Agentic AI Foundation

AGNTCon + MCPCon Japan

Where the **agentic stack** is **being built**.

Agentic AI Foundation

AGNTCon + MCPCon Japan

Where the **agentic stack** is **being built**.

Letting an Agent Upgrade Production Kubernetes — Without Getting Paged at 3 AM

Abhijeet Chaudhuri, SRE II @ Quartic.ai
Sanskar Agrawalla, SRE II @ Quartic.ai

About Us

Abhijeet Chaudhuri

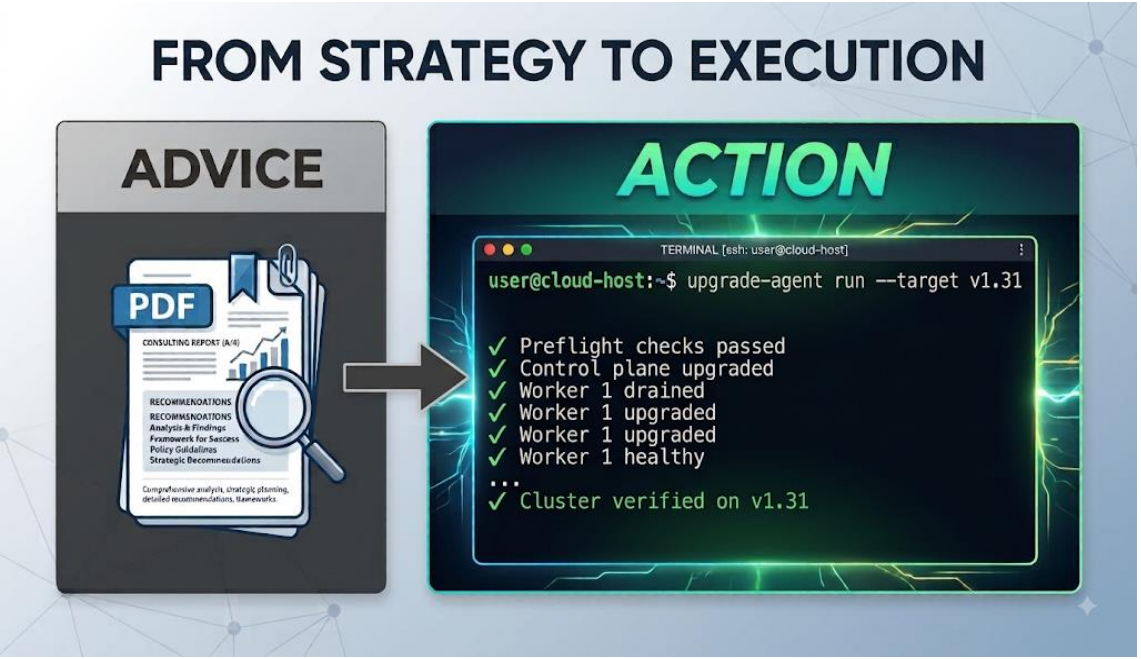
Site Reliability Engineer, runs a production distributed system— where long-running operations are a daily reality, having experience in implementing scalable cloud-native solutions across Kubernetes, AWS, Terraform, Docker, and Jenkins. Skilled in building edge platforms, CI/CD pipelines, GitOps workflows, and observability platforms.

Sanskar Agrawalla

Site Reliability Engineer with a passion for building impactful solutions. Currently, I'm working on cutting-edge projects that drive innovation. From enhancing monitoring processes to optimizing infrastructure, I'm all about crafting efficient and secure systems.

Introduction: Upgrades - Still a Manual Exercise

- **Most of us have upgraded Kubernetes clusters before — but very few would call the process routine.**
- The upgrade command itself is straightforward. The real challenge lies in everything that must be validated **before the first node is touched**.
- Every upgrade can surface incompatibilities across **deprecated APIs, CRDs, webhooks, operators, CSI drivers, and add-ons**.
- This makes Kubernetes upgrades highly repetitive — yet still dependent on **manual assessment, operational experience, and careful judgment.**



We asked a simple question: what if an agent could take on that work — and safely carry the upgrade all the way through?

Agenda

What We'll Explore

- **Why Kubernetes upgrades still page people**
  What makes a routine upgrade surprisingly difficult to automate.
- **What we built — and what we deliberately kept human**
  The boundaries we set before giving an agent access to production.
- **How the system works**
  Assess → Approve → Remediate → Upgrade → Verify
- **What happens under the hood**
  The gate, the decision loop, and the execution engine.
- **What happened in a real upgrade**
  A captured six-node, in-place Kubernetes upgrade — including the failures.
- **What we learned**
  What nearly broke, what we measured, and what we would do differently.

Why upgrades still page people

Reflow: Every quarter, the treadmill restarts

- Kubernetes upgrades are recurring, unavoidable, and unforgiving
- A new minor version lands roughly every three months.
- Patch releases land whenever a CVE demands one.
- Every upgrade silently re-tests your CRDs, webhooks, operators and CSI drivers.
- Repetitive enough to feel routine. Severe enough to become an incident.

This never stops.

A Kubernetes Upgrade Is More Than an Upgrade Command…

Before the upgrade

- Read release notes, deprecations & version compatibility requirements
- Check CRDs, webhooks, operators, CSI/CNI and add-on compatibility
- Take backups and verify they can actually be restored
- Confirm cluster health and capacity
- Decide rollback criteria and the point of no return

During the upgrade

- Upgrade the control plane in the correct sequence
- Cordon and drain workers **one at a time**
- Handle Pod Disruption Budgets, Daemon Sets and stuck evictions
- Upgrade workers while maintaining cluster capacity
- Upgrade and reconcile add-ons

After the upgrade

- Verify node and control-plane health
- Confirm workloads, networking, storage and webhooks are functioning
- Check for deprecated APIs or failing operators
- Monitor the cluster for regressions

What we set out to build

Where the **agentic stack** is **being built**.

