# Hibernate and Evolving:

# Taming OpenClaw with OpenKruise Agents

Secure Execution, Stateful Upgrades, and Cost-Efficient Hibernation on Kubernetes

**Zhen Zhang**

OpenKruise Maintainer · Alibaba Cloud Container Service

SHANGHAI · 2026

WHY THIS TALK

# Running OpenClaw Is Easy. Operating It Is Not.

A single sandbox is a container. A fleet of long-lived agent sandboxes is an operations problem.

01

## Secure Execution

- LLM-generated code is untrusted
- Network destinations are unpredictable
- Credentials must never reach sandbox code
- Gateways alone are not a boundary

02

## Stateful Evolution

- OpenClaw versions keep changing
- Claimed sandboxes hold user state
- Running and Paused must both evolve
- Pod recreation is never free

03

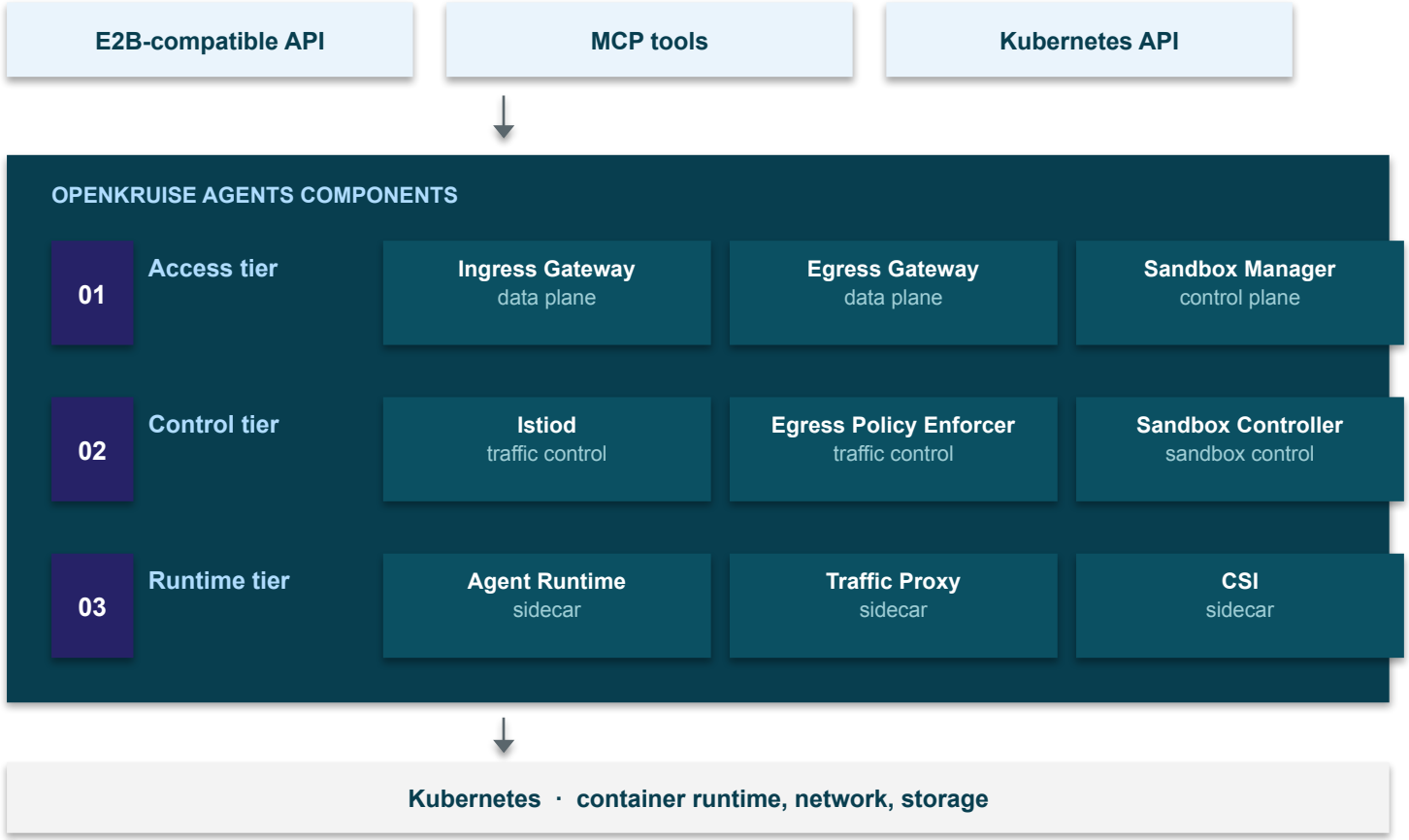
## Cost Optimization

- Agent sessions are long-lived
- Real compute usage is bursty
- Idle sandboxes still consume capacity
- Warm capacity must be right-sized

**Production agent infrastructure must enforce boundaries, evolve stateful runtimes, and reclaim idle compute.**

# OpenKruise Agents as the Operations Control Plane

## NORTHBOUND ACCESS




**A sandbox is managed as a stateful, elastic, policy-controlled runtime — not just another Pod.**

## THREE CONTROL LOOPS

### Secure

L4 isolation, L7 policy, token injection

### Evolve

Pool rollout, claim-time and in-fleet upgrades

### Scale & Hibernate

Pool autoscaling, idle pause, resume

## CORE RESOURCES

Sandbox · SandboxSet · SandboxClaim · PoolAutoscaler · SandboxUpdateOps · Checkpoint

# Layer 4 Network Isolation: Controlling Ingress and Egress Traffic

**THREAT MODEL**

Generated code may use raw sockets, custom DNS or direct HTTP · a sandbox may probe internal services · cloud metadata must stay unreachable · a recommended gateway can always be skipped

**PLATFORM BASELINE**
**GlobalTrafficPolicy**

- Mandatory cluster-wide rules
- Block metadata, private ranges and high-risk ports
- Cannot be overridden by lower layers

**TEAM-LEVEL CONTROL**
**TrafficPolicy**

- Namespace / team egress policy
- FQDN, CIDR, Service and workload destinations
- Port, protocol and DNS TTL-aware controls

