

Sino Control Hub Deployment Guide

Running your company's own Control Hub: what you need, Docker Compose on your own server, settings, databases, proxies, sizing, backups, integrations and day-to-day tasks

Sino Privacy Vault (<https://sinosecure.app>)

1. Before You Start

The Control Hub is how a company enrolls, configures and audits its Sino vaults. Your company runs it, on its own server or in its own cloud. **Sino runs no server and never sees your hub.** The hub is a control plane, never a data plane: files go straight from each device to your storage, encrypted, and never pass through the hub. The hub holds public keys, the device list, policy and audit events, and cannot open a vault.

You need:

- **A place to run one container.** Your own server with Docker, Kubernetes, Azure Container Apps, AWS, Google Cloud Run or Cloudflare Containers (section 2).
- **A permanent DNS name** for the hub, such as `hub.example.com`. Every enrollment link and every device keeps it, so choose it once.
- **HTTPS in front of the hub.** Caddy (included in section 3), your ingress controller, or the cloud's load balancer.
- **A database.** A SQLite file on a volume, or your own Postgres, MySQL or MariaDB (section 5).
- **Storage for the vaults.** AWS S3, Backblaze B2, Cloudflare R2 or Azure Blob Storage that the company owns. You add it in the console after setup.
- **A safe place for the master key.** It seals the hub's keys and every credential the hub stores. Without it the database cannot be opened.
- **The hub image.** The Control Hub is in early access: the image (`sino-hub:<version>`) and its deploy bundle (Compose files, Caddyfile, Helm chart and Microsoft Sentinel templates) come from Sino. Write to `support@sinosecure.app`.

Wherever it runs, keep exactly **one** hub process. Its timers, rate limits and setup code live in that process, so two copies must never run at once. Point health checks at `GET /healthz`, which answers 200 while the database does.

2. Where to Run It

Where	Guide	Database
Your own server	Section 3 of this guide	SQLite file, Postgres, MySQL or MariaDB
Kubernetes (AKS, EKS, GKE, on-premises)	<i>Control Hub on Kubernetes</i>	Your Postgres, MySQL or MariaDB
Azure Container Apps	<i>Control Hub on Azure</i>	Azure Database for PostgreSQL or for MySQL
AWS (ECS Fargate or App Runner)	<i>Control Hub on AWS</i>	Amazon RDS for PostgreSQL, MySQL or MariaDB
Google Cloud Run	<i>Control Hub on Google Cloud</i>	Cloud SQL for PostgreSQL or for MySQL
Cloudflare Containers	<i>Control Hub on Cloudflare</i>	A database server outside Cloudflare (required)
Anything else that runs a container	The settings in section 4	SQLite on a volume, Postgres, MySQL or MariaDB

The cloud guides have not yet been checked against live cloud accounts. Tell us anything that differs.

3. On Your Own Server with Docker Compose

1. **Load the image** Sino sent you, so `docker images` lists `sino-hub`.
2. **Copy the Compose files** from the deploy bundle (also in Appendix A) to the server: `compose.yaml` and `Caddyfile`, plus `compose.postgres.yaml` or

`compose.mariadb.yaml` if you want a database server. Point the hub's DNS name at the server, and open ports 80 and 443.

3. **Make the master key**, and back it up somewhere safe:

```
mkdir -p secrets && openssl rand -base64 32 > secrets/master_key &&  
chmod 600 secrets/master_key
```

4. **Start it** with the hub's permanent address:

```
SINO_HUB_DOMAIN=hub.example.com docker compose up -d  
docker compose logs hub
```

The log shows a one-time **setup code**. Open `https://hub.example.com`, enter it, and register your passkey: that makes you the owner. The code stops working once used, and a new one is printed at every start until then.

5. **Sign the hub's certificate**. In the console, under **Hub setup**, make the setup link and open it in the Sino app on your own device. The app runs the kit ceremony: it makes your organisation kits, which your admins keep, and signs the hub's certificate with them. Then enroll the first vault from **Enroll a vault**.

With Postgres: `openssl rand -base64 24 > secrets/db_password`, then `docker compose -f compose.yaml -f compose.postgres.yaml up -d`. With MariaDB, the same with `compose.mariadb.yaml`.