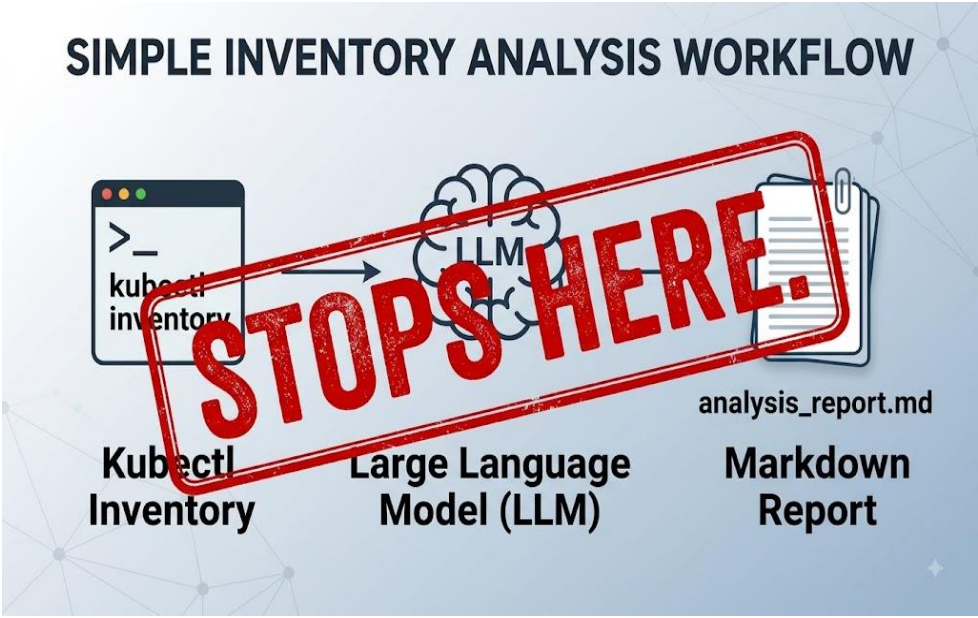
Prior art: **Assessment**

First, Prove the Upgrade Is Safe

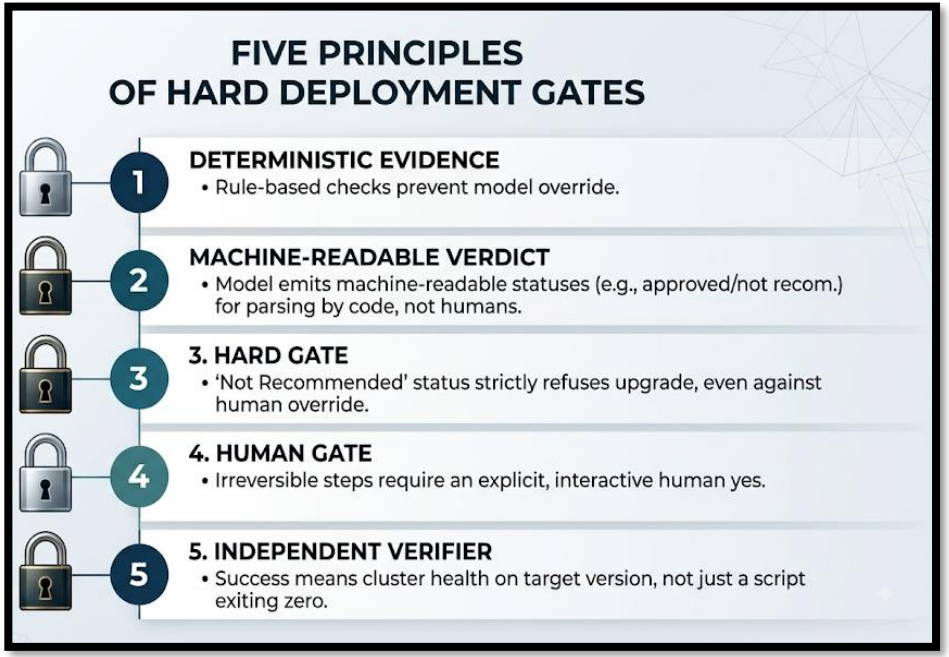
- The agent begins by **reading the cluster and building a complete picture of its upgrade readiness.**
- **Cluster inventory**
  Nodes · workloads · APIs · CRDs · Helm releases · webhooks · operators · storage
- **Compatibility analysis**
  Deprecated APIs · version skew · add-on compatibility · upgrade blockers
- **AI-assisted assessment**
  A capable reasoning model reviews the collected evidence and produces the upgrade recommendation.

We kept that engine and asked a different question:

What would have to be true before we let it execute?



Design Requirements



Before We Let the Agent Execute, Five Rules Had to Hold

- **1. Evidence over opinion**
  Deterministic checks establish the facts the model must work within.
- **2. A machine-readable decision**
  APPROVED · CONDITIONAL · NOT RECOMMENDED
- **3. A hard stop**
  NOT RECOMMENDED means **no execution** — even when asked to proceed.
- **4. Human control at the point of no return**
  Every irreversible action requires explicit approval.
- **5. Independent verification**
  An upgrade is successful only when the **cluster is healthy on the target version**

How the system works
&
What happens under the hood

One Loop. Three Layers. Zero Silent Failures

The loop **stops at the first honest "NO."**

Agent

Drives the workflow, interprets results, and asks for human approval.

MCP Server

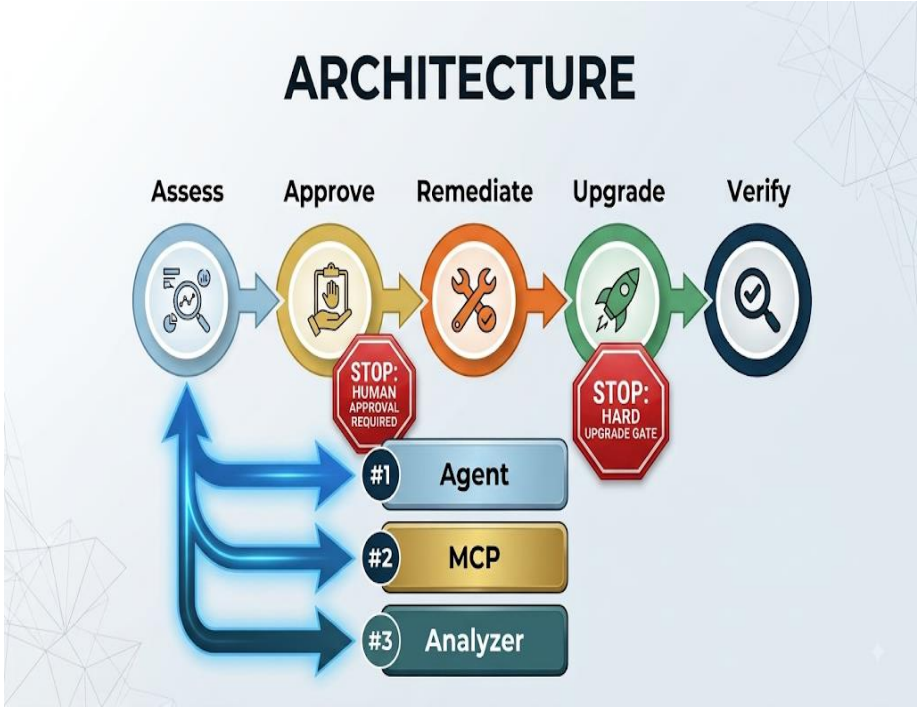
Exposes the capabilities as **gated tools** — not unrestricted kubectl.

Analyzer

Collects cluster state, runs compatibility checks, and produces the evidence the decision depends on.

The key idea……..

