Kubernetes ImagePullBackOff Runbook

A senior engineer's incident runbook for a pod stuck `ImagePullBackOff` / `ErrImagePull`. DevOps AI ToolKit · devopsaitoolkit.com

The kubelet couldn't pull the container image, so the pod never starts. The good news: unlike a crash loop, this failure **names its own cause** — `kubectl describe pod` prints the exact registry error the kubelet got back. Your job is to read that line and match it to one of a small, finite set of causes: a bad image reference, an unreachable registry, missing or wrong pull credentials, a TLS/CA problem, or a rate limit. Distinguish the two states you'll see: `ErrImagePull` is the *first* failed attempt; `ImagePullBackOff` is the kubelet backing off (10s, 20s… capped at 5m) after repeated failures. Same root cause — the second just means it's been failing for a while.

> Examples use namespace `prod`, node `node-1`, and image `registry.example.com/myapp:1.4.2`. Substitute your own. Every check here is read-only until the final remediation — read the describe message before you edit a Secret or a manifest.

1. Read the Event — It Names the Cause

```
kubectl describe pod myapp-7d9f8c6b5-abcde -n prod | sed -n '/Events:/,$p'
```

Look at the `Failed` event's message. It is almost always one of:

- `Failed to pull image "...": ... not found` / `manifest unknown` → bad ref or tag (§2).
- `... no such host` / `dial tcp ... i/o timeout` / `connection refused` → registry unreachable from the node (§3).
- `... unauthorized: authentication required` / `denied` → missing or wrong `imagePullSecrets` (§4).
- `... x509: certificate signed by unknown authority` → private-registry TLS/CA (§5).
- `... toomanyrequests` / `You have reached your pull rate limit` → Docker Hub rate limit (§6).

Everything below is just confirming which of these the message already told you.

2. Verify the Exact Image Reference and Tag

```
kubectl get pod myapp-7d9f8c6b5-abcde -n prod \
  -o jsonpath='{range .spec.containers[*]}{.name}: {.image}{"\n"}{end}'
```

- **Typos** in the repo path or a **missing registry host** (`myapp:1.4.2` with no `registry.example.com/`) make Kubernetes default to Docker Hub (`docker.io/library/...`), which then 404s. The host must be explicit for private registries.
- **Tag drift / missing tag:** the manifest references `:1.4.2` but only `:1.4.1` was pushed → `manifest unknown`. Confirm the tag exists in the registry.
- **`:latest` drift:** `imagePullPolicy: Always` re-pulls `:latest` every start; if `latest` was retagged or deleted, a previously-working pod suddenly fails. Pin to an immutable tag or digest.

3. Test Registry Reachability From the Node

```
# Which node is stuck, and is the whole cluster affected or just one node?
kubectl get pod myapp-7d9f8c6b5-abcde -n prod -o wide

# Reach the registry API from inside the cluster (namespace-scoped, disposable)
kubectl run netcheck --rm -it --image=busybox:1.36 -n prod --restart=Never \
  -- sh -c 'nc -zv registry.example.com 443; wget -qO- https://registry.example.com/v2/ || true'
```

- `no such host` → DNS on the node can't resolve the registry (CoreNode DNS, `/etc/hosts`, split-horizon DNS).
- `i/o timeout` / `connection refused` → a network path/firewall/egress-policy problem, or the registry itself is down. If *all* nodes fail identically, suspect the registry or a shared egress rule; if only `node-1`, suspect that node.
- A healthy registry returns HTTP 401 on `/v2/` (auth challenge) — that's *reachable*, and points you at §4, not the network.

4. Check imagePullSecrets and Auth

```
# Does the pod reference a pull secret, and does it exist in THIS namespace?
kubectl get pod myapp-7d9f8c6b5-abcde -n prod \
  -o jsonpath='{.spec.imagePullSecrets[*].name}{"\n"}'
kubectl get secret -n prod
kubectl get secret regcred -n prod -o jsonpath='{.type}{"\n"}'   # want kubernetes.io/dockerconfigjson

# Decode the configured registry host to catch a wrong-registry secret
kubectl get secret regcred -n prod \
  -o jsonpath='{.data.\.dockerconfigjson}' | base64 -d
```

- A pull secret is **namespace-scoped**. A working `regcred` in `default` does nothing for a pod in `prod` — the most common auth miss. Recreate it in the right namespace:

```
kubectl create secret docker-registry regcred -n prod \
  --docker-server=registry.example.com \
  --docker-username=deploy --docker-password='***' \
  --docker-email=ops@example.com
```

- The `--docker-server` must match the image host **exactly** (scheme/port included if used). A secret for `registry.example.com` won't authenticate `registry.example.com:5000`.
- Confirm the pod (or its ServiceAccount) actually references the secret via `imagePullSecrets` — creating it isn't enough.

