Troubleshooting

ECR Image Pull Permission Issues

Symptom:

Pods fail to start with image pull errors:
```
Failed to pull image "xxxxx.dkr.ecr.region.amazonaws.com/image:tag":
rpc error: code = Unknown desc = Error response from daemon:
Get "https://xxxxx.dkr.ecr.region.amazonaws.com/v2/":
net/http: TLS handshake timeout
```
or
```
Error response from daemon: pull access denied for xxxxx.dkr.ecr.region.amazonaws.com/image
```

Diagnosis:

```bash
# Check pod events for image pull errors
kubectl describe pod <pod-name> -n <namespace> | grep -A 10 "Events:"

# Verify IAM role has ECR permissions
aws iam get-role-policy --role-name <node-role-name> --policy-name <policy-name>
```

Solution:

1. **Verify ECR repository exists and is accessible:**
   ```bash
   aws ecr describe-repositories --repository-names <repo-name> --region <region>
   aws ecr get-login-password --region <region> | docker login --username AWS --password-stdin <account-id>.dkr.ecr.<region>.amazonaws.com
   ```

2. **Configure ECR Repository Policies:**

   ECR repositories require repository policies to explicitly allow IAM roles to pull images. Even if the IAM roles have ECR permissions, the repository policies must also grant access.

   For each ECR repository, add a repository policy:

   ```json
   {
     "Version": "2008-10-17",
     "Statement": [
       {
         "Sid": "ECRImageRetrievalPolicyFromRoles",
         "Effect": "Allow",
         "Principal": {
           "AWS": "arn:aws:iam::<target's account ID>:root"
         },
         "Action": [
           "ecr:BatchGetImage",
           "ecr:GetDownloadUrlForLayer"
         ],
   ```

```
      "Condition": {
        "StringLike": {
          "aws:PrincipalArn": "arn:aws:iam::<target's account ID>:role/*"
        }
      }
    }
  ]
}
```

Note: Replace `<target's account ID>` with the customer's AWS account ID. This policy allows any IAM role in the customer's account to pull images from the repository.

Apply the policies using AWS Console:
1. Navigate to ECR in the customer's account
2. Select the repository
3. Go to "Permissions" tab
4. Click "Edit policy JSON"
5. Add the policy statement above (replace `<target's account ID>` with the customer's AWS account ID)
6. Save changes

Required ECR repositories to configure:
- `classifier-docker-images/classifier_generic_k8s/*`
- `policy-engine-and-autopilot-docker-images/classifier_generic_k8s/*/*`
- Any other repositories used by the deployment

Verify repository policy:
```bash
aws ecr get-repository-policy --repository-name <repo-name> --region <region>
```

3. **Ensure IAM Roles Have ECR Permissions:**

Since images are stored in ECR within the customer's own account, the IAM roles attached to the Kubernetes nodes (master and worker nodes) must have permissions to pull images from ECR.

Required IAM permissions for node roles:
```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:DescribeImages"
      ],
      "Resource": [
```

```
        "arn:aws:ecr:<region>:<account-id>:repository/classifier-docker-images/classifier_generic_k8s/*",
        "arn:aws:ecr:<region>:<account-id>:repository/policy-engine-and-autopilot-docker-images/classifier_generic_k8s/*/*"
      ]
    }
  ]
}
```

Verify IAM role permissions:
```bash
# Get the node role name
kubectl get nodes -o jsonpath='{.items[0].spec.providerID}'  # Shows instance ID

# Find the IAM role attached to the instance
aws ec2 describe-instances --instance-ids <instance-id> --query 'Reservations[0].Instances[0].IamInstanceProfile.Arn'

# Check the role's policies
aws iam list-attached-role-policies --role-name <role-name>
aws iam get-role-policy --role-name <role-name> --policy-name <policy-name>
```

Note: The Terraform configuration should automatically attach the ECR pull policy to the node roles. If images fail to pull, verify that:
- The IAM role is correctly attached to the EC2 instances
- The ECR pull policy includes all required repositories
- The ECR repositories exist in the customer's account
- The ECR credential provider is properly configured on the nodes

4. **Quick verification checklist:**
   - Confirm ECR repositories exist and are accessible (see solution 1 above)
   - Check that ECR repository policies are configured (see solution 2 above)
   - Ensure the IAM role has the required ECR permissions (see solution 3 above for details)
   - Verify the role is attached to the EC2 instances

Terraform Issues

Terraform State Lock Issues

Symptom:

Terraform operations fail with "Error acquiring the state lock".