4. Settings

Variable	Meaning
<code>SINO_HUB_MASTER_KEY</code> or <code>SINO_HUB_MASTER_KEY_FILE</code>	32 random bytes, base64. Required
<code>SINO_HUB_DATABASE_URL</code>	<code>jdbc:postgresql://host:5432/db</code> , <code>jdbc:mysql://host:3306/db</code> or <code>jdbc:mariadb://host:3306/db</code> ; without it the hub uses <code>SINO_HUB_DATA/hub.db</code>

<code>SINO_HUB_DATABASE_USER</code> , <code>SINO_HUB_DATABASE_PASSWORD</code> or <code>..._PASSWORD_FILE</code>	The database account
<code>SINO_HUB_DATA</code>	The data directory for the SQLite file (<code>/data</code> in the image)
<code>SINO_HUB_PORT</code>	HTTP port, 8080 by default. Serve it only through your HTTPS proxy
<code>SINO_HUB_TRUST_PROXY</code>	<code>true</code> when the proxy appends the client's address to <code>X-Forwarded-For</code> , so rate limits count per client
<code>SINO_HUB_CONSOLE</code>	A directory of console pages to serve instead of the built-in ones

The `..._FILE` settings read a secret from a file, which is how Docker and Kubernetes mount secrets. Prefer them to putting secrets in the environment.

5. Databases

- **SQLite.** No server to run, for a smaller hub on one machine. The hub keeps the file in WAL mode and holds it for as long as it runs, so a second copy of the hub on the same file stops at start.
- **PostgreSQL.** Make an empty database and an account that owns it; the hub makes its tables at start. Add `?sslmode=require` to the address.
- **MySQL 8.0.16 or later, or MariaDB 10.5 or later.** Make an empty database with the `utf8mb4` character set and an account that can create tables in it; the hub makes its tables (InnoDB, binary collation) at start, and talks to both through MariaDB Connector/J.
 - **TLS:** add `sslMode=verify-full` to the address. When the server's certificate comes from a private CA (Amazon RDS, Cloud SQL, your own), add `serverSslCert=<path to the CA's PEM file>` too, or use `sslMode=verify-ca` when the certificate does not name the host you connect to.
 - **MySQL 8 without TLS** (a test server only) needs `allowPublicKeyRetrieval=true` to sign in.

SQL Server, Oracle and Cloudflare D1 are not supported.

6. Proxies and Load Balancers

While Sino is open on a Windows PC, it keeps a call open to the hub, so commands reach it within seconds:

- **A stream** of up to 3 minutes (`/device/v1/stream` , server-sent events with a keep-alive line every 20 seconds). The device signs one call per stream.
- **Held waits** of up to 25 seconds (`/device/v1/wait`) where a proxy holds streams back. Sino notices and switches by itself.

Every proxy and load balancer in front of the hub must:

- **Pass `text/event-stream` answers as they come.** Caddy, AWS and Azure load balancers, Cloud Run and Cloudflare do. nginx does for the hub, which sends `X-Accel-Buffering: no` . A proxy that buffers them still works, with held waits.
- **Allow a request to last 3 minutes, with up to 20 seconds of silence.** The defaults of Caddy, nginx (60 seconds of silence), AWS load balancers (60), Cloudflare (100), Azure Container Apps (240 for a request) and Cloud Run (300) all do.
- **Accept request bodies of 4 MB.** Devices send audit uploads of up to about 1 MB. With ingress-nginx, add `nginx.ingress.kubernetes.io/proxy-body-size: 4m` .
- **Append the client's address to `X-Forwarded-For` , and set `SINO_HUB_TRUST_PROXY=true` .** If nothing in front sets the header, leave it `false` : otherwise a client could dodge the rate limits with a forged header.

The console's files are cached by browsers (a year for its scripts and styles, a day for fonts and icons) and sent compressed when the browser asks. The API's answers are never cached or compressed. **Leave compression off for them in the proxy too:** their secrets sit next to what a caller sent, which compression can leak (the BREACH attack).

7. Sizing and Performance

The hub uses a few hundred MB of memory. Half a CPU and 1 GB of memory are enough to start; give it a full CPU for a large company.

Measured on one 8-core PC, with the hub, its database and 2,000 simulated PCs on the same machine, all connecting at once:

	SQLite	MySQL 8.4
Database work while 2,000 PCs keep streams open	22 transactions a second	22 transactions a second
Heartbeats handled at the same time	580 a second	185 a second
Time for a command to reach its PC	9 ms typical, 59 ms at most	12 ms typical, 57 ms at most
Failed calls	None	None

With held waits instead of streams, the open PCs cost about 165 transactions a second, and heartbeats went through at 490 (SQLite) and 150 (MySQL) a second.

