
BUILD GUIDE

# Your Own AI Chat, On Your Own GPU

One UpCloud cluster with an NVIDIA L4, one Ankra stack profile, and you end with a chat UI with document RAG and an OpenAI-compatible API — chat and embeddings — on your own generated domain. TLS included, weights cached, nothing shared.

| MOVES | TIME | GPU | MODEL |
|---|---|---|---|
| Two | ~35 minutes | NVIDIA L4 24 GB | Qwen3 8B FP8 |

ONE PROFILE, THE WHOLE PLATFORM

# What the stack contains

The **gpu-chat** stack profile is the whole serving platform in one deployable unit, wired with dependency ordering so it comes up in the right sequence. Instantiating it produces a stack of 4 Helm addons and 29 manifests on your cluster.

REQUEST PATH


| COMPONENT | WHAT IT DOES |
|---|---|
| **NVIDIA GPU Operator** | Driver, container toolkit, device plugin, DCGM metrics. Turns the raw L4 node into schedulable `nvidia.com/gpu` capacity. |
| **llm-d** | The inference layer: an Envoy gateway fronting an EPP router that schedules each request to the vLLM replica already holding that prompt prefix's KV cache. |
| **Model store** | A persistent volume the weights download into once. Restarts and upgrades reuse it instead of pulling again. |
| **Embeddings + RAG** | A CPU text-embeddings-inference server ( `bge-m3` ) behind `/v1/embeddings` , and Open WebUI on a pgvector Postgres: upload a document in the chat and it is chunked, embedded locally, and retrieved into the conversation. |
| **Open WebUI** | The chat front end, pointed at the router's OpenAI-compatible endpoint. |
| **Two ingresses** | `chat.<domain>` for the UI, `api.<domain>` for the API. DNS records from external-dns, certificates from cert-manager. |
| **Prometheus + Grafana** | A slim kube-prometheus-stack scraping the vLLM and GPU monitors, with a token-usage dashboard reading the broker's key store: spend and budget per key, live. |

**Three layers of caching, compounding**

The model store caches weights on disk, vLLM caches computed prefixes in GPU memory, and the router is cache-aware so requests land where the cache already lives. Chat traffic resends the conversation on every turn, which is exactly the shape this rewards.

MEASURED ON LIVE KUBEADM + L4 BUILDS IN FI-HEL2

# What you wait for, and how long

Most of this build is spent waiting on machines rather than on you. Every phase below finishes on its own or retries itself; none of them need babysitting.

| PHASE | WHAT IS HAPPENING | TYPICAL WAIT |
|---|---|---|
| **Test connection** | Ankra reads your account's zones and plans through the token | seconds |
| **Create → Online** | Network, bastion, control plane, the L4 node, Kubernetes and the agent: a thirteen step sequence | 11–15 min |
| **Worker joins** | The GPU node registers, base add-ons settle, the cluster flips to Healthy | +1–3 min |
| **Stack deploys** | Thirty three resources (four add-ons, twenty nine manifests) deploying as thirty nine jobs | 5–8 min |
| **GPU turns schedulable** | Driver, container toolkit, device plugin, cuda validator, then `nvidia.com/gpu: 1` | ~5 min |
| **DNS and certificate** | external-dns writes the record, cert-manager collects the Let's Encrypt certificate | 1–3 min |
| **vLLM first boot** | About 9 GB of weights into the model store, then loaded into VRAM | 5–10 min |
| **Total** | Create Cluster to the first answer in the chat UI | **~35 min** |

> **Three waits that look like failures and are not**
>
> The provisioning card reads "usually takes 5–10 minutes": a GPU build runs longer, which is normal. A worker step can fail once on an SSH race ( `no route to host` ); the platform retries it by itself, about five minutes lost. And an empty model selector only means vLLM is still pulling weights.

## While it builds

Generate the two secret values stage 2 asks for, ask the assistant for your domain ( `⌘/Ctrl + J` ), and keep the UpCloud console open on **Servers**: the bastion, control plane and L4 appear there as Ankra creates them.