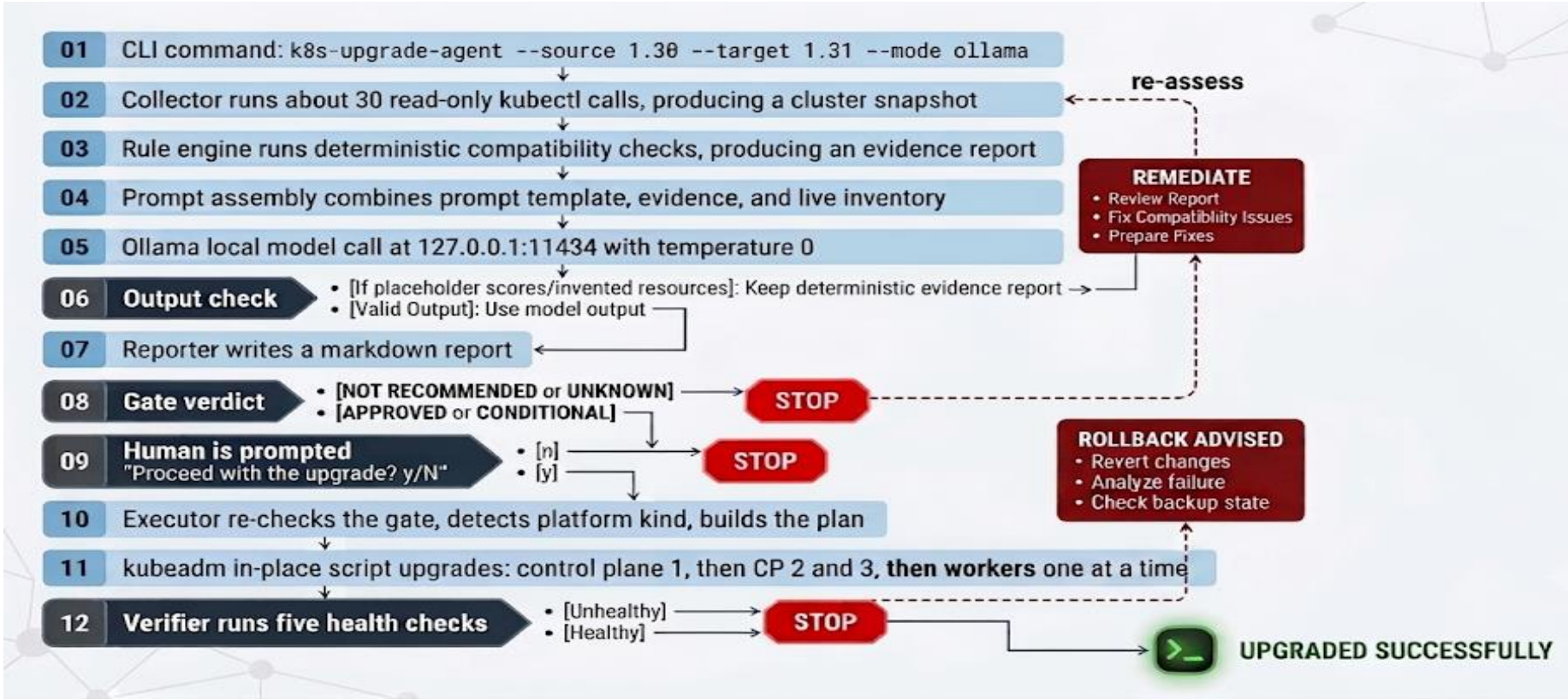
- **The agent decides what to do next.**
- **The system decides what it is allowed to do.**



Meet The Components

Stage	Role	Where it lives	What it does
ASSESS	The surveyor	collector/	~30 read-only kubectl calls → an inventory of the live cluster
	The rule book	analyzer/	deterministic compatibility checks over that inventory
	The assessor	ollama/ claude/ gpt/ + prompts/	explains the evidence, under a strict output contract
	The scribe	reporter/	writes the verdict to reports/*.md
APPROVE	The gatekeeper	gates.py	parses the verdict — allows or refuses execution
REMEDIATE	The fixer	remediate.py	turns blockers into a fix plan, dry-run by default
UPGRADE	The scout	platform.py	identifies the cluster type — kind, kops, EKS, GKE, AKS
	The mechanic	upgrade.py + scripts/kind_inplace_upgrade.sh	performs the upgrade, one node at a time
VERIFY	The inspector	verify.py	five health checks after execution
ALL FIVE	The driver	agent/	runs the loop end to end and asks the human
	The service window	mcp_server/	Lets an AI assistant run the same assess → approve → upgrade → verify loop, without giving it kubectl
	The paper trail	reports/ plans/	every run leaves evidence on disk

How the pieces call each other



What happened in a real upgrade

Cluster and AI Model

1. We used kind cluster for our POC
2. For AI Model currently we only support, Cursor, Ollama and GPT
3. For our demo we are Running Ollama local in the VM only

```
ubuntu@ip-10-0-90-244:~$ kind create cluster \
  --name upgrade-demo \
  --image kindest/node:v1.30.0 \
  --config=- <<EOF
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4

nodes:
- role: control-plane
- role: control-plane
- role: control-plane
- role: worker
- role: worker
- role: worker
EOF
Creating cluster "upgrade-demo" ...
 ✓ Ensuring node image (kindest/node:v1.30.0) 🖼
 ✓ Preparing nodes 📦 📦 📦 📦 📦 📦
 ✓ Configuring the external load balancer ⚖️
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
 ✓ Joining more control-plane nodes 🎮
 ✓ Joining worker nodes 🚜
Set kubectl context to "kind-upgrade-demo"
You can now use your cluster with:

kubectl cluster-info --context kind-upgrade-demo

Not sure what to do next? 😅  Check out https://kind.sigs.k8s.io/docs/user/quick-start/
```

```
ubuntu@ip-10-0-90-244:/etc/init.d$ curl -fsSL https://ollama.com/install.sh | sh      ^C
ubuntu@ip-10-0-90-244:/etc/init.d$ ollama serve      ^C
ubuntu@ip-10-0-90-244:/etc/init.d$ systemctl status ollama
● ollama.service - Ollama Service
     Loaded: loaded (/etc/systemd/system/ollama.service; enabled; preset: enabled)
     Active: active (running) since Tue 2026-08-04 06:58:31 UTC; 1 week 0 days ago
   Main PID: 89244 (ollama)
      Tasks: 10 (limit: 19156)
     Memory: 2.5G (peak: 9.8G)
        CPU: 1h 46min 1.605s
     CGroup: /system.slice/ollama.service
             └─89244 /usr/local/bin/ollama serve

Aug 11 12:17:05 ip-10-0-90-244 ollama[89244]: slot print_timing: id  0 | task 0 |       total time =  216313.61 ms /  2124 tokens
Aug 11 12:17:05 ip-10-0-90-244 ollama[89244]: slot print_timing: id  0 | task 0 |    graphs reused =        246
Aug 11 12:17:05 ip-10-0-90-244 ollama[89244]: slot      release: id  0 | task 0 | stop processing: n_tokens = 2123, truncated = 0
Aug 11 12:17:05 ip-10-0-90-244 ollama[89244]: srv  update_slots: all slots are idle
Aug 11 12:17:05 ip-10-0-90-244 ollama[89244]: [GIN] 2026/08/11 - 12:17:05 | 200 |         3m48s |       127.0.0.1 | POST     "/api/chat"
Aug 11 16:22:29 ip-10-0-90-244 ollama[89244]: time=2026-08-11T16:22:29.988Z level=INFO source=model_recommendations.go:177 msg="model recommendations cache sleep scheduled" wait=3h33m29.688784654s consecutive_fail>
Aug 11 17:34:07 ip-10-0-90-244 ollama[89244]: [GIN] 2026/08/11 - 17:34:07 | 200 |     2.03232ms |       127.0.0.1 | HEAD     "/"
Aug 11 17:34:07 ip-10-0-90-244 ollama[89244]: [GIN] 2026/08/11 - 17:34:07 | 200 |    1.944189ms |       127.0.0.1 | GET      "/api/tags"
Aug 11 17:39:19 ip-10-0-90-244 ollama[89244]: [GIN] 2026/08/11 - 17:39:19 | 200 |      28.642µs |       127.0.0.1 | HEAD     "/"
Aug 11 17:39:19 ip-10-0-90-244 ollama[89244]: [GIN] 2026/08/11 - 17:39:19 | 200 |     510.526µs |       127.0.0.1 | GET      "/api/tags"
lines 1-20/20 (END)
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ubuntu-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ubuntu-writer
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ubuntu-writer
  template:
    metadata:
      labels:
        app: ubuntu-writer
    spec:
      containers:
      - name: ubuntu
        image: ubuntu:24.04
        command:
          - /bin/bash
          - -c
          - |
            apt-get update &&
            apt-get install -y coreutils &&
            while true; do
              echo "Hello World $(date '+%Y-%m-%d %H:%M:%S %Z')" >> /data/hello.log
              sync
              sleep 10
            done
        volumeMounts:
        - name: data
          mountPath: /data
      volumes:
      - name: data
        persistentVolumeClaim:
          claimName: ubuntu-pvc
```

