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=baseline --overwrite
```

3. **Create a Custom AppArmor Profile (If Required)**
   If a custom `AppArmor` profile is needed, define it on the node:

Unset

```
sudo apparmor_parser -r /etc/apparmor.d/custom-profile
```

4. **Restart the Pod and Verify Compliance**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding `AppArmor` security profiles
- Managing Kubernetes security policies (`PSA`, `PSP`)
- Configuring pod security annotations

Conclusion

`AppArmor` enhances container security by restricting unauthorized system calls. Assigning a valid profile or adjusting security policies resolves pod startup issues.

---

36. Kubernetes Pod Fails Due to Denied Privilege Escalation

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows:

Unset

```
Error: Privilege escalation is not allowed
```

This happens when Kubernetes security policies prevent containers from escalating privileges (e.g., using `sudo` or `setuid` binaries).

What Needs to Be Analyzed

- Check if the pod requests privilege escalation:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 allowPrivilegeEscalation
```

- Verify namespace security policies (`PSA`, `PSP`):

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify cluster-wide security policies restricting privilege escalation:

Unset

```
kubectl get psp -o yaml | grep -A5 allowPrivilegeEscalation
```

How to Resolve Step by Step

1. **Disable Privilege Escalation (Recommended Approach)**
   Modify the pod spec to ensure `allowPrivilegeEscalation: false`:

Unset

```
securityContext:

  allowPrivilegeEscalation: false
```

2. **Modify Namespace Security Policies (If Necessary)**
   If the application requires privilege escalation, update the namespace settings:

Unset

```
kubectl label namespace <namespace> pod-security.kubernetes.io/enforce=baseline --overwrite
```

3. **Grant Role-Based Access Control (RBAC) Exemptions (If Required)**

Unset

```
kubectl create rolebinding allow-escalation \

  --clusterrole=privileged \

  --serviceaccount=<namespace>:<serviceaccount> \

  --namespace=<namespace>
```

4. **Restart the Pod and Verify Security Compliance**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes `securityContext` settings
- Configuring Kubernetes security policies (PSA, PSP)
- Managing role-based access control (RBAC)

Conclusion

Privilege escalation is blocked for security reasons. Disabling it or adjusting security policies ensures compliance while maintaining functionality.

---

37. Kubernetes Pod Fails Due to Secured Kernel Capabilities Restriction

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows:

Unset

```
Error: Operation not permitted

Capability "NET_ADMIN" is not allowed
```

This occurs when Kubernetes security policies block certain Linux capabilities that the container tries to use.

What Needs to Be Analyzed

- Check if the pod requests special kernel capabilities:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 capabilities
```

- Verify namespace security settings (`PSA`, `PSP`):

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify cluster-wide policies restricting kernel capabilities:

Unset

```
kubectl get psp -o yaml | grep -A5 capabilities
```

How to Resolve Step by Step

1. **Drop Unnecessary Capabilities (Recommended Approach)**
   Modify the pod spec to drop unused capabilities:

Unset

```
securityContext:
  capabilities:
    drop:
      - ALL
```

2. **Allow Specific Capabilities (If Necessary)**
   If an application requires certain capabilities, explicitly allow them:

Unset

```
securityContext:
  capabilities:
    add:
      - NET_ADMIN
```

3. **Adjust Namespace Security Policies (If Needed)**

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=baseline --overwrite
```

4. **Restart the Pod and Verify Functionality**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Linux capabilities (`NET_ADMIN`, `SYS_TIME`, etc.)
- Managing Kubernetes security policies (`PSA`, `PSP`)
- Configuring `securityContext` settings for pods

Conclusion

Dropping unnecessary capabilities improves security. If specific capabilities are required, they should be explicitly allowed while maintaining minimal privileges.

---

38. Kubernetes Pod Fails Due to HostPath Volume Restrictions

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows:

Unset

```
Error: hostPath volumes are not allowed
```

This occurs when a security policy restricts `hostPath` volumes, preventing containers from accessing the host filesystem.