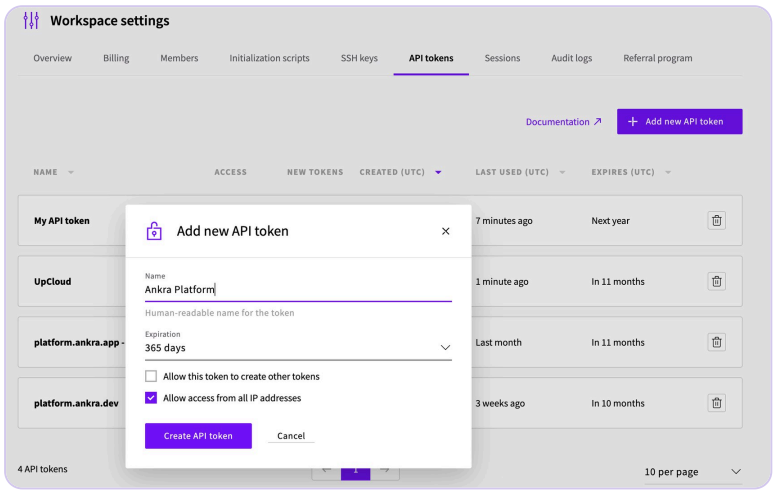
## Tearing it down, in this order

Another fifteen minutes, and skipping a step leaves money running. Delete the **gpu-chat** stack (about five minutes): from profile v10 every volume it created — model stores and databases included — is reclaimed automatically, so watch **Storage → Persistent Volumes** empty by itself within a minute or two (a volume lingering from an older profile version: open its **Manifest → Edit** and flip `persistentVolumeReclaimPolicy` to `Delete` ). Then **Settings → Danger Zone → Terminate**: thirteen destroy jobs, about six minutes, servers and NAT gateway included. *Delete cluster* only clears the record of an already terminated cluster and would leave the L4 billing.

BEFORE STAGE 1: ACCOUNT, TOKEN, CREDENTIAL

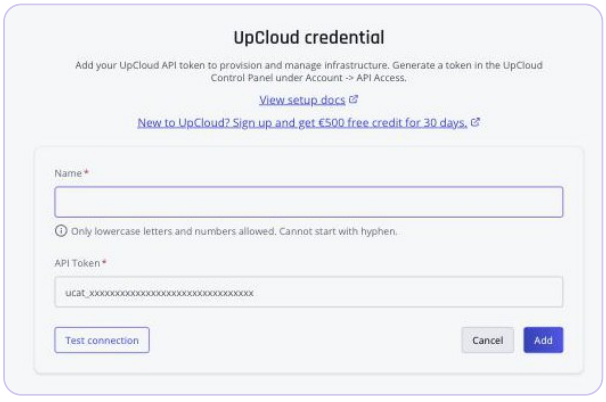
# Give Ankra a key to your account

Sign up at **signup.upcloud.com/?promo=ankraio** (the €500 / 30 day trial), then create the API token Ankra will use: profile menu → account → **API tokens**, or straight to **hub.upcloud.com/account/api-tokens**. Hit **Add new API token**, name it something you will recognise (Ankra Platform works), set an expiration, leave "allow this token to create other tokens" unchecked, and keep "allow access from all IP addresses" ticked so Ankra's workers can reach the API. The token starts with `ucat_` and is shown once: copy it before you close the dialog.




A. The API token, created in the UpCloud control panel and shown once

Then hand it to Ankra: sign up at **platform.ankra.app**, open **Credentials** → **Add credential**, pick UpCloud, name it (lowercase letters and numbers, no leading hyphen), paste the `ucat_` token, and hit **Test connection** before you save. A green result means Ankra can read your account's zones and plans, which is exactly what the create wizard on the next page needs. The credential is stored once at organisation level and reused by every cluster you create afterwards.




B. The UpCloud credential in Ankra, tested before saving

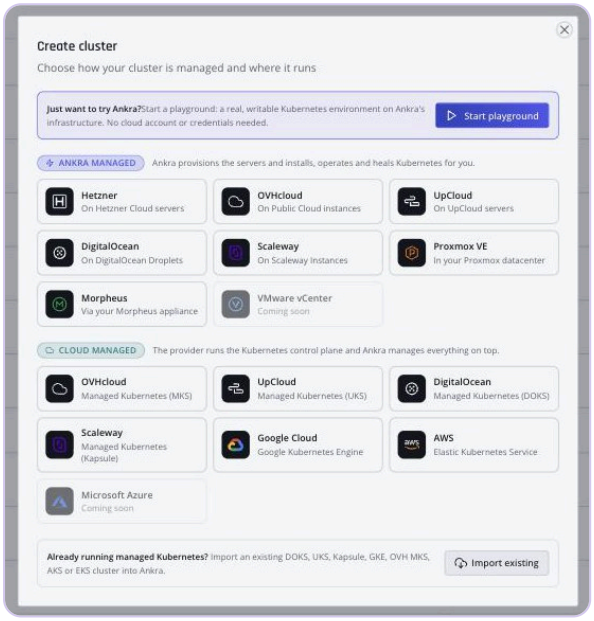
**Card on the account before the event**

UpCloud asks for card details at sign-up as a know-your-customer check: not a charge, but without it the account cannot create servers, and GPU plans in particular are only released to verified accounts. Add the card before the event, not during it.