Deploy sample Pod and PVC

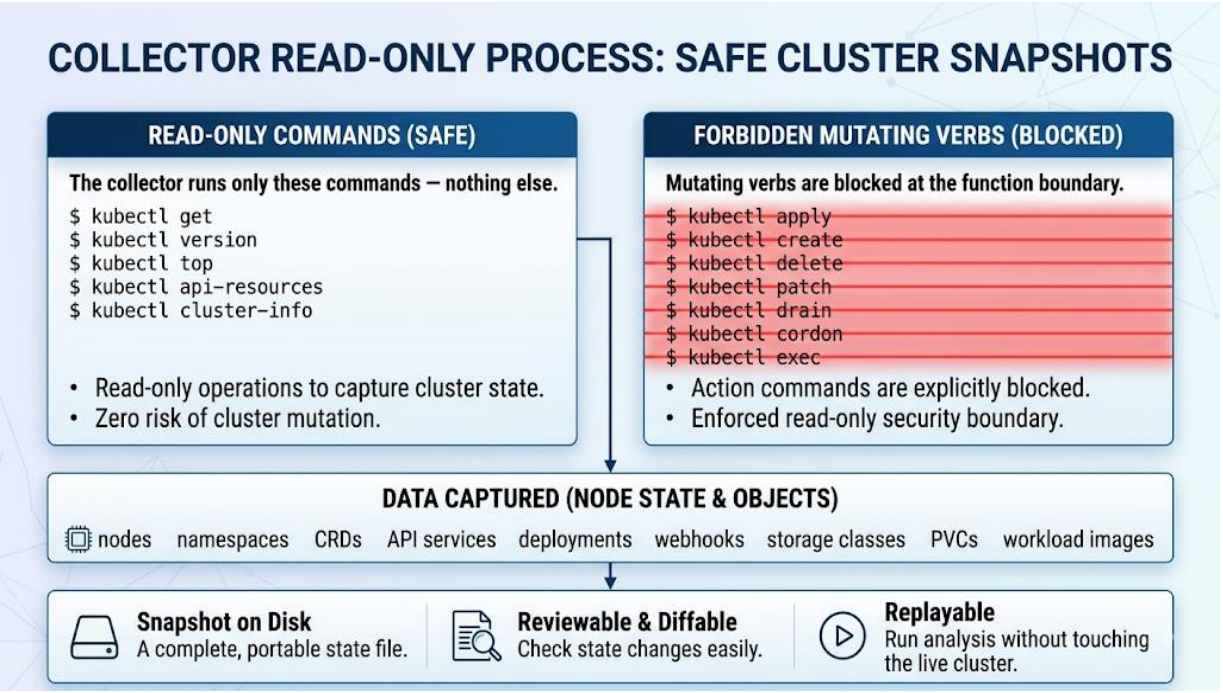
```
(.venv) ubuntu@ip-10-0-90-244:~/kubernetes-upgrade/k8s_upgrade_analyzer$ k8s-upgrade-agent --source 1.30 --target 1.31 --mode ollama --model llama3.1:8b --execute-upgrade

K8s Upgrade Agent
1.30 → 1.31
mode=ollama model=llama3.1:8b
flow: AI assess → yes/no → in-place upgrade → verify

AI assessment (mode=ollama, model=llama3.1:8b)
──────────────────────────────── reports/cluster-upgrade-feasibility-and-risks_1_30_to_1_31.md ────────────────────────────────
<!-- generated_at=2026-08-11T17:57:16Z mode=ollama source=1.30 target=1.31 -->
# Kubernetes Upgrade Feasibility (evidence)
```

Run the Agent

ASSESS: Read-only collector



Before the Agent Can Act, It Can Only Observe

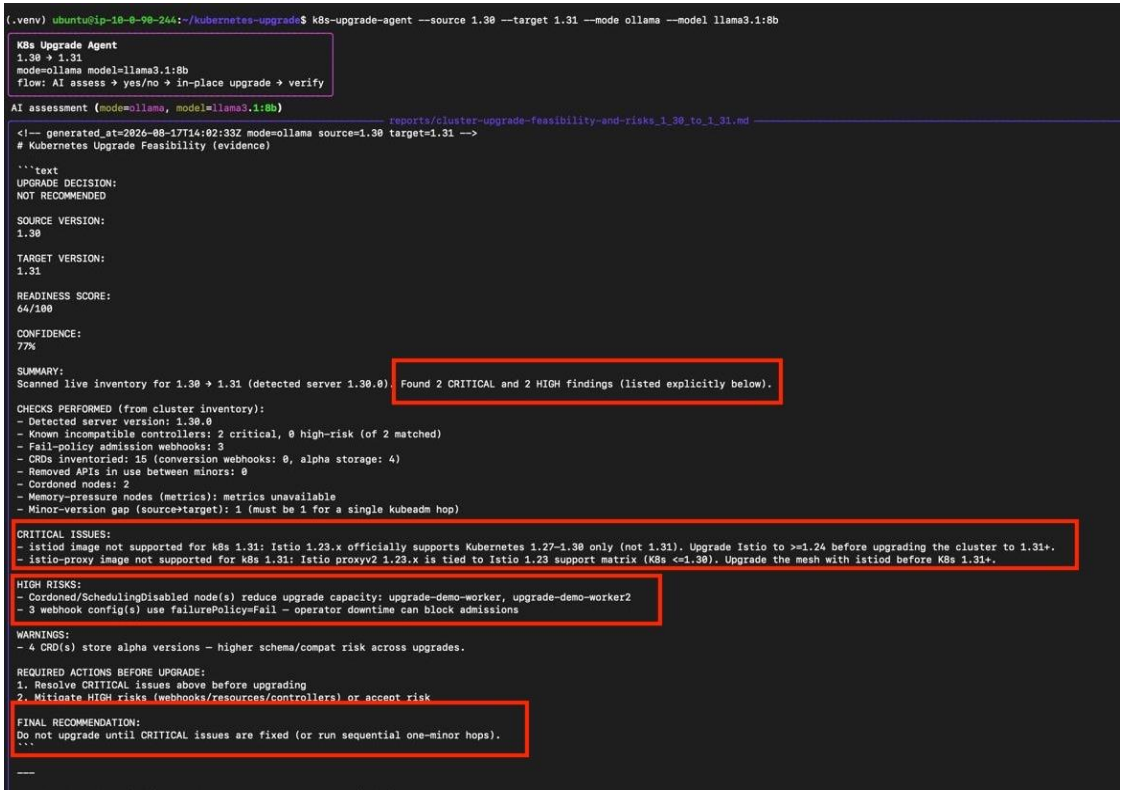
- The collector runs kubectl get, version, top, api-resources, and cluster-info — nothing else.
- Mutating verbs are **blocked** at the function boundary: apply, create, delete, patch, drain, cordon, exec, rollout.
- Captures nodes, namespaces, CRDs, API services, deployments, webhooks, storage classes, PVCs, and workload images.