**SANDBOX NETWORK CONTROL**
**E2B Network API**

- Internet access control
- Allow and deny lists for outbound traffic
- Per-sandbox configuration through the E2B Network API

**ENFORCEMENT PRINCIPLES**

- Transparent ingress and egress interception
- Non-cooperative — no SDK or proxy env required
- Effective before the first packet crosses the boundary
- Deny always outranks a lower-level allowlist

```
curl 100.100.100.200
```

**Cloud metadata service → denied at L4**

Traffic never leaves the sandbox path.

**Network isolation must hold on both ingress and egress — even when the agent tries to bypass it.**

# Layer 7 Security: Policy Enforcement and Token Injection

## REQUEST-LEVEL POLICY

- **Destination domain**
- **HTTP method**
- **Request path**
- **Request headers**

**Actions: block · bypass · audit · mcpToolPolicy**

## HTTPS TRAFFIC CONTROL

- Controlled TLS termination on the egress path
- L7 rules apply to HTTPS requests, not only HTTP
- Trusted CA configuration injected into the sandbox
- Transparent to OpenClaw and its SDKs

**SCOPE L7 rules and token injection can target a workload or a single sandbox, including through the E2B-compatible sandbox API.**

## TRANSPARENT TOKEN INJECTION

**OpenClaw sends a placeholder token**
Authorization: Bearer FAKE_TOKEN

↓

**Egress gateway matches the request**
destination + header rules

↓

**Gateway injects the real credential**
resolved on the trusted side

↓

**Authorized request reaches the API**
sandbox never sees the secret

### SUPPORTED CREDENTIAL TYPES

- LLM API keys
- Generic Bearer Tokens
- Alibaba Cloud STS credentials

**The sandbox can use an external service without ever storing or seeing the real credential.**

# An Unskippable Secure Execution Path

**OpenClaw / generated code**
request initiated inside the sandbox

↓

**traffic-proxy sidecar**
transparent interception of egress traffic

↓

**L4 policy evaluation**
GlobalTrafficPolicy + TrafficPolicy

↓

**Central egress gateway**
trusted side, centralized policy and audit

↓

**L7 policy + token injection**
SecurityProfile rules via ext_proc

↓

**External API**
authorized request forwarded

The enforced path scales to production-sized fleets without becoming the bottleneck.