What Needs to Be Analyzed

- Check if the pod uses a `hostPath` volume:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 hostPath
```

- Verify namespace security policies (PSA, PSP):

Unset
```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify cluster-wide policies restricting hostPath:

Unset
```
kubectl get psp -o yaml | grep -A5 volumes
```

How to Resolve Step by Step

1. **Use an Alternative Storage Solution (Recommended Approach)**
   Instead of hostPath, use emptyDir (for ephemeral storage) or PersistentVolume (for long-term storage):

Unset
```
volumes:

  - name: data-storage

    emptyDir: {}
```

2.
   Explicitly Allow hostPath (If Necessary)
   If hostPath is required, define allowed paths in PodSecurityPolicy:

Unset
```
allowedHostPaths:

  - pathPrefix: "/data"

    readOnly: false
```

3. **Adjust Namespace Security Policies (If Needed)**

```
Unset
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=privileged --overwrite
```

4. **Restart the Pod and Verify Storage Access**

```
Unset
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes storage options (`emptyDir`, `PersistentVolume`, `hostPath`)
- Managing Kubernetes security policies (`PSA`, `PSP`)
- Configuring pod security settings

Conclusion

`hostPath` is restricted due to security risks. Using alternative storage solutions like `emptyDir` or `PersistentVolume` is recommended, or security policies must be adjusted carefully.

---

39. Kubernetes Pod Fails Due to Seccomp Profile Restrictions

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows an error like:

Unset

```
Error: Seccomp profile "unconfined" is not allowed
```

This occurs when Kubernetes enforces `seccomp` (secure computing mode) profiles to restrict system calls available to containers.

What Needs to Be Analyzed

- Check if the pod specifies a `seccomp` profile:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A3 seccompProfile
```

- Verify namespace security policies (`PSA`, `PSP`) restricting `seccomp`:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify default `seccomp` settings for the cluster:

Unset

```
kubectl get psp -o yaml | grep -A5 seccomp
```

How to Resolve Step by Step

1. **Use a Default Seccomp Profile (Recommended Approach)**
   Modify the pod spec to use the `runtime/default` profile:

Unset

```
securityContext:

  seccompProfile:
    type: RuntimeDefault
```

2. **Create a Custom Seccomp Profile (If Required)**
   - Define a custom seccomp profile (example: `/var/lib/kubelet/seccomp/profiles/custom.json`):

Unset

```
{
  "defaultAction": "SCMP_ACT_ERRNO",
  "syscalls": [
    {
      "names": ["read", "write", "exit"],
      "action": "SCMP_ACT_ALLOW"
    }
  ]
}
```

   - Apply it to the pod:

Unset

```
securityContext:
  seccompProfile:
    type: Localhost
    localhostProfile: profiles/custom.json
```

3. **Modify Namespace Security Policies (If Necessary)**

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=baseline --overwrite
```

4. **Restart the Pod and Verify Compliance**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding `seccomp` and system call restrictions
- Managing Kubernetes security policies (`PSA`, `PSP`)
- Configuring pod `securityContext` settings

Conclusion

`Seccomp` enhances security by restricting system calls. Using `runtime/default` or a custom profile ensures compliance while maintaining necessary functionality.

---

40. Kubernetes Pod Fails Due to Denied Host Network Access

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows:

Unset

```
Error: Host network is not allowed
```

This happens when Kubernetes security policies prevent containers from using the host network (`hostNetwork: true`).

What Needs to Be Analyzed

- Check if the pod requests host networking:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A3 hostNetwork
```

- Verify namespace security policies (`PSA`, `PSP`) restricting `hostNetwork`:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify cluster-wide policies that restrict host networking:

Unset

```
kubectl get psp -o yaml | grep -A5 hostNetwork
```

How to Resolve Step by Step

1. **Use a Pod Network Instead of Host Network (Recommended Approach)**
   Modify the pod spec to remove `hostNetwork: true` and rely on `ClusterIP` or `NodePort`:

Unset

```
spec:

  hostNetwork: false

  dnsPolicy: ClusterFirst
```

2.
   Explicitly Allow Host Networking (If Necessary)
   If host networking is required, adjust the security settings:

Unset

```
securityContext:

  hostNetwork: true
```

3. **Modify Namespace Security Policies (If Needed)**

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=privileged --overwrite
```

4. **Restart the Pod and Verify Network Connectivity**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes networking (`ClusterIP`, `NodePort`, `hostNetwork`)
- Managing Kubernetes security policies (`PSA`, `PSP`)
- Configuring pod networking settings

Conclusion

`hostNetwork` is restricted for security reasons. Using a pod network is recommended, but if host networking is necessary, security policies must be adjusted.

---

41. Kubernetes Pod Fails Due to Forbidden HostPID Usage

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows:

Unset

```
Error: HostPID is not allowed
```

This occurs when Kubernetes security policies prevent the use of `hostPID: true`, which allows containers to access the host process namespace.

What Needs to Be Analyzed

- Check if the pod requests `hostPID`:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A3 hostPID
```

- Verify namespace security policies (`PSA`, `PSP`) restricting `hostPID`:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify cluster-wide security policies that restrict `hostPID`:

Unset

```
kubectl get psp -o yaml | grep -A5 hostPID
```

How to Resolve Step by Step

1. **Use a Pod-Specific Process Namespace Instead of HostPID (Recommended Approach)**
   Modify the pod spec to remove `hostPID: true`:

Unset

```
spec:

  hostPID: false
```

2. **Explicitly Allow HostPID (If Necessary)**
   If `hostPID` is required (e.g., for monitoring host processes), modify security settings:

Unset

```
securityContext:

  hostPID: true
```

3. **Adjust Namespace Security Policies (If Needed)**

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=privileged --overwrite
```

4. **Restart the Pod and Verify Process Namespace Access**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Linux process namespaces (`PID`, `IPC`, `NET`)
- Managing Kubernetes security policies (`PSA`, `PSP`)
- Configuring pod security settings

Conclusion

`hostPID` is restricted to prevent containerized processes from interfering with host processes. Using isolated process namespaces is recommended, but policies can be adjusted if necessary.

42. Kubernetes Pod Fails Due to Forbidden HostIPC Usage

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows:

Unset

```
Error: HostIPC is not allowed
```

This happens when Kubernetes security policies prevent containers from sharing the host inter-process communication (IPC) namespace.

What Needs to Be Analyzed

- Check if the pod requests `hostIPC`:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A3 hostIPC
```

- Verify namespace security policies (`PSA`, `PSP`) restricting `hostIPC`:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify cluster-wide policies that restrict `hostIPC`:

Unset

```
kubectl get psp -o yaml | grep -A5 hostIPC
```

How to Resolve Step by Step

1. **Use a Pod-Specific IPC Namespace Instead of HostIPC (Recommended Approach)**
   Modify the pod spec to remove `hostIPC: true`:

Unset

```
spec:

  hostIPC: false
```

2. **Explicitly Allow HostIPC (If Necessary)**
   If `hostIPC` is required (e.g., for shared memory applications), modify security settings:

Unset

```
securityContext:

  hostIPC: true
```

3. **Modify Namespace Security Policies (If Needed)**

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=privileged --overwrite
```

4. **Restart the Pod and Verify IPC Namespace Functionality**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Linux inter-process communication (IPC) mechanisms
- Managing Kubernetes security policies (PSA, PSP)
- Configuring pod security settings

Conclusion

hostIPC is restricted for security reasons to prevent containers from accessing shared memory on the host. It should only be enabled when absolutely necessary.

---

43. Kubernetes Pod Fails Due to AppArmor Profile Restrictions

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows an error like:

```
Unset
Error: AppArmor profile "unconfined" is not allowed
```

This happens when a Kubernetes security policy enforces the use of specific AppArmor profiles to restrict process behavior.

What Needs to Be Analyzed

- Check if the pod specifies an AppArmor profile:

```
Unset
kubectl get pod <pod-name> -o yaml | grep -A3
apparmor.security.beta.kubernetes.io
```

- Verify namespace security policies (PSA, PSP) restricting AppArmor:

```
Unset
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify cluster-wide policies enforcing AppArmor restrictions:

Unset

```
kubectl get psp -o yaml | grep -A5 apparmor
```

How to Resolve Step by Step

1. **Use the Default AppArmor Profile (Recommended Approach)**
   Modify the pod spec to use the `runtime/default` profile:

Unset

```
metadata:
  annotations:
    container.apparmor.security.beta.kubernetes.io/<container-name>: runtime/default
```

2. **Create a Custom AppArmor Profile (If Required)**
   - Define a custom AppArmor profile (example: `/etc/apparmor.d/custom-profile`):

Unset

```
profile custom-profile flags=(attach_disconnected, mediate_deleted) {
  # Allow basic operations
  network,
  file,
  capability,
}
```

   - Apply it to the pod:

```
Unset
metadata:

  annotations:

    container.apparmor.security.beta.kubernetes.io/<container-name>:
localhost/custom-profile
```

3. **Modify Namespace Security Policies (If Needed)**

```
Unset
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=baseline --overwrite
```

4. **Restart the Pod and Verify AppArmor Compliance**

```
Unset
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding AppArmor security profiles
- Managing Kubernetes security policies (PSA, PSP)
- Configuring pod security settings

Conclusion

AppArmor enhances security by restricting process behavior. Using `runtime/default` or a custom profile ensures compliance while maintaining necessary functionality.

---

44. Kubernetes Pod Fails Due to Read-Only Filesystem Restriction

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows an error like:

Unset
```
Error: Read-only file system
```

This happens when Kubernetes enforces a read-only root filesystem to enhance security, preventing write operations inside the container.

What Needs to Be Analyzed

- Check if the pod is using a read-only root filesystem:

Unset
```
kubectl get pod <pod-name> -o yaml | grep -A3 readOnlyRootFilesystem
```

- Verify namespace security policies (`PSA`, `PSP`) enforcing read-only filesystems:

Unset
```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify security policies enforcing read-only restrictions:

Unset
```
kubectl get psp -o yaml | grep -A5 readOnlyRootFilesystem
```

How to Resolve Step by Step

1. **Use a Read-Write Volume for Writable Files (Recommended Approach)**
   Modify the pod spec to mount a writable volume for temporary file storage:

Unset
```
volumes:
  - name: temp-storage
    emptyDir: {}
containers:
  - name: app
    volumeMounts:
      - mountPath: /tmp
        name: temp-storage
```

2. **Disable Read-Only Filesystem (If Necessary)**
   If the application needs write access, explicitly disable read-only mode:

Unset

```
securityContext:
  readOnlyRootFilesystem: false
```

3. **Modify Namespace Security Policies (If Needed)**

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=baseline --overwrite
```

4. **Restart the Pod and Verify Write Permissions**

Unset

```
kubectl delete pod <pod-name>
kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes storage (`emptyDir`, `PersistentVolume`)
- Managing Kubernetes security policies (PSA, PSP)
- Configuring pod `securityContext` settings

Conclusion

Enforcing a read-only filesystem improves security. If applications need write access, use writable volumes instead of disabling read-only mode.

---

45. Kubernetes Pod Fails Due to Forbidden Privileged Mode

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows an error like:

Unset

```
Error: Privileged mode is not allowed
```

This occurs when Kubernetes security policies restrict containers from running in **privileged mode**, which grants full access to host resources.