Output

A point-in-time cluster snapshot
Reviewable · Diffable · Replayable

ASSESS: **Deterministic Checks First**

- Plain Python rules run before any model is called.
- What it checks:
  - removed APIs in use
  - incompatible controllers
  - fail-closed webhooks, CRD conversion webhooks
  - minor-version gap
- Vendor support matrices encoded as rules — ingress-nginx, Rancher, cert-manager, KEDA, Prometheus Operator
- Output: counted checks, not prose
- Unverified compatibility is treated as risk, not as silence



APPROVE: The gate

The Model Can Recommend. It Cannot Authorize.

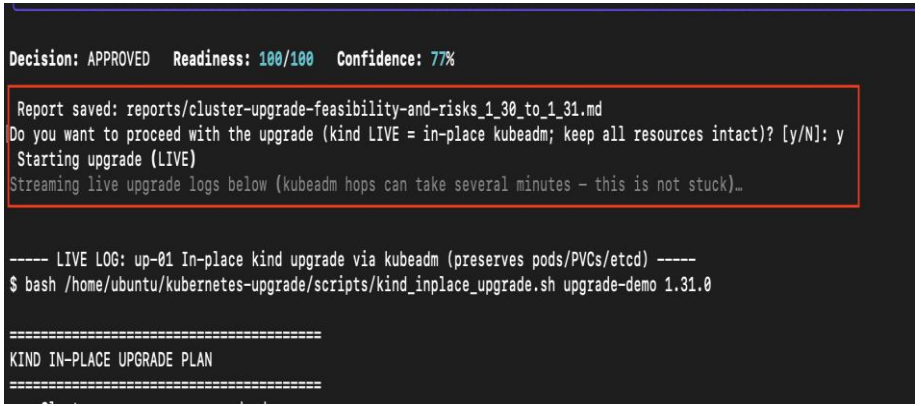
The assessment is stored as a report and is parsed by code into **four states**:

- **APPROVED**
  Ready to proceed — with human approval.
- **CONDITIONAL**
  Requires remediation or narrowly scoped risk acceptance.
- **NOT RECOMMENDED**
  Execution is blocked.
- **UNKNOWN**
  Unparseable or incomplete output is **also blocked.**

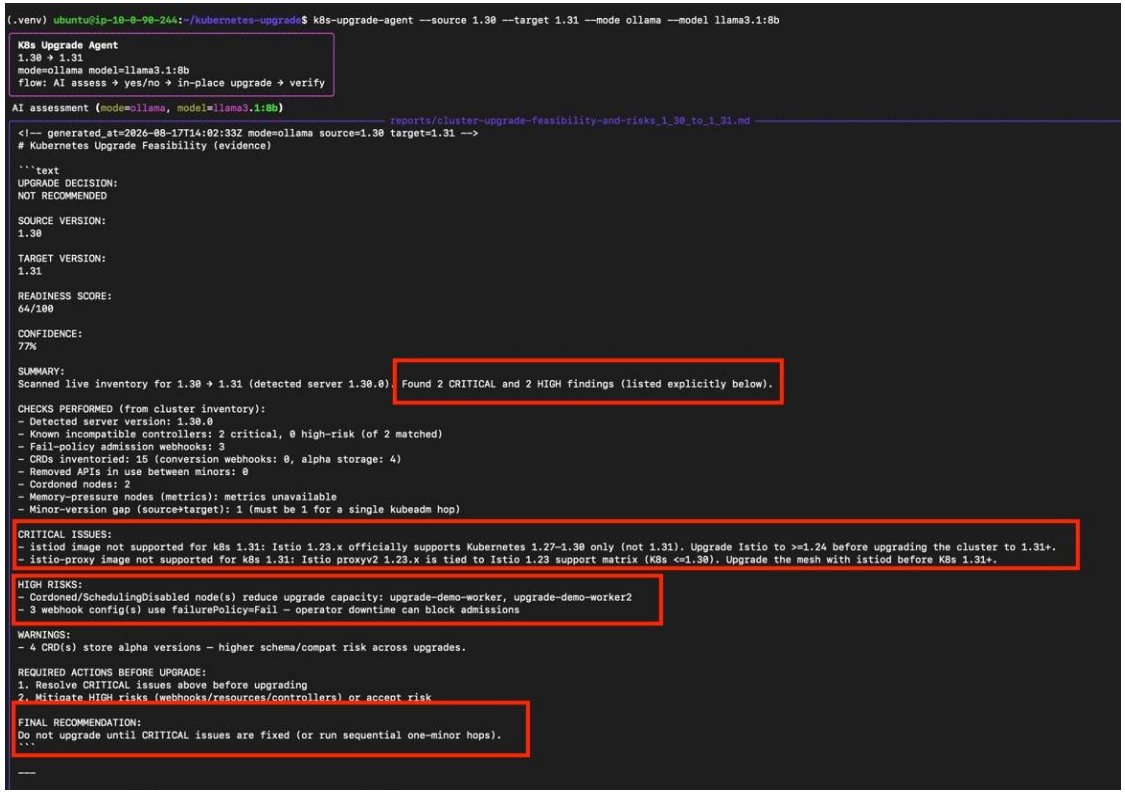
Then comes the hard stop.

The agent pauses and prompts- **"Proceed with the upgrade? [y/n]"**

- Only an explicit **y or n** continues the workflow.



REMEDIATE · Fix, re-check, then upgrade



A Failed Assessment Is Not a Dead End.

When the assessment finds a blocker:

- **IDENTIFY**
  Understand the incompatibility
  ↓
- **REMEDIATE**
  Manually fix the resource / configuration
  ↓
- **RE-ASSESS**
  Run the same checks again
  ↓
- **PROCEED**
  Only when the readiness decision changes

No bypass. No “just continue.”

Every remediation must earn its way back through the gate.