## VALIDATED AT LARGE SCALE

| | |
|---|---|
| **100K+**<br>sandboxes in one cluster | **200K**<br>TrafficPolicy rule capacity |
| **25K**<br>rules in one TrafficPolicy | **500K**<br>rules across the cluster |
| **35K RPS**<br>per replica · P99 <5ms | **P99 4.9 ms**<br>ext_proc call latency |
| **< 2 s**<br>policy propagation · 20K rules | **10m CPU / 40 MB**<br>sidecar overhead · 10K rules |

**Graceful gateway rolling restart · zero traffic loss**

# Why Stateful Sandbox Upgrades Are Different

01

## User state

- Workspace files
- Root filesystem changes
- Runtime-generated artifacts
- Live session context

02

## Compatibility

- OpenClaw versions
- Browser and runtime builds
- Toolchain dependencies
- Sidecar expectations

03

## Recreation cost

- Pod restart time
- Temporary interruption
- IP changes
- Readiness and route recovery

04

## Mixed states

- Running sandboxes
- Paused sandboxes
- Transitioning instances
- Claimed vs. pool capacity

AVAILABILITY BOUNDARY — SAY IT PLAINLY

**Fleet capacity can stay available throughout a rollout, while an individual sandbox may still see a short, bounded interruption.**

**Every upgrade must answer what is replaced, what is preserved, and how disruption is bounded.**

# Two Upgrade Paths: Warm Pools and Claimed Sandboxes

WARM POOL — UNCLAIMED CAPACITY

## SandboxSet rollout

- Update SandboxSet.spec.template
- Roll unclaimed instances in batches
- Scale-out instances use the new version
- Scale-in removes old versions first

| maxUnavailable | Completion |
|---|---|
| default 20% | updatedAvailable == replicas |

Track: replicas · availableReplicas · updatedReplicas

CLAIMED SANDBOXES — LIVE USER STATE

## SandboxUpdateOps

- Create a one-time SandboxUpdateOps
- Select targets with a label selector
- Apply a strategic merge patch
- Controller drives per-sandbox progress

| maxUnavailable | Concurrency |
|---|---|
| default 1 | one Updating op per namespace |

Track: Total · Updating · Updated · Failed

**Warm pools replace disposable ready capacity; claimed sandboxes evolve stateful user workloads.**

# Dynamic Image and Resource Updates at Claim Time

## ALLOCATION FLOW

**Standardized warm pool**
stable image, baseline CPU

↓

**SandboxClaim arrives**
declares the target runtime

↓

**In-place update on claim**
image override + CPU resize

↓

**Claimed sandbox is ready**
tenant-specific runtime, no rebuild

## WHY IT MATTERS

- One warm pool serves multiple runtime profiles
- Pre-warm small, expand resources only when claimed
- Canary versions need no separate full pool

**SANDBOXCLAIM**

```yaml
apiVersion: agents.kruise.io/v1alpha1
kind: SandboxClaim
metadata:
  name: demo-sandbox-claim
spec:
  templateName: openclaw-pool
  inplaceUpdate:
    image: openclaw:next
    resources:
      requests:
        cpu: "1"
      limits:
        cpu: "2"
```

## LIMITS TO STATE ON STAGE

- Image replacement adds claim latency — no sub-second promise
- CPU resize relies on in-place pod vertical scaling
- Memory resize is not claimed as supported
- The pod QoS class must not change
- With no warm instance, create directly at target spec

**Pre-warm a common baseline; specialize the runtime at the moment it is claimed.**

# Upgrading Running and Paused Sandboxes

## TARGET SELECTION

By default only Running sandboxes are upgraded.
**Paused instances must be included explicitly.**

**INCLUDE HIBERNATED SANDBOXES**

```
kind: SandboxUpdateOps
spec:
  stateFilter:
    states:
    - Running
    - Paused
```