---

5. Private Registry TLS / CA

```
kubectl describe pod myapp-7d9f8c6b5-abcde -n prod | grep -i "x509\|certificate"

# Inspect what cert the registry presents (from inside the cluster)
kubectl run tlscheck --rm -it --image=nicolaka/netshoot -n prod --restart=Never \
  -- sh -c 'openssl s_client -connect registry.example.com:443 -servername registry.example.com </dev/null 2>/dev/null | openssl x509 -noout -
issuer -subject -dates'
```

- `x509: certificate signed by unknown authority` → the node's container runtime doesn't trust the registry's CA. Fix is to add the CA to the node trust store (containerd: `/etc/containerd/certs.d/registry.example.com/`), not to disable TLS.
- An **expired** cert (`notAfter` in the past) presents the same symptom — check the dates from the `openssl` output.
- This is a node/runtime fix; it usually needs the node owner. Don't paper over it with `insecure` registry config in production.

---

6. Docker Hub Rate Limits and Digest vs Tag

```
kubectl describe pod myapp-7d9f8c6b5-abcde -n prod | grep -i "toomanyrequests\|rate limit"
```

- `toomanyrequests: You have reached your pull rate limit` → anonymous/free Docker Hub pulls are throttled per source IP. Options: add an authenticated Hub `imagePullSecret` (raises the limit), pull through a pull-through cache/mirror, or host the image in your own registry.
- **Digest vs tag:** pinning by digest (`registry.example.com/myapp@sha256:...`) is immutable and safe, but if the digest was garbage-collected or never existed you get `manifest unknown` — re-resolve the digest from a tag that exists.
- **Air-gapped / proxy:** if the node has no direct internet, the image must live in the internal registry and `HTTP(S)_PROXY` / `NO_PROXY` on the runtime must be set correctly. A proxy that doesn't cover the registry host looks exactly like §3 unreachability.

---

7. Decision Tree — Keyed on the describe Message

```
ImagePullBackOff / ErrImagePull → kubectl describe pod → read the Failed event message
  ├─ "not found" / "manifest unknown" ──────── bad image ref, tag, or digest (§2)
  ├─ "no such host" ──────────────────────────── DNS: node can't resolve registry (§3)
  ├─ "i/o timeout" / "connection refused" ──── network/firewall/egress or registry down (§3)
  │     └─ only one node fails? → that node's networking; all nodes? → registry / shared egress
  ├─ "unauthorized" / "denied" ──────────────── imagePullSecret missing/wrong/wrong-namespace (§4)
  ├─ "x509 ... unknown authority" ──────────── registry TLS/CA not trusted by node runtime (§5)
  ├─ "toomanyrequests" / "rate limit" ──────── Docker Hub throttle → authenticate or mirror (§6)
  └─ air-gapped node ─────────────────────────── image not in internal registry / proxy misconfigured (§6)
```

---

8. Copy/Paste One-Shot Triage

```
NS=prod; POD=myapp-7d9f8c6b5-abcde
echo "== image ==";   kubectl get pod $POD -n $NS -o jsonpath='{range .spec.containers[*]}{.name}: {.image}{"\n"}{end}'
echo "== node ==";    kubectl get pod $POD -n $NS -o wide --no-headers
echo "== events ==";  kubectl describe pod $POD -n $NS | sed -n '/Events:/,$p' | grep -Ei "failed|pull|unauthorized|x509|toomanyrequests|no such
host|timeout"
echo "== secret ==";  kubectl get pod $POD -n $NS -o jsonpath='{.spec.imagePullSecrets[*].name}{"\n"}'; kubectl get secret -n $NS | grep -i
dockerconfig
echo "== reach ==";   kubectl run netcheck --rm -it --image=busybox:1.36 -n $NS --restart=Never -- nc -zv registry.example.com 443 2>&1 || true
```

---

9. Incident Notes Template

```
INCIDENT: Kubernetes ImagePullBackOff / ErrImagePull
Started (UTC):        ____   Detected by: ____   Sev: ____
Pod / namespace:      ____ / ____   State: [ ] ErrImagePull (first) [ ] ImagePullBackOff (backing off)
Node(s) affected:     ____   ( [ ] one node  [ ] all nodes )
Image + tag/digest:   ____
describe message:     ____ (the exact "Failed to pull image" line)
Category:             [ ] bad ref/tag  [ ] DNS/network  [ ] auth/pull-secret  [ ] TLS/CA
                      [ ] Docker Hub rate limit  [ ] air-gapped/proxy
Pull secret:          name ____  namespace correct? Y/N  docker-server matches host? Y/N
Action taken:         (fixed ref / recreated secret / added CA / added mirror + who + when)
Result:               [ ] recovered @ ____   [ ] escalated to ____
Root cause:           ____
Follow-ups:           (immutable tags, registry mirror, CA rollout, egress rules, alerts) ____
```