Upgrade: In-Place, One Minor at a Time

The Upgrade Follows Kubernetes' Rules — Not the Agent's Shortcut

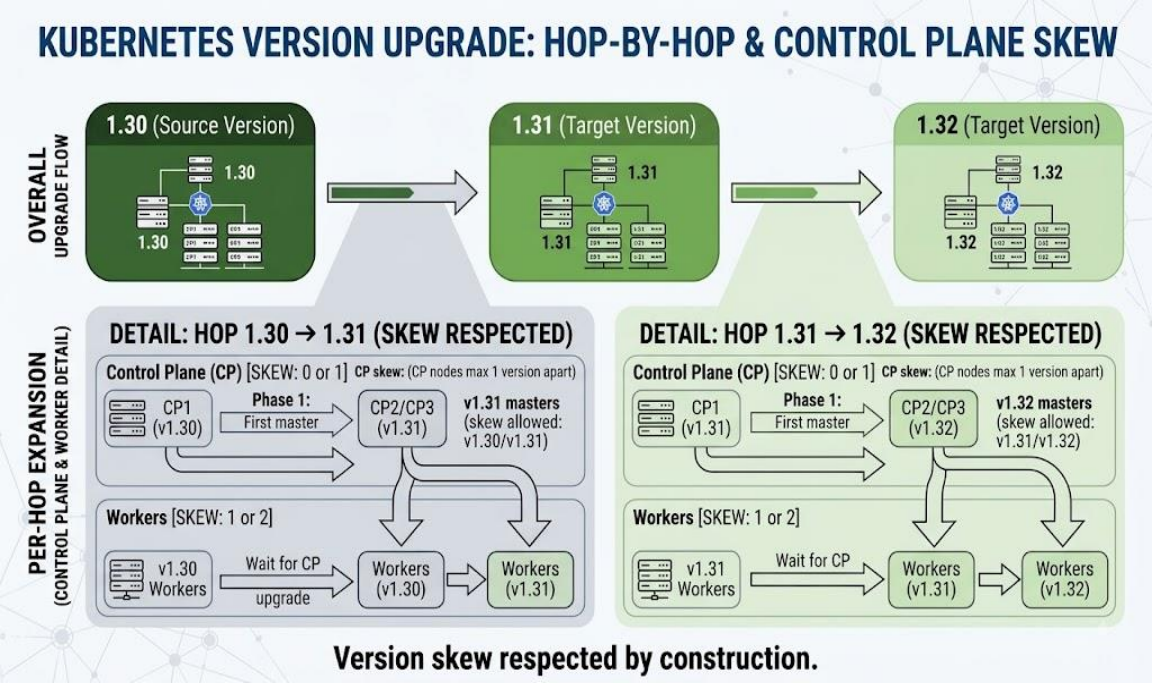
Kubernetes requires single-minor version hops.

For each hop:

- Pin the required kubeadm version
- Upgrade the first control plane
- Upgrade remaining control planes
- Upgrade workers one at a time
- Restart and verify kubelet on every node

Nothing is Recreated.

etcd · Pods · Persistent Volume Claims survive the upgrade.



UPGRADE · Control planes first, then workers

One Node at a Time. One Verified Transition at a Time.

CONTROL PLANE

First control plane → API recovery → Remaining control planes

WORKERS

Cordon → Drain → Upgrade kubelet → Uncordon → Verify

The agent never advances on assumption.
Node upgraded → Version verified → Health confirmed → Next node

```
ubuntu@ip-10-0-90-244:~$ kubectl get nodes
NAME                        STATUS   ROLES           AGE   VERSION
upgrade-demo-control-plane  Ready    control-plane   32m   v1.31.14
upgrade-demo-control-plane2 Ready    control-plane   32m   v1.30.0
upgrade-demo-control-plane3 Ready    control-plane   31m   v1.30.0
upgrade-demo-worker         Ready    <none>          31m   v1.30.0
upgrade-demo-worker2        Ready    <none>          31m   v1.30.0
upgrade-demo-worker3        Ready    <none>          31m   v1.30.0
ubuntu@ip-10-0-90-244:~$
```

Upgrading Control-planes first

```
ubuntu@ip-10-0-90-244:~$ kubectl get nodes
NAME                       STATUS                     ROLES           AGE   VERSION
upgrade-demo-control-plane    Ready                      control-plane   37m   v1.31.14
upgrade-demo-control-plane2   Ready                      control-plane   37m   v1.31.14
upgrade-demo-control-plane3   Ready                      control-plane   37m   v1.31.14
upgrade-demo-worker           Ready                      <none>          37m   v1.31.14
upgrade-demo-worker2          Ready,SchedulingDisabled   <none>          37m   v1.30.0
upgrade-demo-worker3          Ready                      <none>          37m   v1.30.0
ubuntu@ip-10-0-90-244:~$
```

```
=====================================
WORKER NODE: upgrade-demo-worker3 → v1.31.x
=====================================
==> drain upgrade-demo-worker3
node/upgrade-demo-worker3 cordoned
Warning: ignoring DaemonSet-managed Pods: kube-system/kindnet-skgss, kube-system/kube-proxy-5ljz9
evicting pod kube-system/coredns-7c65d6cfc9-2p5rv
evicting pod default/ubuntu-writer-59d5bbcd68-vwrbq
pod/coredns-7c65d6cfc9-2p5rv evicted
pod/ubuntu-writer-59d5bbcd68-vwrbq evicted
node/upgrade-demo-worker3 drained
==> [upgrade-demo-worker3] install kubeadm pinned to v1.31.*
```

Upgrading Worker Nodes

kubectl get nodes

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME
upgrade-demo-control-plane	Ready	control-plane	39m	v1.31.14	172.18.0.6	<none>	Debian GNU/Linux 12 (bookworm)	6.17.0-1017-aws	containerd://1.7.15
upgrade-demo-control-plane2	Ready	control-plane	38m	v1.31.14	172.18.0.7	<none>	Debian GNU/Linux 12 (bookworm)	6.17.0-1017-aws	containerd://1.7.15
upgrade-demo-control-plane3	Ready	control-plane	38m	v1.31.14	172.18.0.3	<none>	Debian GNU/Linux 12 (bookworm)	6.17.0-1017-aws	containerd://1.7.15
upgrade-demo-worker	Ready	<none>	38m	v1.31.14	172.18.0.5	<none>	Debian GNU/Linux 12 (bookworm)	6.17.0-1017-aws	containerd://1.7.15
upgrade-demo-worker2	Ready	<none>	38m	v1.31.14	172.18.0.4	<none>	Debian GNU/Linux 12 (bookworm)	6.17.0-1017-aws	containerd://1.7.15
upgrade-demo-worker3	Ready	<none>	38m	v1.31.14	172.18.0.8	<none>	Debian GNU/Linux 12 (bookworm)	6.17.0-1017-aws	containerd://1.7.15

pods/pvc

/kube-proxy-5ljz9		1/1	Running	1 (4s ago)	2m36s
kube-system	pod/kube-proxy-8hszx	1/1	Running	1 (66s ago)	2m33s
kube-system	pod/kube-proxy-b7t22	1/1	Running	0	2m30s
kube-system	pod/kube-proxy-nnb2b	1/1	Running	1 (2m25s ago)	2m29s
kube-system	pod/kube-proxy-pkwv6	1/1	Running	0	2m41s
kube-system	pod/kube-proxy-xc8pb	1/1	Running	1 (104s ago)	2m39s
kube-system	pod/kube-scheduler-upgrade-demo-control-plane	1/1	Running	1 (7m39s ago)	7m39s
kube-system	pod/kube-scheduler-upgrade-demo-control-plane2	1/1	Running	1 (5m31s ago)	5m31s
kube-system	pod/kube-scheduler-upgrade-demo-control-plane3	1/1	Running	1 (2m25s ago)	2m26s
local-path-storage	pod/local-path-provisioner-988d74bc-fgshm	1/1	Running	1 (7m9s ago)	39m

NAMESPACE	NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	VOLUMEATTRIBUTESCLASS	AGE
default	persistentvolumeclaim/ubuntu-pvc	Bound	pvc-ae3b0333-e828-4fdf-a527-e4a9bfe25097	1Gi	RWO	standard	<unset>	37m

Trace: /home/ubuntu/kubernetes-upgrade/plans/agent_trace_1_30_to_1_31.json

```
UPGRADED SUCCESSFULLY
Cluster is on target 1.31.
Existing pods/PVCs were preserved (in-place upgrade).
```