## BEHAVIOUR BOUNDARIES

- Only a fully paused sandbox is eligible
- Pausing / Resuming instances are skipped, then retried
- Explicit user wake-up takes precedence
- A sandbox already Upgrading is not paused mid-flight

## PAUSED SANDBOX UPGRADE FLOW

**Paused**
fully hibernated, selected by the operation

↓

**Wake up**
controller resumes the sandbox first

↓

**Upgrade**
patch applied, pod recreated

↓

## FINAL STATE FOLLOWS THE DESIRED PAUSE STATE

`spec.paused still true`
**Returns to Paused**

`Explicitly set to false`
**Stays Running**

**A hibernated fleet is still part of the fleet — plan its upgrade window and its return to sleep.**

**Upgrade the whole fleet without forgetting the sandboxes that are currently asleep.**

# Recreate + Lifecycle: Explicit State Handoff

### PreUpgrade

- Quiesce OpenClaw sessions
- Flush the workspace
- Archive selected paths

### UpgradePod

- Recreate the pod
- Apply the new image
- Reattach persistent volumes

### PostUpgrade

- Restore archived data
- Validate agent state
- Return traffic when ready

**BACK UP AND RESTORE THE OPENCLAW WORKSPACE**

```
kind: SandboxUpdateOps
spec:
  lifecycle:
    preUpgrade:
      exec:
        command: [/bin/bash, -c,
          "tar -czf /backup/openclaw-ws.tgz
           -C /root .openclaw"]
        timeoutSeconds: 600
    postUpgrade:
      exec:
        command: [/bin/bash, -c,
          "rm -rf /root/.openclaw &&
           tar -xzf /backup/openclaw-ws.tgz
           -C /root"]
        timeoutSeconds: 600
```

**WHAT SURVIVES A RECREATE**

| State | Default | With lifecycle |
|---|---|---|
| **Rootfs writable layer** | No | Archived paths |
| **Workspace** | No | Yes |
| **Memory** | No | No |
| **Pod IP** | No | No |
| **External PVC / NAS** | Yes | Yes |

**FAILURE HANDLING**

- PreUpgradeFailed — pod not yet recreated, data intact
- UpgradePodFailed / PostUpgradeFailed — fix, then recreate the op
- Drop a completed preUpgrade hook on retry to protect the backup

**Recreate does not preserve state by itself; lifecycle hooks make the handoff explicit.**

# CheckpointRestore: Preserve Rootfs, Not Everything

## UPGRADE FLOW

**Checkpoint the rootfs**
capture the container writable layer

↓

**Recreate the pod**
apply the non-image change

↓

**Restore the writable layer**
rootfs returns with the sandbox

↓

**Validate the sandbox**
readiness and agent state checks

**UPDATE STRATEGY**

```
kind: SandboxUpdateOps
spec:
  updateStrategy:
    type: CheckpointRestore
  patch:
    metadata: { annotations: { ... } }
```

**CheckpointRestore preserves the writable layer — not the entire live process.**

## CHOOSE BY WHAT MUST SURVIVE

| Dimension | Recreate | Checkpoint |
|---|---|---|
| **New business image** | Supported | Restricted |
| **Full rootfs** | No | Preserved |
| **Selected paths** | External backup | Included |
| **Memory** | No | No guarantee |
| **IP / connections** | No | No |

## SUITABLE CHANGES

- Platform sidecar updates
- Environment variables and arguments
- CPU resources
- Volume mounts and other non-image fields

**CRITICAL RESTRICTION**

**Do not replace the business container image in this mode and still expect its rootfs to survive.**

On CheckpointFailed: confirm the sandbox reached a stable state, correct the operation, then delete and recreate the one-time task.