Devices check in once an hour by default (every 15 minutes to once a day, set in the policy), so either database serves many thousands of PCs. The hub uses one database connection, so a database server across a network is slower than these figures: keep it close to the hub.

8. Backups and Recovery of the Hub

Back up two things:

- **The master key**, outside the platform the hub runs on. Without it the database cannot be opened, and the hub's keys and every stored credential are lost.
- **The database**. Back up Postgres, MySQL or MariaDB as you back up any other database. For SQLite, copy `hub.db` together with `hub.db-wal` beside it, from a stopped hub or a snapshot of the volume; a copy of `hub.db` alone misses the latest changes.

Losing the hub never loses data. Files and recovery copies live in your storage, and the kits stay with your admins. A lost hub loses its device list and recent audit events. Rebuild it, sign its new certificate with the kits, and devices reconnect at their next check-in, without enrolling again.

While the hub is down, people keep working: vaults open as usual, and sync goes on for as long as the storage access each device holds lasts, then waits for the hub. That access lasts 8 hours by default (1 hour on AWS S3), set on each storage target. Commands, policy changes and audit uploads also wait until the hub is back. A device that cannot reach the hub for the policy's **Lock after this many hours without reaching the hub** (336 hours, 14 days, by default) locks, and unlocks again only after reaching it. If you turn on **Unlock only while the**

hub can be reached, nobody can unlock while the hub is down: keep the hub highly available first.

The hub is one process by design, not an active-active cluster. Kubernetes and the cloud platforms restart it within seconds if it stops, and devices retry by themselves.

9. Microsoft Sentinel

1. Deploy `sentinel/sino-sentinel.json` from the deploy bundle to the resource group of your Sentinel workspace. It makes the `SinoHub_CL` table, a data collection endpoint and a rule.
2. Register an app in Entra ID with a client secret, and give it the **Monitoring Metrics Publisher** role on the rule.
3. In the console, add a Sentinel destination with the tenant id, the app's client id and secret, and the template's outputs (`ingestionEndpoint` , `ruleId` , `stream`). **Send a test event** checks it.
4. Optionally, deploy `sentinel/sino-sentinel-content.json` to the same workspace. It adds:
 - the ASIM parsers `vimAuthenticationSinoHub` and `vimAuditEventSinoHub` ;
 - scheduled analytics rules for approved destructive actions, several devices wiped, a broken audit chain, repeated wrong passwords, recovery and key changes, refused commands and refused console sign-ins;
 - a **Sino Control Hub** workbook.

`sentinel/queries.kql` has sample queries. The content follows Microsoft's published schemas but has not yet been deployed to a live workspace.

10. Splunk, Syslog and Webhooks

- **Splunk.** Make an HTTP Event Collector token, then add a Splunk destination in the console with the collector's address (`https://splunk.company.com:8088` , no path), the token and, if you like, an index. Each event arrives as one HEC event, with sourcetype `sino:ocsf` , the hub as host, and the OCSF event as its body.
- **Syslog** goes over TLS (RFC 5425) with CEF messages, to port 6514 by default. Give your CA's certificate if the server's certificate is not publicly trusted.

- **Webhooks** receive `{"events": [...]}` in OCSF, signed: `X-Sino-Timestamp` (Unix seconds) and `X-Sino-Signature: v1=<hex>`, the HMAC-SHA256 of `<timestamp>`. `<body>` under the key the console shows once. Refuse timestamps more than five minutes old.
-

11. Console Sign-In with Company Accounts

Admins always have a passkey. Under **Hub setup**, an owner can also turn on sign-in with the company's own provider; each admin then links their account once from **Your account**, while signed in with their passkey. **Approvals always need the passkey**, and company sign-in never unlocks a vault.