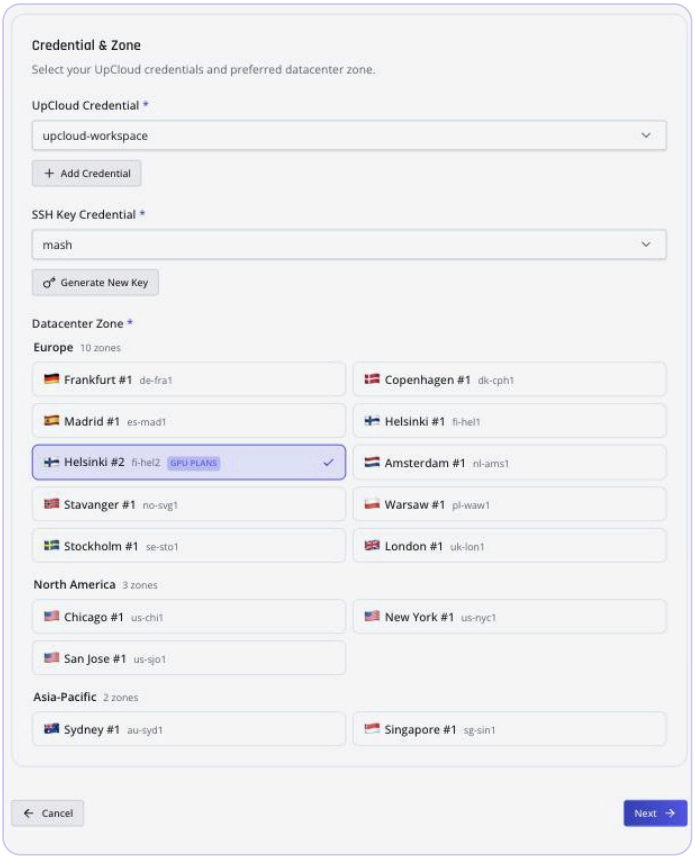
1 CREATE THE GPU CLUSTER

# One cluster, one GPU node

In Ankra go to **Clusters**, hit **Create Cluster**, and pick **UpCloud** under Ankra Managed. Select your UpCloud credential and an SSH key credential (no key yet? **Generate New Key** mints one in the form), then the zone: **Helsinki #2 (fi-hel2)** is the only UpCloud zone with GPU plans, and the picker badges it. The plans and prices are read live from your UpCloud account through your credential: Ankra's Bring Your Own Key principle. The servers it creates, the L4 included, are yours, in your account, with Ankra holding only the configuration and operations.




1. UpCloud under Ankra Managed




2. fi-hel2 carries the GPU plans badge

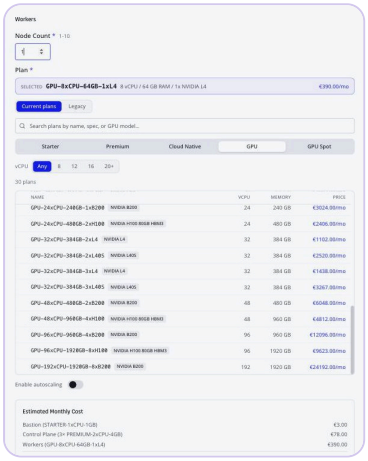
**The three sizes that matter**

Bastion `STARTER-1xCPU-1GB` at €3/mo (SSH jump host only), control plane `1× PREMIUM-2xCPU-4GB` at €26; the wizard offers **3** nodes, drop it to one (keep three if you call it production). Take the **Kubernetes** defaults as offered (kubeadm with Cilium). If `10.0.0.0/16` is already taken in your account, give the network another range. The GPU node comes next.