# Why Idle Agents Still Consume Expensive Compute

A TYPICAL OPENCLAW SESSION


**Blue = compute actually used**

Grey = allocated but idle

## State that must survive

- Sessions and conversation context
- Workspace files and artifacts
- Installed runtime dependencies
- Scheduled jobs inside the agent

## Compute that need not stay

- Execution windows are short
- Waiting dominates the session
- User activity is unpredictable
- Every Running Pod holds capacity

## Optimization hierarchy

- Right-size the warm pool
- Pause idle claimed sandboxes
- Resume before scheduled work
- Resume on incoming traffic

**Preserve agent state, not idle compute.**

# Elastic Warm Pools: Scaling by Capacity and Cron Policies

TWO ELASTICITY STRATEGIES

AVAILABILITY-DRIVEN

## Capacity policy

- Keep a target share of Available sandboxes
- targetAvailable takes a count or a percentage
- tolerance absorbs minor fluctuation
- Separate scale-up / scale-down stabilization

SCHEDULE-DRIVEN

## Cron policies

- Set an explicit target replica count
- Cron expression with an explicit time zone
- Prepares capacity before demand arrives
- Fits working hours and planned events

**PRECEDENCE** When both exist, cron policies win · suspend disables autoscaling

**Cron policies set the plan; the capacity policy keeps the pool ready.**

POOLAUTOSCALER

```yaml
apiVersion: agents.kruise.io/v1alpha1
kind: PoolAutoscaler
metadata:
  name: openclaw-pool-autoscaler
spec:
  scaleTargetRef:
    kind: SandboxSet
    name: openclaw-warm-pool
  minReplicas: 2
  maxReplicas: 50
  # availability-driven elasticity
  capacityPolicy:
    targetAvailable: "30%"
    tolerance: "10%"
    scaleUp:
      stabilizationWindowSeconds: 60
    scaleDown:
      stabilizationWindowSeconds: 300
  # schedule-driven elasticity
  cronPolicies:
  - name: morning-scale-up
    schedule: "0 9 * * 1-5"
    timeZone: "Asia/Shanghai"
    targetReplicas: 20
  - name: evening-scale-down
    schedule: "0 18 * * 1-5"
    timeZone: "Asia/Shanghai"
    targetReplicas: 5
```

# Two Paths to Automatic Hibernation

## A Timer-Based Hibernation

`spec.pauseTime`

- Pause at a known absolute deadline
- Independent of agent activity
- Fits TTL sessions and fixed windows
- Simple, deterministic, hard deadline

May pause an active agent if the deadline is set too aggressively.

`Sets spec.paused = true`

## B Probe-Based Idle Hibernation

`autoPausePolicy.pause`

- Probe reports active or inactive
- Requires sustained idleness
- Pauses only after thresholdDuration
- Fits long-lived, intermittent sessions

Needs a reliable in-sandbox idle check; probe failures do not trigger a pause.

`Sets spec.paused = true`

SHARED EXECUTION PATH

**Both paths converge on spec.paused = true — the existing Sandbox pause mechanism releases the compute**

**Timers impose a deadline; probes reveal when the agent is actually idle.**

# Detecting Agent Idleness with Probes

PROBE EXECUTION PATH

**Sandbox defines the probe**
spec.probes[]

↓

**Controller writes pod probe config**
PodProbeMarker protocol

↓

**Kruise Daemon executes the command**
asynchronous, outside Reconcile

↓

**Result enters Pod Conditions**
status, reason, message

↓

**Mirrored to Sandbox Conditions**
agents.kruise.io/<probe-name>

**Planned: move probe execution to Agent Runtime.**

**SEMANTIC DISTINCTION**

Condition True means the probe executed successfully.
**messageRegex decides whether the agent is idle.**

**The platform runs the control loop; the agent defines what idle means.**

**OPENCLAW IDLE CHECK**

```
# is any session active in the last 10 minutes?
openclaw sessions --active 10 --json
{ "count": 0, "activeMinutes": 10, "sessions": [] }
```

`count > 0 → active`  `count == 0 → inactive`

**PROBE AND AUTO-PAUSE POLICY**

```
kind: Sandbox
spec:
  probes:
  - name: Active
    containerName: openclaw
    exec:
      command: [sh, -c, "<idle check above>"]
    periodSeconds: 30
    timeoutSeconds: 10
    failureThreshold: 3
  autoPausePolicy:
    pause:
      whenProbedIdleState:
        probe: Active
        messageRegex: "^inactive$"
        thresholdDuration: 15m
```