Solution:

1. **Check if lock exists:**
   ```bash
   aws s3api head-object --bucket <state-bucket> --key <state-key>/.terraform.tfstate.lock.info
   ```

2. **Force unlock (use with caution):**
   ```bash
   terraform force-unlock <lock-id>
   ```

3. **If using S3 backend, manually remove lock:**
   ```bash
   ```

```
aws s3 rm s3://<state-bucket>/<state-key>/.terraform.tfstate.lock.info
```

Kubeconfig Connection Issues

Symptom:

Cannot connect to cluster, getting certificate errors or connection refused.

Solution:

1. Verify the kubeconfig uses the DNS endpoint, not the private IP:
   ```bash
   kubectl config view --minify -o jsonpath='{.clusters[0].cluster.server}'
   # Should show: https://selfhosted-k8s.<domain>:6443
   ```

2. Update kubeconfig to use DNS endpoint:
   ```bash
   kubectl config set-cluster <cluster-name> --server="https://selfhosted-k8s.<domain>:6443" --kubeconfig=~/.kube/privaclave-generic-k8s-kubeconfig
   ```

3. Verify DNS resolution:
   ```bash
   nslookup selfhosted-k8s.<domain>
   nc -zv selfhosted-k8s.<domain> 6443
   ```

Storage Issues (Longhorn)

Pods Stuck in `Init:0/1` or `ContainerCreating` State

Symptom:

Pods cannot start, volumes not attaching.

Diagnosis:
```bash
# Check PVC status
kubectl get pvc -n privaclave-app

# Check volume status in Longhorn
kubectl get volumes.longhorn.io -n longhorn-system

# Check if worker nodes have Longhorn labels
kubectl get nodes -l '!node-role.kubernetes.io/master,!node-role.kubernetes.io/control-plane' --show-labels | grep longhorn
```

Solution:

1. Ensure worker nodes have Longhorn labels:
   ```bash
   kubectl label nodes <worker-node-name> node.longhorn.io/create-default-disk=true --overwrite
   ```

2. Check Longhorn disk status:
   ```bash
   kubectl get nodes.longhorn.io -n longhorn-system -o yaml | grep -A 10 "disk"
```

```

3. Verify volume provisioning:
```bash
kubectl describe volume <volume-name> -n longhorn-system
```

PVCs Stuck in `Terminating` State

Symptom:

PVCs cannot be deleted, stuck in Terminating state.

Solution:

1. Get the associated PV name:
```bash
kubectl get pvc <pvc-name> -n <namespace> -o jsonpath='{.spec.volumeName}'
```

2. Clean up the PV first:
```bash
PV_NAME=$(kubectl get pvc <pvc-name> -n <namespace> -o jsonpath='{.spec.volumeName}')
kubectl patch pv $PV_NAME -p '{"metadata":{"finalizers":[]}}' --type=merge
kubectl delete pv $PV_NAME
```

3. Clean up the PVC:
```bash
kubectl patch pvc <pvc-name> -n <namespace> -p '{"metadata":{"finalizers":[]}}' --type=merge
kubectl delete pvc <pvc-name> -n <namespace>
```

4. Clean up Longhorn volumes if needed:
```bash
kubectl delete volumes.longhorn.io <volume-name> -n longhorn-system
```

Helm Release Issues

Helm Releases Stuck During Deletion

Symptom:

`helm uninstall` hangs or fails with timeout errors.

Solution:

1. Check release status:
```bash
helm list -A
helm get all <release-name> -n <namespace>
```

2. Force delete Helm secrets:
```bash
kubectl delete secret -n <namespace> sh.helm.release.v1.<release-name>.v1
```

3. Clean up resources manually if needed:
```

```bash
# Delete StatefulSets, Deployments, etc.
kubectl delete statefulset <name> -n <namespace>
kubectl delete deployment <name> -n <namespace>
```

4. Use `--no-hooks` flag to skip hooks:
```bash
helm uninstall <release-name> -n <namespace> --no-hooks
```

Namespace Stuck in `Terminating` State

Symptom:

Namespace cannot be deleted, stuck in Terminating state.