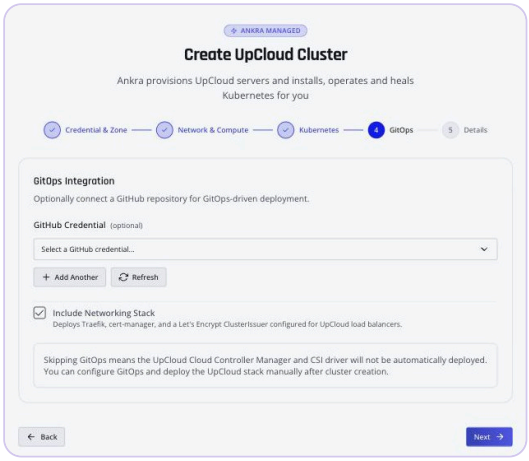
THE GPU NODE AND THE TWO SWITCHES

# Pick the L4, keep the boxes ticked

In the workers section open the **GPU** tab and select `GPU-8xCPU-64GB-1xL4`: 8 vCPU, 64 GB RAM, one NVIDIA L4 with 24 GB VRAM, at **€390/month**; the tab opens on the 12xCPU / €470 plan, so change it, and set the worker count to **1** (it offers 2). The wizard totals the cluster at **€419/month** (bastion €3, control plane €26, the L4 €390), and that estimate covers the servers only: the load balancer, NAT gateway and CSI volumes are billed on top. The **GPU Spot** tab prices the same L4 at €383, about two percent off a node that can be reclaimed under you: take the on-demand plan. On the GitOps step two switches are ticked by default: **Include Networking Stack** (Traefik, cert-manager and the Let's Encrypt issuer — automatic certificates) and **Include Public DNS** (your generated ankra.cc subdomain plus external-dns — automatic DNS records). Keep both. From **Create** to online is **11–15 minutes**; page 3 times every wait in the build.




**3.** The GPU tab, L4 selected




**4.** The switch behind automatic https

### Find your generated domain

Once online, press `⌘/Ctrl + J` and ask "what is my cluster's public domain?"; the answer names the `chat.` and `api.` hostnames the stack will serve. This build answered `4sgrlisp5r.2hqbgfae9a.ankra.cc`. Any hostname under the suffix is yours, and once the stack is up **Overview** → **Endpoints** lists the live ones.

TERMINAL EQUIVALENT

```
ankra cluster upcloud create --name gpu-chat \
  --credential-id <upcloud-credential-id> --ssh-key-credential-id <ssh-key-id> \
  --zone fi-hel2 --distribution kubeadm --network-ip-range 10.84.0.0/16 \
  --worker-count 1 --worker-plan GPU-8xCPU-64GB-1xL4
ankra cluster domain <cluster-id>   # needs CLI v0.11.0+
```

ALTERNATIVE TO STAGE 1: SKIP IF YOUR CLUSTER ALREADY EXISTS

# Create it in UpCloud, import it into Ankra

An alternative to stage 1, not an addition: if Ankra already created your cluster, skip to stage 2. Choose this path to keep the cluster resource itself in UpCloud's console (node groups, scaling, version upgrades) rather than having Ankra provision servers. Ankra manages everything on top exactly as in the rest of this guide, and never deprovisions infrastructure it did not create.

## Let Ankra create the UKS cluster

Ankra can create the managed cluster itself: **Create Cluster**, then under **Cloud Managed** pick **UpCloud — Managed Kubernetes (UKS)**, choose your credential and **Helsinki #2**, and give the node pool the `GPU-8xCPU-64GB-1xL4` plan. Mind the pool defaults here too: two nodes on the 12xCPU / €470 plan, so set one node and change the plan. UpCloud still owns the control plane and the node pools.

## Or create it in UpCloud and import it

In the UpCloud control panel open **Kubernetes** → **Create cluster**. Pick **Helsinki #2 (fi-hel2)**, the zone that stocks GPU plans, and give the cluster a node group on `GPU-8xCPU-64GB-1xL4` with one node. Wait until it shows as running.


## Import it into Ankra

**Clusters** → **Create Cluster**, then the **Already running managed Kubernetes?** link at the bottom of the dialog. Pick **UpCloud**, select your credential, hit **Discover** and import from the list: Ankra fetches the kubeconfig through the UpCloud API and installs the agent, nothing to run by hand. If Discover answers with a raw `incomplete_regions` validation error instead of a list, the portal is failing to parse the provider's reply — the Ankra-side UKS lane above avoids it.