Upgrade completed


```
----- LIVE LOG: up-02 Verify nodes Ready on target version -----
$ kubectl get nodes -o wide
NAME                          STATUS   ROLES           AGE   VERSION    INTERNAL-IP    EXTERNAL-IP   OS-IMAGE                         KERNEL-VERSION     CONTAINER-RUNTIME
upgrade-demo-control-plane    Ready    control-plane   39m   v1.31.14   172.18.0.6     <none>        Debian GNU/Linux 12 (bookworm)   6.17.0-1017-aws   containerd://1.7.15
upgrade-demo-control-plane2   Ready    control-plane   38m   v1.31.14   172.18.0.7     <none>        Debian GNU/Linux 12 (bookworm)   6.17.0-1017-aws   containerd://1.7.15
upgrade-demo-control-plane3   Ready    control-plane   38m   v1.31.14   172.18.0.3     <none>        Debian GNU/Linux 12 (bookworm)   6.17.0-1017-aws   containerd://1.7.15
upgrade-demo-worker           Ready    <none>          38m   v1.31.14   172.18.0.5     <none>        Debian GNU/Linux 12 (bookworm)   6.17.0-1017-aws   containerd://1.7.15
upgrade-demo-worker2          Ready    <none>          38m   v1.31.14   172.18.0.4     <none>        Debian GNU/Linux 12 (bookworm)   6.17.0-1017-aws   containerd://1.7.15
upgrade-demo-worker3          Ready    <none>          38m   v1.31.14   172.18.0.8     <none>        Debian GNU/Linux 12 (bookworm)   6.17.0-1017-aws   containerd://1.7.15
----- END LOG: up-02 (exit=0) -----

 Upgrade execution finished.
   up-01 In-place kind upgrade via kubeadm (preserves pods/PVCs/etcd)
      bash /home/ubuntu/kubernetes-upgrade/scripts/kind_inplace_upgrade.sh upgrade-demo 1.31.0
      ==> Hop OK — ALL nodes (control-plane + workers) are on v1.31.x
      UPGRADED SUCCESSFULLY → v1.31.14
   up-02 Verify nodes Ready on target version
      kubectl get nodes -o wide
 Checking nodes/workloads for successful upgrade…
 ── kubectl get nodes ──
 NAME                          STATUS   ROLES           AGE   VERSION    INTERNAL-IP    EXTERNAL-IP   OS-IMAGE                         KERNEL-VERSION     CONTAINER-RUNTIME
 upgrade-demo-control-plane    Ready    control-plane   39m   v1.31.14   172.18.0.6     <none>        Debian GNU/Linux 12 (bookworm)   6.17.0-1017-aws   containerd://1.7.15
 upgrade-demo-control-plane2   Ready    control-plane   38m   v1.31.14   172.18.0.7     <none>        Debian GNU/Linux 12 (bookworm)   6.17.0-1017-aws   containerd://1.7.15
 upgrade-demo-control-plane3   Ready    control-plane   38m   v1.31.14   172.18.0.3     <none>        Debian GNU/Linux 12 (bookworm)   6.17.0-1017-aws   containerd://1.7.15
 upgrade-demo-worker           Ready    <none>          38m   v1.31.14   172.18.0.5     <none>        Debian GNU/Linux 12 (bookworm)   6.17.0-1017-aws   containerd://1.7.15
 upgrade-demo-worker2          Ready    <none>          38m   v1.31.14   172.18.0.4     <none>        Debian GNU/Linux 12 (bookworm)   6.17.0-1017-aws   containerd://1.7.15
 upgrade-demo-worker3          Ready    <none>          38m   v1.31.14   172.18.0.8     <none>        Debian GNU/Linux 12 (bookworm)   6.17.0-1017-aws   containerd://1.7.15
 ── pods/pvc ──
 /kube-proxy-5ljz9                                    1/1     Running   1 (4s ago)     2m36s
 kube-system          pod/kube-proxy-8hszx                                1/1     Running   1 (66s ago)    2m33s
 kube-system          pod/kube-proxy-b7t22                                1/1     Running   0              2m30s
 kube-system          pod/kube-proxy-nnb2b                                1/1     Running   1 (2m25s ago)  2m29s
 kube-system          pod/kube-proxy-pkwv6                                1/1     Running   0              2m41s
 kube-system          pod/kube-proxy-xc8pb                                1/1     Running   1 (104s ago)   2m39s
 kube-system          pod/kube-scheduler-upgrade-demo-control-plane       1/1     Running   1 (7m39s ago)  7m39s
 kube-system          pod/kube-scheduler-upgrade-demo-control-plane2      1/1     Running   1 (5m31s ago)  5m31s
 kube-system          pod/kube-scheduler-upgrade-demo-control-plane3      1/1     Running   1 (2m25s ago)  2m26s
 local-path-storage   pod/local-path-provisioner-988d74bc-fgshm           1/1     Running   1 (7m9s ago)   39m

 NAMESPACE   NAME                             STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
 default     persistentvolumeclaim/ubuntu-pvc Bound    pvc-ae3b0333-e828-4fdf-a527-e4a9bfe25097   1Gi        RWO            standard       <unset>                 37m
 Trace: /home/ubuntu/kubernetes-upgrade/plans/agent_trace_1_30_to_1_31.json
 UPGRADED SUCCESSFULLY
 Cluster is on target 1.31.
 Existing pods/PVCs were preserved (in-place upgrade).
 Report: reports/cluster-upgrade-feasibility-and-risks_1_30_to_1_31.md
 Upgrade result: /home/ubuntu/kubernetes-upgrade/plans/upgrade_1_30_to_1_31.json
 (.venv) ubuntu@ip-10-0-90-244:~/kubernetes-upgrade/k8s_upgrade_analyzer$
```

Upgrade completed

Verify: Success is a health check, not an exit code

A Successful Command Does Not Mean a Healthy Cluster

The verifier independently re-checks the cluster after execution.

