10 NOVEMBER 5, 2026 3 PP

Safe rollouts and a dial-out WireGuard tunnel for on-site energy controllers

Felipe Arce · Tim Child · Collin Rijock

Every Exowatt site runs its energy management system on a small host at the plant. This is how we built the plane that provisions and upgrades those hosts without git, downgrades or avoidable downtime, and the WireGuard tunnel that brings each site's console back to the cloud.

INFRASTRUCTURE DEPLOYMENTS WIREGUARD KUBERNETES EMS

On July 23 a one-line API call took down one of our sites. An agent session asked our deployments server to roll out a site without naming a version. The server shipped the version pinned in its checkout, which was older than what the site ran. The old image’s migration step looked for a database revision it had never seen and failed, and the site’s only control process stayed down until we rolled forward within the hour.

Felipe Arce had built that server two weeks earlier. The fix set the rule the rest of this system follows: the deploy plane, which every caller goes through, decides what is safe to ship.

Since mid-July the plane has turned a bare Debian host into a running site in about 12 minutes from one API call. It ran its first rollout on July 16 and had logged about 490 rollout runs by October 15, and in October it upgraded 17 sites in two and a half hours with no failures or retries. UI deploys no longer drop a request, and one guard fix avoided resizing 15 hosts.

The constraint

Each site runs its energy management system on a small host at the plant: Debian, single-node k3s, PostgreSQL with TimescaleDB, and about a dozen Helm bundles around the controller that speaks Modbus and OPC UA to batteries and inverters. The

site stack runs on-site only, never in the cloud, so a plant keeps control of its equipment when its uplink drops. The deploy system has to reach these hosts behind NAT on plant networks and must never change a running plant by accident.

No git on the site

Our first design, on July 7, was ordinary GitOps: each site's Fleet agent polled a GitHub branch. That afternoon we replaced it with a promotion branch, then replaced that too. Both let a merge or a bad push reach a plant, and both put a GitHub credential on every site.

The model we kept is apply-once. Ansible connects over SSH, installs k3s, PostgreSQL, secrets and a standalone Fleet controller, then runs `fleet apply` once over the bundle directories. A site never talks to GitHub or updates itself. To ship again, someone runs `emsctl rollout` or calls `POST /rollout` on deployments-server, a FastAPI service that runs the same playbooks through ansible-runner. The fleet console and agents use the API.

What a rollout checks

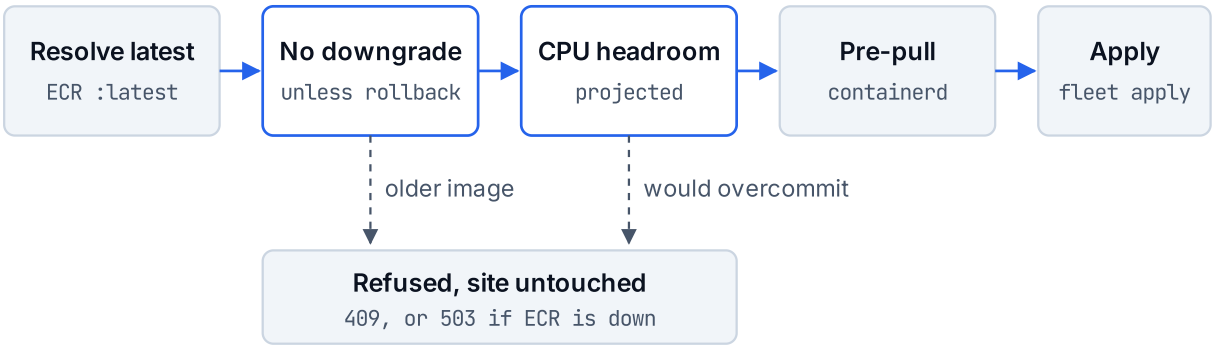


Figure 1. A rollout resolves the newest image, passes two guards that can refuse it before anything on the site changes, pulls images ahead of time, then applies.

After the July outage we made the plane ship the newest image unless the caller marks the request as a rollback. `/rollout` resolves ECR's `latest` tag at request time. `/deploy` returns 409 for anything older unless `rollback: true` is set, and

fails closed with 503 if it cannot reach ECR. A rollback skips ECR, so it still works during an ECR outage.

Every workload used to restart with `Recreate` and pull a multi-hundred-megabyte image inside its outage window. The rollout now pulls every pinned image into containerd before `fleet apply`, so a restart costs only boot time, about 78 seconds for the controller. We moved the UI off a pod hostPort onto k3s's ServiceLB, which holds port 80 across pod generations, and it now rolls with `maxUnavailable: 0`. During a live roll it answered 300 of 300 probes sent 200 ms apart.