Solution:

```bash
# Force namespace deletion by removing finalizers
kubectl get namespace <namespace-name> -o json | jq '.spec.finalizers = []' | kubectl replace --raw "/api/v1/namespaces/<namespace-name>/finalize" -f -
```

Application-Specific Issues

MariaDB Pod Not Starting

Symptom:

MariaDB pod stuck in `Init:0/1` or `ContainerCreating`.

Diagnosis:

```bash
kubectl describe pod mariadb-0 -n privaclave-app
kubectl get pvc data-mariadb-0 -n privaclave-app
kubectl get events -n privaclave-app --sort-by='.lastTimestamp' | grep mariadb
```

Solution:

1. Verify PVC is bound:
```bash
kubectl get pvc data-mariadb-0 -n privaclave-app
```

2. Check volume attachment:
```bash
kubectl describe pod mariadb-0 -n privaclave-app | grep -A 10 "Events:"
```

3. If volume is not ready, clean up and recreate:
```bash
# Delete PVC (StatefulSet will recreate it)
kubectl delete pvc data-mariadb-0 -n privaclave-app
# Delete pod to trigger recreation
kubectl delete pod mariadb-0 -n privaclave-app
```

```

**Vault Pod Not Starting**

**Symptom:**

Vault pod stuck in `ContainerCreating` or initialization fails.

**Solution:**

1. Check if volume is attached:
   ```bash
   kubectl describe pod vault-0 -n privaclave-app | grep -A 10 "Events:"
   ```

2. Verify Longhorn volume status:
   ```bash
   kubectl get volumes.longhorn.io -n longhorn-system | grep vault
   ```

3. If initialization script times out, increase wait time or run manually:
   ```bash
   # Check vault status
   kubectl exec vault-0 -n privaclave-app -c vault -- vault status
   ```

Clean Installation After Issues

Complete Cleanup for Fresh Installation

If you need to start completely fresh:

1. **Clean up Helm releases:**
   ```bash
   helm list -A
   helm uninstall <release-name> -n <namespace> --no-hooks
   ```

2. **Clean up PVCs and PVs:**
   ```bash
   # List all PVCs
   kubectl get pvc -A

   # For each PVC, get PV and clean up
   for pvc in $(kubectl get pvc -n <namespace> -o name); do
     PV_NAME=$(kubectl get $pvc -n <namespace> -o jsonpath='{.spec.volumeName}')
     if [ -n "$PV_NAME" ]; then
       kubectl patch pv $PV_NAME -p '{"metadata":{"finalizers":[]}}' --type=merge
       kubectl delete pv $PV_NAME
     fi
     kubectl patch $pvc -n <namespace> -p '{"metadata":{"finalizers":[]}}' --type=merge
     kubectl delete $pvc -n <namespace>
   done
   ```

3. **Clean up Longhorn volumes:**
   ```bash
   kubectl get volumes.longhorn.io -n longhorn-system
   kubectl delete volumes.longhorn.io <volume-name> -n longhorn-system
   ```
   ```

4. **Clean up namespaces:**
   ```bash
   kubectl get namespaces | grep Terminating
   # Force delete if needed (see namespace section above)
   ```

5. **Remove Terraform state (if needed):**
   ```bash
   terraform state rm <resource-path>
   ```

Getting Help

- Check pod logs: `kubectl logs <pod-name> -n <namespace>`
- Check pod events: `kubectl describe pod <pod-name> -n <namespace>`
- Review initialization logs on nodes: `/var/log/k8s-master-init.log` or `/var/log/k8s-worker-init.log`