> **Three things change on this path**
>
> The gpu-chat profile carries the GPU operator, so the drivers still arrive with the stack in stage 2. Load balancers and volumes come from UKS's own integrations, not the Ankra-installed CCM/CSI. And the generated ankra.cc domain is not part of an import: enable it with `ankra cluster domain`, and add the networking stack from **Stack Profiles** if the cluster has no ingress yet.

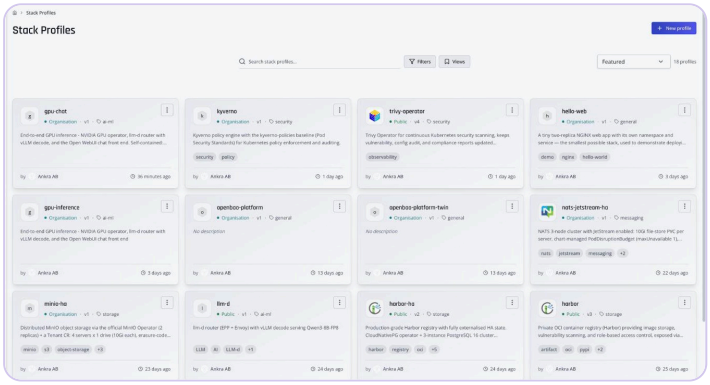
TERMINAL EQUIVALENT

```
ankra cluster managed discover --provider uks --credential-id <credential-id>
ankra cluster managed import --provider uks --credential-id <credential-id> \
  --provider-cluster-id <uks-cluster-id> --name gpu-chat
```

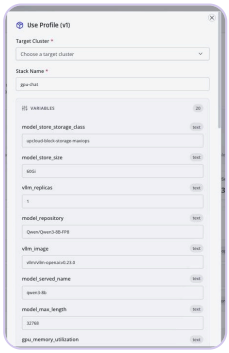
2 INSTANTIATE THE PROFILE

# gpu-chat: three values, guided

Open **Stack Profiles**: **gpu-chat** is the public, self-contained profile: it ships the Prometheus Operator CRDs it needs and defaults to the traefik class, the L4-sized `Qwen3-8B-FP8` and the maxiops storage class. Hit **Use Profile**, pick the cluster, and fill three values, each explained in the form: `public_domain` (your generated domain; the stack serves chat., api. and /grafana on it), the api key (a strong random value: the internal key between the broker, the UI and the model — nobody types it again), and a master key for the token broker (starts with `sk-`): it mints the per person API tokens on the next pages. Every other field says "leave it" and means it — the CPU embedding model (`bge-m3`) behind document RAG and `/v1/embeddings` included; once the cluster is picked, a **Use cluster domain** button fills `public_domain` for you. Use Profile creates a *draft* on the cluster: review it and hit **Create Stack**. The stack is **33 resources** deploying as **39 jobs** and lands in **five to eight minutes** (about three on a later redeploy), the GPU operator needs roughly **five more** before the L4 is schedulable, and vLLM then downloads the model into the volume, **five to ten minutes**, once ever.