The controller stays a `Recreate` singleton, and we kept it that way on purpose. AWS IoT lets a second connection with the same MQTT client ID take over the session, and two processes sending Modbus commands to one battery is a safety problem.

In October the CPU headroom guard refused 15 hosts: 3,850 of 4,000 millicores requested, 500 required free. The upgrade shrank their requests, but the guard only read the live cluster. Rather than move 15 hosts to larger instances, we made the guard render the bundles with Fleet's own CLI and judge the budget after the apply. That came to 3,290 millicores with 710 free, and all 15 passed.

Teardown

Teardown broke more often than provisioning. On July 15 a deprovision on a field host left Rancher's embedded Cluster API controller running while the play deleted the namespaces holding the objects it watched. The controller reacted by deleting the cluster's only Node. With no nodes, metrics-server's API went stale, which blocks namespace deletion cluster-wide, and the next provision failed at its first write.

Since then the play stops that controller and waits for its pods to exit before it deletes anything, because nothing that reacts to deletions may stay running during teardown. Provisioning opens with a `cluster_preflight` role that re-registers a missing Node, removes dead webhooks, repairs API registrations and unsticks terminating namespaces. Each step is a no-op on a healthy cluster, so a provision converges from whatever the last run left.

The console tunnel

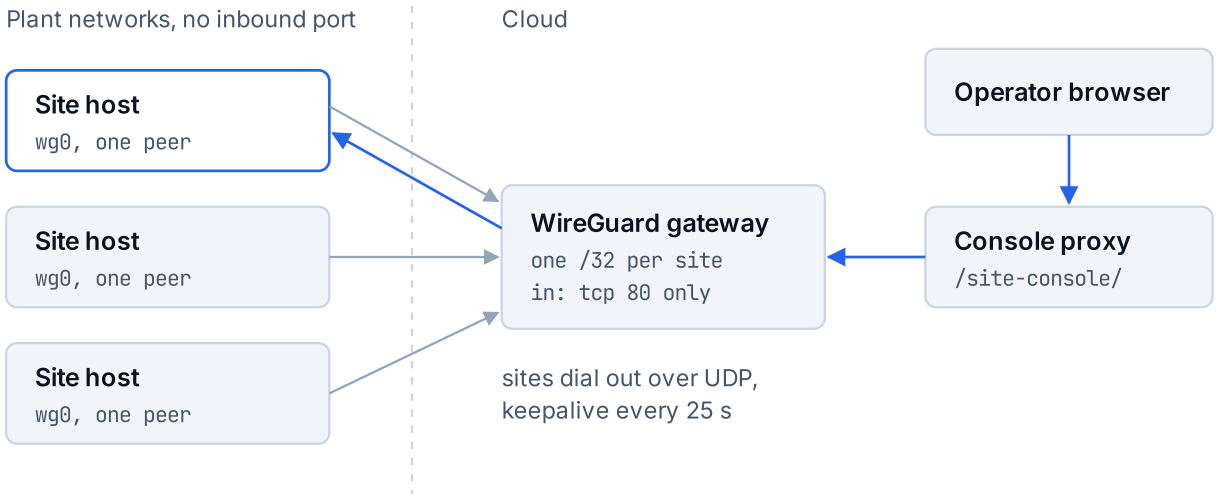


Figure 2. Every site dials out to one gateway (grey). The console request (blue) rides one site's tunnel back in; the gateway may open TCP 80 to that site and nothing else, and sites cannot reach each other.

Operators need each site's console from the cloud, and plant networks should accept no inbound connections, so each site host dials out. It runs `wg-quick@wg0` outside Kubernetes with one peer, the cloud gateway, and a 25-second keepalive to hold NAT open. The gateway has one peer per site, with `AllowedIPs` limited to that site's /32, and an nftables table on `wg0` that lets the cloud open TCP 80 to a site and nothing more.

Each host generates its own key and stores it in AWS Secrets Manager with check-and-set writes, and a re-run never rotates it. The gateway host has no SSH, so a one-minute systemd timer lists registered sites and re-renders its peers and the console proxy map. Provisioning assigns the next free overlay address, so a new site's console is up within a minute with no manual step.

`wg-quick` resolves a hostname endpoint once, at start. One field host came up while its LAN DNS still pointed at a retired lab gateway, and it kept sending handshakes there after we fixed the record. Its config file had not changed, so nothing restarted it. The tunnel role now restarts the interface whenever the latest handshake is more than three minutes old.

Results and what is next

Shipping to a plant is now an API call that refuses unsafe requests, with no git credential on any site and no inbound port on any plant network. Rollouts still go one site at a time, picked by an operator; next we want rings with health gates, so a bad build stops at the first site. The plane also keeps job history in memory, so it cannot say what a site ran last week. A durable per-site deploy record would fix that and let the downgrade guard compare against the site itself instead of the newest image in ECR.