5 Questions Decide Whether We Succeeded

01 Are all nodes **Ready**?
02 Are all nodes on the **target version**?
03 Are all **API services available**?
04 Are any pods stuck outside **Running / Succeeded**?
05 Are all **admission webhooks** still available?

```
ubuntu@ip-10-0-90-244:~$ kubectl get nodes
NAME                        STATUS   ROLES           AGE   VERSION
upgrade-demo-control-plane    Ready    control-plane   41m   v1.31.14
upgrade-demo-control-plane2   Ready    control-plane   41m   v1.31.14
upgrade-demo-control-plane3   Ready    control-plane   41m   v1.31.14
upgrade-demo-worker           Ready    <none>          40m   v1.31.14
upgrade-demo-worker2          Ready    <none>          40m   v1.31.14
upgrade-demo-worker3          Ready    <none>          40m   v1.31.14
ubuntu@ip-10-0-90-244:~$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
ubuntu-writer-59d5bbcd68-b7vf6   1/1     Running   0          3m29s
ubuntu@ip-10-0-90-244:~$ kubectl get pvc
NAME         STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
ubuntu-pvc   Bound    pvc-ae3b0333-e828-4fdf-a527-e4a9bfe25097   1Gi        RWO            standard       <unset>                 39m
ubuntu@ip-10-0-90-244:~$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                 STORAGECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
pvc-ae3b0333-e828-4fdf-a527-e4a9bfe25097   1Gi        RWO            Delete           Bound    default/ubuntu-pvc    standard       <unset>                          39m
ubuntu@ip-10-0-90-244:~$
```

```
========================================
POST-UPGRADE VERIFICATION
========================================
==> Node version matrix:
NAME                          ROLES    VERSION    STATUS
upgrade-demo-control-plane              v1.31.14   Ready
upgrade-demo-control-plane2             v1.31.14   Ready
upgrade-demo-control-plane3             v1.31.14   Ready
upgrade-demo-worker           <none>    v1.31.14   Ready
upgrade-demo-worker2          <none>    v1.31.14   Ready
upgrade-demo-worker3          <none>    v1.31.14   Ready
==> Workloads (pods/pvc sample):
NAMESPACE           NAME                                                 READY   STATUS    RESTARTS        AGE
default             ubuntu-writer-59d5bbcd68-b7vf6                       1/1     Running   0               64s
kube-system         coredns-7c65d6cfc9-khcpt                             1/1     Running   0               64s
kube-system         coredns-7c65d6cfc9-z4h2z                             1/1     Running   0               102s
kube-system         etcd-upgrade-demo-control-plane                      1/1     Running   1 (7m39s ago)   7m39s
kube-system         etcd-upgrade-demo-control-plane2                     1/1     Running   1 (5m32s ago)   5m31s
kube-system         etcd-upgrade-demo-control-plane3                     1/1     Running   1 (2m26s ago)   2m26s
kube-system         kindnet-4mnqw                                        1/1     Running   1 (7m39s ago)   39m
kube-system         kindnet-8glvm                                        1/1     Running   1 (104s ago)    38m
kube-system         kindnet-l6swv                                        1/1     Running   1 (66s ago)     38m
kube-system         kindnet-skgss                                        1/1     Running   1 (3s ago)      38m
kube-system         kindnet-sss7c                                        1/1     Running   1 (2m25s ago)   38m
kube-system         kindnet-vxgp7                                        1/1     Running   1 (5m31s ago)   38m
kube-system         kube-apiserver-upgrade-demo-control-plane            1/1     Running   1 (7m39s ago)   7m39s
kube-system         kube-apiserver-upgrade-demo-control-plane2           1/1     Running   1 (5m32s ago)   5m31s
kube-system         kube-apiserver-upgrade-demo-control-plane3           1/1     Running   2 (2m25s ago)   2m26s
kube-system         kube-controller-manager-upgrade-demo-control-plane   1/1     Running   1 (7m39s ago)   7m39s
kube-system         kube-controller-manager-upgrade-demo-control-plane2  1/1     Running   1 (5m31s ago)   5m31s
kube-system         kube-controller-manager-upgrade-demo-control-plane3  1/1     Running   1 (2m25s ago)   2m26s
kube-system         kube-proxy-5ljz9                                     1/1     Running   1 (4s ago)      2m36s
kube-system         kube-proxy-8hszx                                     1/1     Running   1 (66s ago)     2m33s
kube-system         kube-proxy-b7t22                                     1/1     Running   0               2m30s
kube-system         kube-proxy-nnb2b                                     1/1     Running   1 (2m25s ago)   2m29s
kube-system         kube-proxy-pkwv6                                     1/1     Running   0               2m41s
kube-system         kube-proxy-xc8pb                                     1/1     Running   1 (104s ago)    2m39s
kube-system         kube-scheduler-upgrade-demo-control-plane            1/1     Running   1 (7m39s ago)   7m39s
kube-system         kube-scheduler-upgrade-demo-control-plane2           1/1     Running   1 (5m31s ago)   5m31s
kube-system         kube-scheduler-upgrade-demo-control-plane3           1/1     Running   1 (2m25s ago)   2m26s
local-path-storage  local-path-provisioner-988d74bc-fgshm                1/1     Running   1 (7m9s ago)    39m
NAMESPACE   NAME         STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
default     ubuntu-pvc   Bound    pvc-ae3b0333-e828-4fdf-a527-e4a9bfe25097   1Gi        RWO            standard       <unset>                 37m

========================================
UPGRADED SUCCESSFULLY → v1.31.14
========================================
All control-plane and worker nodes are on v1.31.x.
Cluster 'upgrade-demo' workloads/PVCs were preserved (in-place kubeadm).
```

The failure cases that nearly broke this

Case 1- The Control Plane Upgraded. The Workers Didn't.

The Control Plane Upgraded. The Workers Didn't.

THE FAILURE

Our HA cluster:

3 Control Planes + 3 Workers

The first implementation checked only the **first node** after each upgrade hop.

Result:

CPs → 1.31
Workers → still 1.30

The hop was incorrectly marked successful.

THE FIX

The executor now discovers nodes by Kubernetes role and upgrades them in order:

1st Control Plane → Remaining Control Planes → Every Worker

For each worker:

Cordon → Drain → Upgrade → Kubelet → Uncordon → Verify

Next version hop is blocked until every node matches.

Lesson:

A cluster upgrade succeeds only when the whole cluster has crossed the version boundary.

Case 2- Kubeadm Minor-Version Skew Broke the Upgrade Hop

THE FAILURE

During the 1.30 → 1.31 hop:

- apt installed **kubeadm 1.32** because the package was not pinned
- kubeadm 1.32 refused to upgrade a **1.30** control plane
- The upgrade stopped before the intended hop could begin

THE FIX

Pin the toolchain to the target minor version.

1.30 → kubeadm 1.31.x → 1.31 → kubeadm 1.32.x → 1.32

- Clear stale / conflicting APT repositories
- Pin kubeadm and kubelet for every hop
- Enforce one minor version at a time automatically

LESSON:

The cluster version is not the only version you have to control.
Your upgrade tooling must move one minor at a time too.

Three things to take home

So, Can We Really Let an Agent Upgrade Kubernetes?

Yes — but not by trusting the model more.
By designing the system better.

01 — Autonomy Is a System Property

A capable model alone does not create a reliable agent.

Tools, policies, gates, verification and recovery are what make autonomy safe.

02 — Real-World Failures Are the Best Test

Our biggest improvements came from things going wrong:

version skew · accidental recreation · incomplete node upgrades

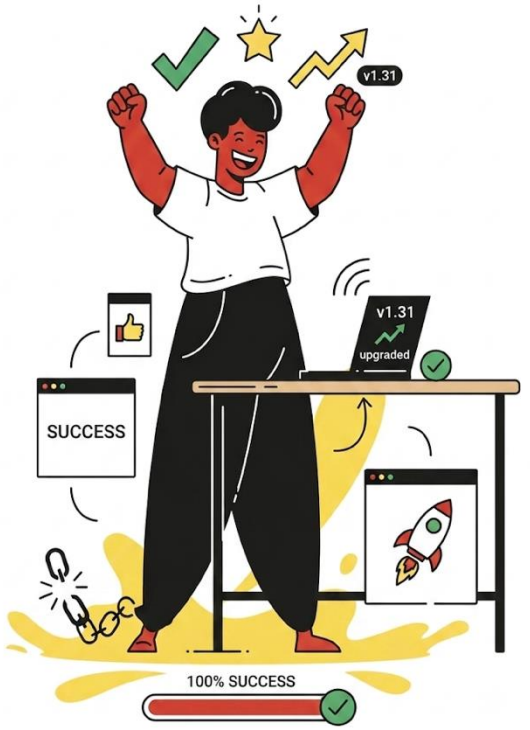
Each failure became a **new guardrail**.

03 — The Blueprint Goes Beyond Upgrades

What we built is a reusable pattern for infrastructure automation:

Observe → Reason → Act → Verify

Kubernetes upgrades were our proving ground.
The pattern is the real takeaway.



Agentic AI Foundation
AGNTCon + MCPCon Japan

Explore the Project on GitHub

Thank You

Where the **agentic stack** is **being built**.