**5.** The profile catalog, gpu-chat up front




**6.** Every field explains itself; three are yours

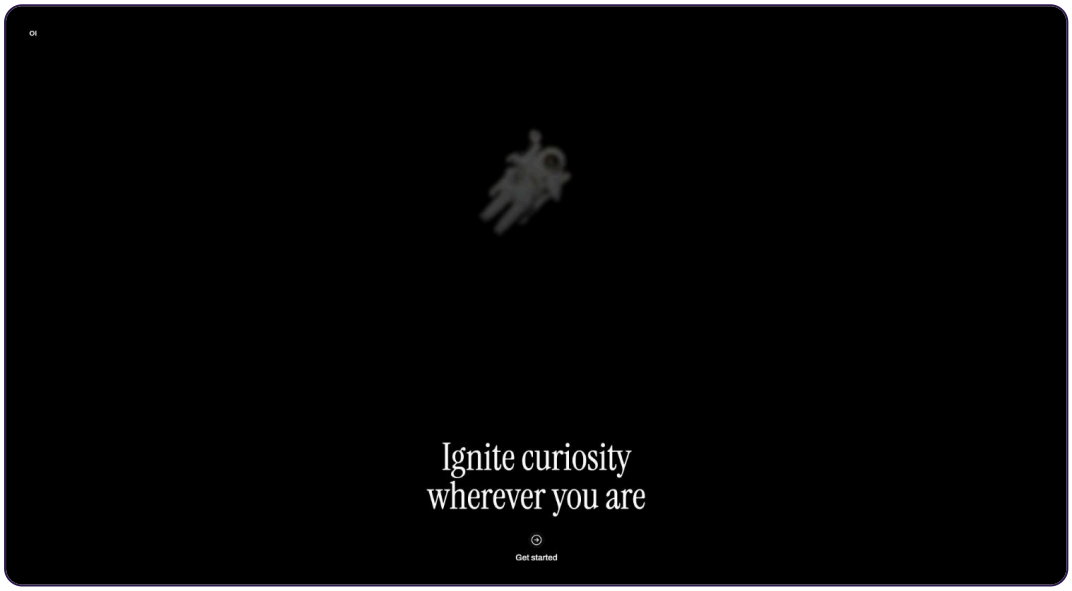
## TERMINAL EQUIVALENT

```
ankra stack-profiles apply gpu-chat --cluster <cluster> --deploy \
  --set public_domain=<your-domain> \
  --set-env manifest.llm-api-credentials.stringData.api-key=LLM_API_KEY \
  --set-env manifest.litellm-credentials.stringData.master-key=LITELLM_MASTER_KEY
```

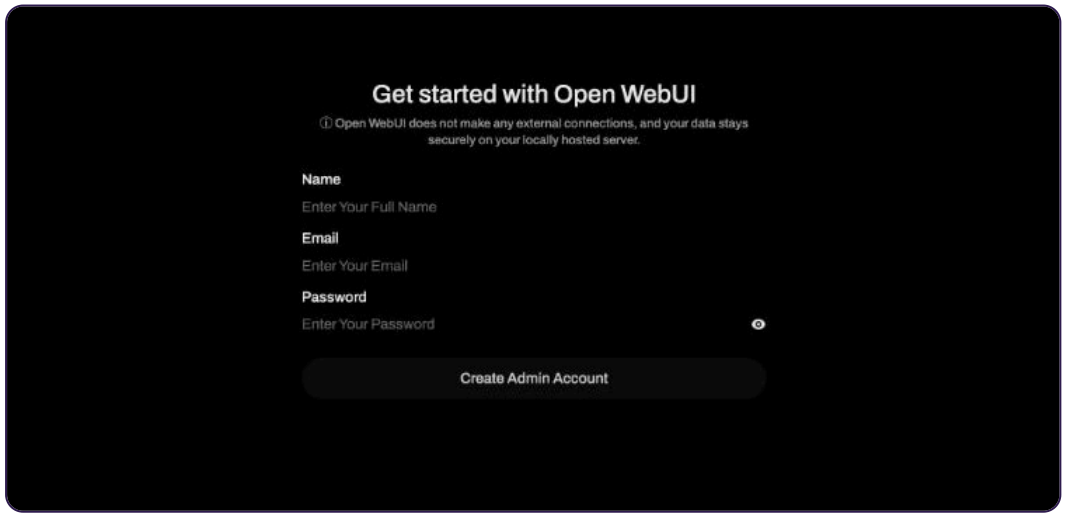
3 OPEN IT

# Chat on your own silicon

Open `https://chat.<your-domain>`. The DNS record and certificate were created while the stack deployed. If the model is not listed yet, vLLM is still pulling the ~9 GB of weights: first boot takes five to ten minutes, and every restart after reloads from the volume in about two. The first account you register becomes the admin; pick `qwen3-8b` and chat, the tokens come off your own L4. Upload a document with the **+** button and ask about it: chunked, embedded on the CPU (`bge-m3`), retrieved from the built-in Postgres — nothing leaves the cluster. The router gives you an OpenAI-compatible front door with prefix-cache-aware scheduling: chat resends the conversation every turn, and cached prefixes become skipped computation.




**7.** Live on your generated domain, padlock and all




**8.** First account becomes the admin

TERMINAL EQUIVALENT

```
curl https://api.<your-domain>/v1/chat/completions -H "Authorization: Bearer <minted-token>" \
  -d '{"model": "qwen3-8b", "messages": [{"role": "user", "content": "Hello"}]}'
```

MEASURED ON THE LIVE BUILD

# Auth, rate limits, and what a token costs

**Auth.** Every `/v1` route needs a broker-minted bearer token (or the master key), and behind it vLLM independently checks the internal model key: no unauthenticated path to the model, even inside the cluster. Verified: no key 401, wrong key 401, right key 200.

