Administrator Guide — Kubernetes Auto Resource Cleanup (Deployments, Pods & Services)

> Audience: Cluster Administrators (operations / platform owners). This guide explains how the Auto Cleanup tool works, how to deploy and operate it safely, and what administrator actions are required when incidents occur.

---

1. Purpose and Scope

The Auto Cleanup tool automatically frees stale resources in the Kubernetes cluster to preserve compute (GPU/CPU), reduce contention, and keep the DGX environment usable for all users. It enforces time-based policies (soft and hard limits) with administrative controls for exclusions and logging.

This document explains the **operational** steps an administrator needs to take — installation, configuration, scheduling, monitoring, exclusions, and incident responses.

---

2. Key Concepts

- **Hard limit** — strict maximum age (minutes). When a resource reaches this age it is deleted immediately.
- **Soft limit** — earlier age (minutes). When a resource reaches this age it becomes a candidate for deletion; the presence of a `keep-alive: "true"` label prevents deletion under soft-limit conditions.
- **Keep-alive label** — a user-level label (`metadata.labels.keep-alive`) that prevents soft-limit deletes when set to `true`.
- **Exclusions** — administrator-maintained lists that exempt whole namespaces or individual resources from deletion.
- **Batch pod deletion** — the script collects pods that should be deleted and issues batched `kubectl delete pod pod1 pod2 ...` commands so deletions proceed in parallel and the script does not block.
- **Locking** — only one instance of the script runs at a time to prevent cron overlap.
- **Logging** — every action (decision and delete) is logged with timestamps to a centralized log file.

---

3. Files delivered with the tool

Place these files in the same directory on your control node (head node):

- `auto-pod-deletion.sh` — the main script (executable)
- `cleanup_config.env` — runtime configuration
- `exclude_namespaces.txt` — namespaces to skip (one per line)
- `exclude_deployments.txt` — specific deployment names to skip

- `exclude_pods.txt` — specific pod names to skip
- `exclude_services.txt` — specific service names to skip
- `README.md` — user-facing overview (already supplied)

Keep the directory owned by root and readable only by administrators.

---

4. Prerequisites & environment assumptions

- The script runs as a user with permission to delete Deployments, Pods, and Services in the `dgx-*` namespaces.
- A persistent location for logs (e.g. `/var/log/giindia/auto_cleanup_logs/`). Ensure the `LOG_FILE` path in `cleanup_config.env` is writable by the script user.
- Cron or another scheduler for periodic execution.

---

5. Installation and initial setup (administrator steps)

1. Upload the script and config files to the control node (use an admin account):

```
scp auto-pod-deletion.sh cleanup_config.env admin@headnode:/opt/auto-cleanup/
cd ~/auto-cleanup/
sudo chown root:root *
sudo chmod 755 auto-pod-deletion.sh
```

1. Edit `cleanup_config.env` to match your policy (examples and defaults are included in the file). Key variables to set are `STUDENT_SOFT`, `STUDENT_HARD`, `FACULTY_SOFT`, `FACULTY_HARD`, `INDUSTRY_SOFT`, `INDUSTRY_HARD`, and `LOG_FILE`.

2. Create the log directory and ensure permissions:

```
sudo mkdir -p /var/log/giindia/auto_cleanup_logs
sudo chown root:root /var/log/giindia/auto_cleanup_logs
sudo chmod 755 /var/log/giindia/auto_cleanup_logs
```

1. (Optional) Create the exclusion files if you need to protect specific namespaces or resources:

2. `exclude_namespaces.txt` — one namespace per line (example `dgx-admin`)

3. `exclude_deployments.txt` — one deployment name per line
4. `exclude_pods.txt` — one pod name per line

5. `exclude_services.txt` — one service name per line

6. Dry-run validation (recommended):

7. Temporarily set `Pod=False` and `Service=False` in `cleanup_config.env` and run the script to validate logging and scanning only.

8. Check logs for expected scanning behavior. Once validated, enable desired resources and re-run.

---

6. Configuration details (what each key means)

This section is a quick reference for policy-maintenance.

General toggles

- `Deployment`, `Pod`, `Service` — set `True` or `False`. When `False`, that entire resource type is skipped.

Limit feature toggles

- `Deployment_HardLimit`, `Deployment_SoftLimit` — control whether the hard/soft path is active for deployments.
- `Pod_HardLimit`, `Pod_SoftLimit` — control pod behaviors.
- `Service_HardLimit`, `Service_SoftLimit` — control service behaviors.

  If both hard and soft are set to `False` for a resource, the script treats that resource category as disabled.

Time limits (minutes)

- `STUDENT_SOFT`, `STUDENT_HARD` — minutes applied to namespaces matching `dgx-s*`.
- `FACULTY_SOFT`, `FACULTY_HARD` — minutes applied to `dgx-f*`.
- `INDUSTRY_SOFT`, `INDUSTRY_HARD` — minutes applied to `dgx-i*`.

Set values in minutes. Example: `STUDENT_SOFT=120` (2 hours).