What Needs to Be Analyzed

- Check if the pod is requesting privileged mode:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A3 privileged
```

- Verify namespace security policies (`PSA`, `PSP`) restricting privileged mode:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify PodSecurityPolicies (PSPs) or admission controllers enforcing restrictions:

Unset

```
kubectl get psp -o yaml | grep -A5 privileged
```

How to Resolve Step by Step

1. **Remove Privileged Mode (Recommended Approach)**
   Modify the pod spec to remove `privileged: true`:

Unset

```
securityContext:

  privileged: false
```

2. **Use Specific Capabilities Instead of Privileged Mode**
   If the application needs elevated permissions, grant only necessary capabilities:

Unset

```
securityContext:

  capabilities:

    add:

      - NET_ADMIN

      - SYS_TIME
```

3. **Explicitly Allow Privileged Mode (If Necessary)**
   If privileged mode is required, adjust security settings:

Unset

```
securityContext:

  privileged: true
```

4. However, this should be done cautiously, as it grants root-like access.
5. **Modify Namespace Security Policies (If Needed)**

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=privileged --overwrite
```

6. **Restart the Pod and Verify Privileges**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes security contexts
- Managing PodSecurityPolicies (PSPs) and Pod Security Admission (PSA)
- Configuring Linux capabilities in Kubernetes

Conclusion

Privileged mode should be avoided unless absolutely necessary. Instead, use **capabilities** or fine-tuned security settings to meet the application's requirements.

46. Kubernetes Pod Fails Due to Forbidden Root User Access

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows:

Unset

```
Error: Running as root is not allowed
```

This happens when Kubernetes security policies enforce **non-root user restrictions**, preventing containers from running as UID `0` (root).

What Needs to Be Analyzed

- Check if the pod is running as root:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A3 runAsUser
```

- Verify namespace security policies (`PSA`, `PSP`) enforcing non-root access:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify cluster-wide policies that enforce non-root users:

Unset

```
kubectl get psp -o yaml | grep -A5 runAsNonRoot
```

How to Resolve Step by Step

1. **Run the Container as a Non-Root User (Recommended Approach)**
   Modify the pod spec to explicitly set a non-root user:

Unset

```
securityContext:

  runAsUser: 1000

  runAsGroup: 1000

  fsGroup: 2000
```

2. **Verify the Image Supports Non-Root Execution**
   - Check the default user in the container image:

Unset

```
docker inspect <image-name> | grep User
```

   - If necessary, modify the `Dockerfile` to specify a non-root user:

Unset

```
RUN addgroup --system appgroup && adduser --system --ingroup appgroup appuser

USER appuser
```

3. **Allow Root User (If Absolutely Necessary)**
   If the application requires root access, modify the security settings cautiously:

Unset

```
securityContext:

  runAsNonRoot: false
```

4. **Modify Namespace Security Policies (If Needed)**

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=baseline --overwrite
```

5. **Restart the Pod and Verify the Running User**

Unset

```
kubectl delete pod <pod-name>

kubectl exec -it <pod-name> -- id
```

Skills Required to Resolve This Issue

- Understanding Linux user/group management
- Managing Kubernetes security contexts
- Configuring security policies (`PSA`, `PSP`)

Conclusion

Running containers as non-root users enhances security. If root access is required, it should be explicitly justified and configured with additional security measures.

---

47. Kubernetes Pod Fails Due to Forbidden Host Network Access

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows an error like:

Unset

```
Error: HostNetwork is not allowed
```

This occurs when Kubernetes security policies restrict the use of `hostNetwork: true`, which allows containers to use the host network namespace.

What Needs to Be Analyzed

- Check if the pod is requesting host network access:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A3 hostNetwork
```

- Verify namespace security policies (PSA, PSP) restricting host network usage:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify cluster-wide security policies restricting hostNetwork:

Unset

```
kubectl get psp -o yaml | grep -A5 hostNetwork
```

How to Resolve Step by Step

1. **Use Pod Networking Instead of Host Networking (Recommended Approach)**
   Modify the pod spec to remove hostNetwork: true:

Unset

```
spec:

  hostNetwork: false
```

2. **Use NodePort or LoadBalancer Instead of Host Network (If Possible)**
   - Instead of using hostNetwork, expose the application using a Kubernetes Service:

Unset

```
apiVersion: v1

kind: Service

metadata:

  name: my-service

spec:

  type: NodePort

  ports:

    - port: 80

      targetPort: 8080

      nodePort: 30080

  selector:

    app: my-app
```

3. **Explicitly Allow Host Network (If Absolutely Necessary)**
   If `hostNetwork` is required (e.g., for network monitoring tools), modify security settings:

Unset