**Rate limits.** The API ingress carries a Traefik rate-limit middleware: 10 requests/second average, burst 100, per source IP (sized so the dashboard's own assets load through it); tune it in the stack's `api-ratelimit` manifest. Verified live with a stricter test limit: 30 parallel requests came back as 20× 200 and 10× 429.

| LOAD | THROUGHPUT | € / 1M OUTPUT TOKENS |
|---|---|---|
| Single stream | 25.3 tok/s | 5.86 GPU · 6.32 cluster |
| 8 concurrent requests | 185 tok/s aggregate | 0.80 GPU · 0.86 cluster |

Basis: €390/mo for the L4 (€0.53/h), about €420/mo for the cluster (€0.58/h), on Qwen3-8B-FP8. The 7× gap is the batching dividend: vLLM serves concurrent requests nearly for free until the GPU saturates, so the busier the endpoint, the cheaper the token.

## Give everyone their own token

From profile v4 the api host **is** a token broker: LiteLLM fronts the llm-d router, keeps the model key inside the cluster, and authenticates every request with a personal token. Mint one per person with the master key from the launch form:

```
curl -s -X POST https://api.<your-domain>/key/generate \
  -H "Authorization: Bearer <master-key>" -H "Content-Type: application/json" \
  -d '{"key_alias": "alice", "max_budget": 5}'
```

The reply's `key` field is the token you hand out; it works in any OpenAI client, for chat and `/v1/embeddings` (`bge-m3`, €0.02/1M) alike, on the same per-key meter. Usage by token (the holder can run this for their own key, with their own key as the bearer):

```
curl -s "https://api.<your-domain>/key/info?key=<their-key>" \
  -H "Authorization: Bearer <master-key>"
```

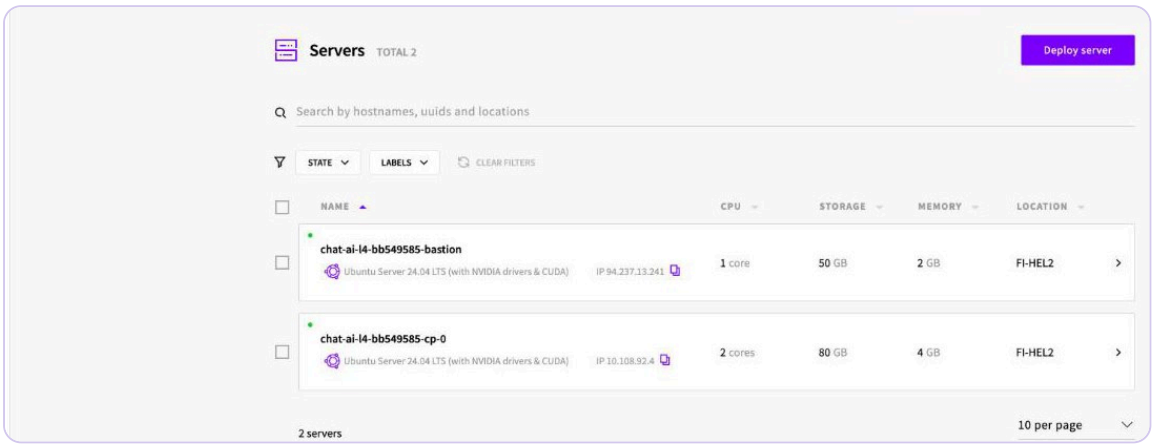
Past its budget a key gets `429`; revoke with `POST /key/delete`, the model key never changes. Two dashboards ship with the stack, both `admin` + the master key: `api.<your-domain>/ui` to manage keys, and `api.<your-domain>/grafana` to watch them: spend and budget per key, tokens per minute, plus vLLM and GPU.

> **Verified before it shipped**
>
> Keys authenticate, bogus keys 401, spend accrues per key, over-budget 429, both dashboards serve.