Pod deletion performance controls

- `POD_BG_DELETE` (or similar flag in the deployed script) — background deletion toggle. When true the script issues deletes in background and moves on.
- `POD_FORCE_DELETE` — if set, the script uses `--grace-period=0 --force` when deleting pods (use with caution).
- `POD_BATCH_SIZE` — number of pods deleted per `kubectl delete` batch (tune for API server limits).

  The script comes configured with safe defaults. Only change batch size when you experience API throttling or want faster throughput.

---

7. Running the script via cron (recommended production setup)

Edit root crontab with `sudo crontab -e` and add a scheduled job:

```
# Run the cleanup at minute 0 every hour (example)
0 * * * * /bin/bash /opt/auto-cleanup/auto-pod-deletion-final-v3.sh >> /var/
log/giindia/auto_cleanup_cron.log 2>&1
```

Guidelines:

- Keep the schedule longer than the worst-case runtime of the script (default run is short because pods are deleted in background). An hourly schedule is typical.
- Point the cron stderr/stdout to a cron-specific log for quick checks.

---

8. Monitoring & logs

- **Primary Log**: the `LOG_FILE` path from `cleanup_config.env` — contains timestamps and actions.
- **Cron Log**: `/var/log/giindia/auto_cleanup_cron.log` — captures cron stdout/stderr.

What to look for:

- Large numbers of actions in a single run (mass deletions) — indicates an abnormal cleanup event.
- Repeated errors from `kubectl` — might indicate cluster connectivity, RBAC, or permission issues.
- Unprocessed pods stuck in `Terminating` — check node status and finalizers on pods.

Sample troubleshooting commands:

```
# tail last 200 lines of the main log
sudo tail -n 200 /var/log/giindia/auto_cleanup_logs/auto_cleanup.logs

# check for Terminating pods
kubectl get pods -A | grep Terminating

# describe a stuck pod
kubectl describe pod <name> -n <namespace>
```

---

9. Incident handling & admin actions

A. Accidental deletion (common scenarios)

If a resource was deleted and must be recovered:

1. Confirm deletion time and resource identifier from logs.
2. Check if a backup (user-supplied or stored) exists. If none, the resource must be recreated.
3. If multiple deletions happened due to misconfiguration, revert `cleanup_config.env` immediately and re-run the script in Dry-Run mode (set `Pod=False` etc.) to identify remaining candidates.

Note: The administrator cannot reliably recover data from deleted pods — users should store important output in persistent volumes, object storage, or Git.

B. Too many deletions in a single run

1. Immediately disable the offending resource type: update `cleanup_config.env` (set `Pod=False` or `Deployment=False`) and re-run the script once.
2. Inspect logs to identify which policy (soft/hard) triggered deletion.
3. Notify affected users and restore from backups if available.

C. Pods stuck in Terminating

Common causes: finalizers, volume attachments, node unreachable. Steps:

1. `kubectl describe pod <pod> -n <ns>` to inspect finalizers and events.
2. If finalizers prevent deletion and you must force removal, use `kubectl patch pod <pod> -n <ns> -p '{"metadata":{"finalizers":null}}' --type=merge` (use sparingly).
3. Investigate node status: `kubectl get nodes` and `kubectl describe node <node>`.

---

10. Security & RBAC

- The account running the script must have restricted RBAC permissions: `get,list,delete` on Deployments, Pods, and Services only in the target namespaces.
- Do **not** run the script with cluster-admin unless absolutely necessary.
- Protect `cleanup_config.env` and exclusion files with strict permissions (owner=admin, mode=600 or 640).

Recommended minimal RBAC policy (example Role):

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cleanup-role
  namespace: <target-namespace> # or use multiple roles per namespace
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get","list","delete"]
- apiGroups: [""]
  resources: ["pods","services"]
  verbs: ["get","list","delete"]
```

Bind the Role to a ServiceAccount used by the control node if you prefer not to use an admin user.

---

11. Maintenance & best practices

- Review logs daily (or weekly) during initial rollout.
- Communicate to users the required `keep-alive` label and soft/hard limits.
- Maintain a recovery SOP for users (how to persist outputs to PVC / object storage).
- Tune `POD_BATCH_SIZE` if you receive throttling from the API server. Start with 20–50.

---

12. Appendix — Quick reference commands

```
# Dry run: disable deletes in config and run script
# tail logs
sudo tail -f /var/log/giindia/auto_cleanup_logs/auto_cleanup.logs

# Force delete a specific pod immediately
kubectl delete pod <pod> -n <namespace> --grace-period=0 --force

# Show pods stuck in Terminating
kubectl get pods -A | grep Terminating

# Remove finalizers (use with caution)
kubectl patch pod <pod> -n <ns> -p '{"metadata":{"finalizers":null}}' --
type=merge
```

---

13. Document history

- v1.0 — Initial admin guide (aligned with Auto Cleanup v3).

Prepared for cluster administrators. For developer-oriented design & source please reference the README and implementation scripts.