```
securityContext:

  hostNetwork: true
```

4. **Modify Namespace Security Policies (If Needed)**

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=privileged --overwrite
```

5. **Restart the Pod and Verify Network Access**

```
Unset
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes networking (`ClusterIP`, `NodePort`, `LoadBalancer`)
- Managing Kubernetes security policies (`PSA`, `PSP`)
- Configuring pod security settings

Conclusion

Using `hostNetwork` bypasses Kubernetes' network isolation, which can introduce security risks. Instead, use Kubernetes `Services` to expose applications securely.

---

48. Kubernetes Pod Fails Due to Forbidden Host Path Volume Usage

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows an error like:

```
Unset
Error: HostPath volume is not allowed
```

This happens when Kubernetes security policies prevent the use of `hostPath` volumes, which allow containers to access files on the host filesystem.

What Needs to Be Analyzed

- Check if the pod is using a `hostPath` volume:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 hostPath
```

- Verify namespace security policies (`PSA`, `PSP`) restricting `hostPath`:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify cluster-wide policies restricting `hostPath`:

Unset

```
kubectl get psp -o yaml | grep -A5 hostPath
```

How to Resolve Step by Step

1. **Use a PersistentVolume Instead of HostPath (Recommended Approach)**
   Replace the `hostPath` volume with a Kubernetes `PersistentVolume`:

Unset

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-storage
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: manual
  hostPath:
    path: "/mnt/data"
```

2.
Use EmptyDir for Temporary Storage (If Possible)
If the application only needs temporary storage, use an `emptyDir` volume:

Unset

```
volumes:
  - name: temp-storage
    emptyDir: {}
containers:
  - name: app
    volumeMounts:
      - mountPath: /tmp
        name: temp-storage
```

3.
Explicitly Allow HostPath (If Absolutely Necessary)
If `hostPath` is required (e.g., for log collection), modify security settings:

Unset

```
volumes:
  - name: host-volume
    hostPath:
      path: "/var/log"

      type: Directory
```

4. **Modify Namespace Security Policies (If Needed)**

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=privileged --overwrite
```

5. **Restart the Pod and Verify Volume Mounting**

Unset

```
kubectl delete pod <pod-name>

kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Kubernetes storage (`PersistentVolumes`, `EmptyDir`)
- Managing Kubernetes security policies (`PSA`, `PSP`)
- Configuring pod security settings

Conclusion

Using `hostPath` volumes can expose the host filesystem to security risks. Instead, use `PersistentVolumes` or `emptyDir` for safe and isolated storage.

---

49. Kubernetes Pod Fails Due to Forbidden SYS_ADMIN Capability

Problem Statement

A pod fails to start, and running `kubectl describe pod <pod-name>` shows an error like:

Unset

```
Error: Capability SYS_ADMIN is not allowed
```

This happens when Kubernetes security policies prevent the use of the **`SYS_ADMIN` capability**, which grants extensive privileges similar to root access.

What Needs to Be Analyzed

- Check if the pod is requesting `SYS_ADMIN` capability:

Unset

```
kubectl get pod <pod-name> -o yaml | grep -A5 capabilities
```

- Verify namespace security policies (`PSA`, `PSP`) restricting capabilities:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify cluster-wide security policies restricting capabilities:

Unset

```
kubectl get psp -o yaml | grep -A5 capabilities
```

How to Resolve Step by Step

1. **Remove the SYS_ADMIN Capability (Recommended Approach)**
   Modify the pod spec to remove `SYS_ADMIN` from the capabilities list:

Unset

```
securityContext:

  capabilities:

    drop:

      - SYS_ADMIN
```

2. **Grant Only the Necessary Capabilities (If Required)**
 If the application needs specific privileges, add only those capabilities:

Unset

```
securityContext:

  capabilities:

    add:

      - NET_ADMIN

      - SYS_TIME
```

3. **Modify Namespace Security Policies (If Needed)**

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=baseline --overwrite
```