# Two Paths to Wake Up a Sandbox

## A Traffic-Triggered Resume

driven by an incoming request

1. Request reaches the sandbox gateway
2. Gateway finds the sandbox Paused
3. Resume is triggered on demand
4. Wait for readiness, refresh routes
5. Forward, retry or reconnect the call

**TYPICAL TRIGGERS**

- A user opens OpenClaw again
- An API or SDK call arrives
- An interactive session reconnects

**Planned: wake a paused sandbox from an inbound IM message.**

## B Schedule-Aware Resume

driven by the cron probe output

1. Cron probe reports the next task time
2. Controller parses the timestamp
3. nextResumeTime = taskTime − leadTime
4. Schedule persisted before pausing
5. Resume fires when the timer expires

**RESUME POLICY**

```
kind: Sandbox
spec:
  autoPausePolicy:
    resume:
      whenProbedScheduleTime:
        probe: Cron
        timeFormat: unix
        leadTime: 5m
```

**BOUNDARIES** In-sandbox probes stop while Paused, so the resume time must already be persisted · traffic-triggered resume is a gateway integration path · long-lived connections need reconnect logic · a stale pauseTime can immediately pause a resumed sandbox

**Unpredictable work resumes on demand; predictable work resumes by schedule.**

# Probing OpenClaw Scheduled Tasks

SCHEDULE DISCOVERY PATH

**Cron probe runs inside the sandbox**
reads the OpenClaw job store

↓

**Probe prints the next task timestamp**
message becomes a Sandbox Condition

↓

**Controller parses the timestamp**
timeFormat: unix or datetime

↓

**nextResumeTime is persisted**
status.schedules, written before pausing

↓

**Sandbox pauses, timer survives**
in-sandbox probes stop while Paused

`nextResumeTime = taskTime − leadTime`

**WHY IT MUST BE PERSISTED**

A paused sandbox cannot run its own probe.
**Without a stored schedule, resume needs an external trigger.**

**OPENCLAW CRON INSPECTION**

```
# list the configured jobs
openclaw cron list --json
# report the next scheduled wake-up
openclaw cron status --json
{ "enabled": true, "jobs": 3,
  "nextWakeAtMs": 1776970800000 }
```

`nextWakeAtMs → probe message → nextResumeTime`

**PROBE AND AUTO-PAUSE POLICY**

```
kind: Sandbox
spec:
  probes:
  - name: Cron
    containerName: openclaw
    exec:
      command: [sh, -c, "openclaw cron status
        --json | jq -r .nextWakeAtMs"]
  autoPausePolicy:
    resume:
      whenProbedScheduleTime:
        probe: Cron
        timeFormat: unix
```

**The agent reports when it must wake; the platform stores that answer before it sleeps.**

CLOSING

# Three Control Loops for Production-Grade OpenClaw

## Secure

- Intercept every outbound request
- Enforce L4 baselines and tenant policy
- Authorize at L7, inject tokens at the edge
- Keep real credentials out of the sandbox

## Evolve

- Separate pool and claimed upgrades
- Specialize image and CPU at claim time
- Upgrade Running and Paused instances
- Match the strategy to the state at risk

## Scale & Hibernate

- Right-size pools by capacity and cron
- Detect sustained agent idleness
- Release idle compute on pause
- Resume by schedule or by traffic

**OpenKruise Agents turns OpenClaw from a collection of long-lived Pods into an operable agent fleet.**

GET INVOLVED

# Join the OpenKruise Agents Community

Kubernetes-native infrastructure for scalable, secure and stateful AI agent sandboxes.

REPOSITORY

`github.com/openkruise/agents`

DOCUMENTATION

`openkruise.io/kruiseagents/introduction`

UPCOMING RELEASE

## v0.6.0 · targeting September 15

Auto-pause, traffic management and upgrade APIs land with this release — follow the repo for the tag.

SCAN FOR THE PROJECT

`github.com/openkruise/agents`

Issues, proposals and design reviews welcome

**Explore the project, try the APIs, and help shape the future of agent sandbox operations.**