- **Microsoft Entra ID.** Register a web app in your tenant with the redirect address the page shows (`https://<hub>/v1/auth/entra/callback`) and a client secret. Enter the tenant id, client id and secret.
 - **Google Workspace.** In a Google Cloud project whose OAuth consent screen is **Internal**, make an OAuth client of type **Web application** with the redirect address the page shows (`https://<hub>/v1/auth/google/callback`). Enter your Workspace's primary domain, the client id and its secret. Only accounts of that domain can sign in.
 - **SAML 2.0** (Okta, AD FS, Google Workspace, Entra ID and others). Add a SAML app in your identity provider with the hub's entity id and metadata, `https://<hub>/v1/auth/saml/metadata` , and its reply (ACS) address, `https://<hub>/v1/auth/saml/acs` . Have it sign assertions with SHA-256, and send a name id that never changes (a persistent id or an immutable user id, rather than an e-mail address that might change). Enter its entity id (issuer), its single sign-on address and its signing certificate (PEM). When your provider rotates its signing certificate, paste the new one in; links stay.
-

12. People from Entra ID (SCIM)

Under **People**, turn on SCIM: the console shows a tenant URL (`https://<hub>/scim/v2`) and a token, once. In Entra ID, add a non-gallery enterprise application, set provisioning to automatic, and enter both. Entra ID then adds, updates and removes people in the hub. When it removes someone, their devices are locked, lose their storage access or get a remote wipe request (which the approving admins still decide), as **Hub setup** says.

13. API Keys

Owners make API keys under **API keys**, each with scopes. Scripts send `Authorization: Bearer sino_...`. The API is described at `https://<hub>/openapi/hub-api.yaml`, and the console uses nothing else, so your own tools can do anything it does. A key can ask for a remote wipe, but only the approving admins' passkeys approve it.

14. Lost or Stolen Devices

Open the device under **Devices**, then:

1. **Lock the vault** right away (no approval needed), and **Ask for a remote wipe** (the approvers decide). A command reaches a Windows PC with Sino open within seconds, and any other device at its next check-in while it is on and online. The policy's **Minutes between check-ins** sets how often that is: 60 by default, 15 at the soonest.
2. **Stop its storage access**, so it cannot sync even before the wipe runs.
3. Do not **Remove from the hub** until the wipe has run: a removed device gets no more commands.

A device kept offline locks itself after the policy's **Lock after this many hours without reaching the hub**, an open vault too, and unlocks only after reaching the hub.

With an MDM (Intune or another). An MDM reaches a phone through Google's or Microsoft's push service, which Sino does not use, so it is the fastest way to wipe a lost phone that is not checking in. Wiping the device, or on Android the work profile Sino is installed in, removes Sino and its vault with it. The files stay in your storage. The hub does not hear of an MDM wipe: the device stops checking in and is flagged as silent. Remove it under **Devices** once the MDM reports the wipe.

15. Someone Leaving with Company Data

- **Unlock only while the hub can be reached.** Every unlock checks in first, so once you remove or wipe a device it never opens again, even if it goes offline afterwards. Combine it with a short **Lock after this many idle minutes**, so an open vault does not stay open. While the hub is down, or a device has no network, nobody can unlock.
- **Taking files out of the vault.** Restoring files to the device, saving Local Vault files or shared folder copies outside Sino, and the Windows drive (or other Android apps opening

vault files) each have a rule: allowed, **the hub confirms** (the default), **an admin approves each time**, or off (the apps then hide it).

- "The hub confirms" lets the action run only after the hub's signed yes. It is recorded in the audit log and your SIEM, refused for a device whose storage access you stopped, and refused past **Files a vault may take out in 24 hours**.
- "An admin approves" sends each request to **File requests** in the console, showing who, what, how much (never file names) and the reason the person typed. Sino tells the person the answer, and once approved they do it again within 24 hours, once.
- **Admins with a vault of their own:** on **Admins**, link each one to their person in the directory. Another admin then answers their requests.
- **Apps trusted to open vault files.** These open vault files without asking anyone, whatever the rule says; the audit log shows each file they open. Apps devices saw trying are listed under **Seen trying to open vault files** with a **Trust** button. An Android app is its package and signing certificate; a Windows program is its file name and publisher, so a renamed or unsigned copy is not trusted. Trusting an app trusts what it does with the file: Word can save a copy to OneDrive or attach it to an e-mail.
- **Screenshots and screen recording of Sino** are always blocked (on by default), and people cannot turn the protection off.

None of this stops someone photographing the screen, retyping what they see, or using a file once another app has opened it.

16. Shared Folders

People share folders with colleagues from their Sino app. Nothing needs setting up beyond organisation storage: each folder lives in your first storage target, and the hub gives each member credentials for that folder alone, read-only for viewers. Folders are end-to-end encrypted: the console's **Shared folders** page shows members and roles, never names or files.