4. **Restart the Pod and Verify Capability Changes**

Unset

```
kubectl delete pod <pod-name>
kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Linux capabilities (SYS_ADMIN, NET_ADMIN, SYS_TIME)
- Managing Kubernetes security policies (PSA, PSP)
- Configuring pod security settings

Conclusion

The SYS_ADMIN capability provides extensive privileges and should be avoided. Instead, use minimal required capabilities to maintain security while allowing necessary functionality.

50. Kubernetes Pod Fails Due to Seccomp Profile Restrictions

Problem Statement

A pod fails to start, and running kubectl describe pod <pod-name> shows an error like:

```
Unset
Error: Seccomp profile "unconfined" is not allowed
```

This happens when Kubernetes security policies enforce the use of **restricted Seccomp profiles** to limit system calls.

What Needs to Be Analyzed

- Check if the pod specifies a Seccomp profile:

```
Unset
kubectl get pod <pod-name> -o yaml | grep -A3 seccompProfile
```

- Verify namespace security policies (PSA, PSP) restricting Seccomp:

Unset

```
kubectl get ns <namespace> -o jsonpath='{.metadata.labels}'
```

- Identify cluster-wide security policies enforcing Seccomp restrictions:

Unset

```
kubectl get psp -o yaml | grep -A5 seccompProfile
```

How to Resolve Step by Step

1. **Use the Default Seccomp Profile (Recommended Approach)**
   Modify the pod spec to use the `runtime/default` profile:

Unset

```
securityContext:

  seccompProfile:

    type: RuntimeDefault
```

2.
   Create a Custom Seccomp Profile (If Necessary)
   - Define a custom Seccomp profile (example: `/var/lib/kubelet/seccomp/custom-profile.json`):

Unset

```
{

  "defaultAction": "SCMP_ACT_ERRNO",

  "syscalls": [

    {

      "names": ["read", "write", "exit", "futex"],
      "action": "SCMP_ACT_ALLOW"
    }
  ]
}
```

- ○ Apply it to the pod:

Unset

```
securityContext:
  seccompProfile:
    type: Localhost
    localhostProfile: custom-profile.json
```

3. **Modify Namespace Security Policies (If Needed)**

Unset

```
kubectl label namespace <namespace>
pod-security.kubernetes.io/enforce=baseline --overwrite
```

4. **Restart the Pod and Verify Seccomp Compliance**

Unset

```
kubectl delete pod <pod-name>
kubectl logs <pod-name>
```

Skills Required to Resolve This Issue

- Understanding Seccomp profiles and system call filtering
- Managing Kubernetes security policies (`PSA`, `PSP`)
- Configuring pod security settings

Conclusion

Seccomp restricts system calls to improve security. Using `RuntimeDefault` ensures compliance while maintaining necessary functionality.

→ TRAININGS

WE ARE DIFFERENT

At CareerByteCode, we redefine training by focusing on real-world, hands-on experience. Unlike traditional learning methods, we provide step-by-step implementation guides, 500+ real-time use cases, and industry-relevant projects across cutting-edge technologies like AWS, Azure, GCP, DevOps, AI, FullStack Development and more.

Our approach goes beyond theoretical knowledge—we offer expert mentorship, helping learners understand how to study effectively, close career gaps, and gain the practical skills that employers value.

16+
Years of operations

91+
Countries worldwide

241 K Happy clients

Our Usecases Platform

https://careerbytecode.substack.com

Our WebShop

https://careerbytecode.shop

CareerByteCode
Learning Made simple

CareerByteCode

All in One Platform

STAY IN TOUCH WITH US!

Website

Our WebShop https://careerbytecode.shop
Our Usecases Platform https://careerbytecode.substack.com

E-mail

careerbytec@gmail.com

HQ address

Belgium, Europe

Social Media

@careerbytecode

Phone

+32 471 40 8908

For any RealTime Handson Projects

And for more tips like this

+ Follow

CareerByteCode

Learning Made simple

Like & ReShare

in @careerbytecode