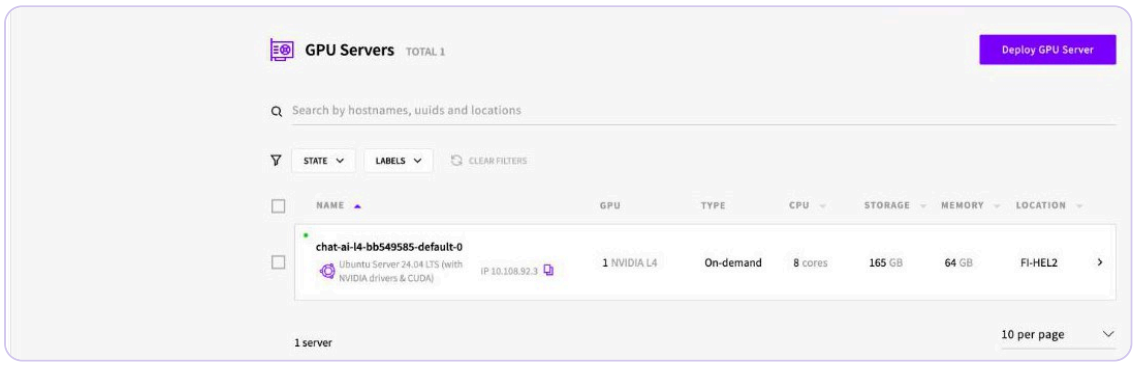
BRING YOUR OWN KEY

# Your resources, your UpCloud account

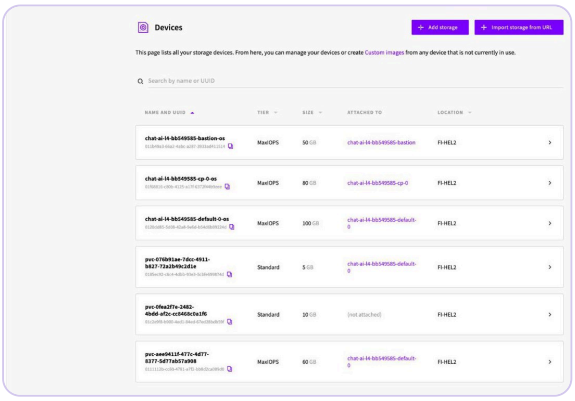
Everything Ankra created is visible in your UpCloud console: the bastion and control plane under Servers, the L4 under GPU Servers, and the OS disks plus the CSI volumes (the 60Gi model store among them) under Block Storage. You own all of it.




**9.** Bastion and control plane




**10.** The L4 node




**11.** OS disks and the CSI volumes

**Teardown, in the right order**

Delete the **gpu-chat** stack, reclaim its volumes, then **Terminate** from Settings → Danger Zone: the order, the timings and the verb that only looks like deprovisioning are on page 3. Check **Block Storage** afterwards — an unattached volume bills forever.

IF SOMETHING WEDGES

# Field notes

**GPU stuck at zero**

If the node reports `nvidia.com/gpu: 0` while the operator validator restart-loops, delete the validator and device-plugin pods **together** so they start in sync: `kubectl -n gpu-operator delete pod <validator> <device-plugin>`. Capacity appears within a minute. From v9 the operator autodetects containerd, which is what kubeadm needs; on a K3s cluster point the **gpu-operator** add-on's `CONTAINERD_CONFIG` and `CONTAINERD_SOCKET` at the K3s paths instead.

**Self-contained by design**

gpu-chat brings its own Prometheus Operator CRDs, and the platform pins the UpCloud load balancer to TCP passthrough so Traefik serves the real certificates out of the box.

**Qwen thinks before it answers**

Thinking tokens count against `max_tokens`; a small budget can return empty content. For short completions pass `"chat_template_kwargs": {"enable_thinking": false}`.

**Claim the chat admin account immediately**

The chat host is public the moment its certificate lands, and the **first person to register becomes its admin**: register yourself before sharing the URL. Plain `http://` is served too, so keys sent to an `http` base URL travel in clear text: hand out the `https` address only.

**Don't recycle the stack in a loop**

Every delete-and-recreate reissues both certificates, and Let's Encrypt allows **five per exact hostname set per week**. Past the limit Traefik serves a self-signed one: browsers warn, SDKs and `curl` refuse. cert-manager retries once the window rolls; redeploy in place rather than deleting.

## What you built

A private, TLS-terminated AI chat service and OpenAI-compatible API on your own GPU, on a domain generated for you, deployed from one public profile: three values, and about thirty five minutes from Create Cluster to the first answer.

| PLATFORM | DOCUMENTATION | UPCLOUD |
|---|---|---|
| platform.ankra.app | docs.ankra.ai | hub.upcloud.com |