- **Turning it off:** the policy's **Allow shared folders** stops new folders and every member's access.
- **Deleting a folder:** an owner deletes it from the app, or admins ask under **Shared folders** and the approvers approve. Its storage is erased 30 days later.
- **Owners who left:** download the folder's recovery file under **Shared folders**, open it with enough kits in an admin's Sino app (**Work vault** → **Shared folders** → **Recover a folder**

whose owners left), name a new owner, and paste the app's answer back on the page.

- **Without Sino:** the open-source Sino Open SDK reads a copy of the folder's storage with the kits.
-

17. Not Yet Supported

- **Commands while the app is closed.** There is no push: a closed app runs no code, and push would need a Sino relay server. Commands wait for the next check-in. Use an MDM to wipe a lost phone that is not checking in.
 - **Google Cloud Storage** as a storage target. Use Azure Blob, AWS S3, Backblaze B2 or Cloudflare R2.
 - **SCIM groups.** Entra ID provisions people, not groups.
 - **SAML** encrypted assertions, single logout and sign-ins started from the identity provider's portal. Admins start from the hub's sign-in page.
 - **iPhone, iPad and Mac.** Managed vaults are for Android and Windows.
 - **More than one hub process** for the same organisation.
-

Appendix A. Compose Files

`compose.yaml`

```
# The Sino Control Hub on one server: the hub, and Caddy in front for
# HTTPS (it gets and renews the certificate
# itself). The hub keeps its data in a SQLite file on the hub-data
# volume; add compose.postgres.yaml for Postgres or
# compose.mariadb.yaml for MariaDB.
name: sino-hub

services:
  hub:
    image: sino-hub:${SINO_HUB_VERSION:-latest}
    restart: unless-stopped
    environment:
      SINO_HUB_MASTER_KEY_FILE: /run/secrets/sino_master_key
      # Caddy appends the client's address to X-Forwarded-For, so rate
limits count per client.
      SINO_HUB_TRUST_PROXY: "true"
    secrets: [sino_master_key]
    volumes:
      - hub-data:/data
    read_only: true
    tmpfs: [/tmp]
    cap_drop: [ALL]
    security_opt: ["no-new-privileges:true"]
    # The hub is reached only through Caddy.
    expose: ["8080"]

  caddy:
    image: caddy:2
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
      - "443:443/udp"
    environment:
      SINO_HUB_DOMAIN: ${SINO_HUB_DOMAIN:?Set SINO_HUB_DOMAIN to the
hub's permanent address, for example hub.example.com}
    volumes:
```

```
- ./Caddyfile:/etc/caddy/Caddyfile:ro
- caddy-data:/data
- caddy-config:/config
depends_on: [hub]
```

secrets:

```
sino_master_key:
  file: ./secrets/master_key
```

volumes:

```
hub-data:
caddy-data:
caddy-config:
```

Caddyfile

```
# HTTPS in front of the hub. Caddy gets the certificate for the domain
and renews it, and replaces any
# X-Forwarded-For a client sends with the client's own address.
{$SINO_HUB_DOMAIN} {
    reverse_proxy hub:8080
}
```

compose.postgres.yaml

```
# Postgres instead of the SQLite file. Use it on top of compose.yaml:
# openssl rand -base64 24 > secrets/db_password
# docker compose -f compose.yaml -f compose.postgres.yaml up -d
# To use the company's own Postgres instead, leave out the db service
and point SINO_HUB_DATABASE_URL at it.
services:
  hub:
    environment:
      SINO_HUB_DATABASE_URL: jdbc:postgresql://db:5432/sino_hub
      SINO_HUB_DATABASE_USER: sino_hub
      SINO_HUB_DATABASE_PASSWORD_FILE: /run/secrets/sino_db_password
    secrets: [sino_master_key, sino_db_password]
    depends_on:
      db:
        condition: service_healthy

  db:
    image: postgres:17
    restart: unless-stopped
    environment:
      POSTGRES_DB: sino_hub
      POSTGRES_USER: sino_hub
      POSTGRES_PASSWORD_FILE: /run/secrets/sino_db_password
    secrets: [sino_db_password]
    volumes:
      - db-data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U sino_hub -d sino_hub"]
      interval: 5s
      retries: 20

secrets:
  sino_db_password:
    file: ./secrets/db_password
```

```
volumes:  
  db-data:
```

compose.mariadb.yaml

```
# MariaDB instead of the SQLite file. Use it on top of compose.yaml:
# openssl rand -base64 24 > secrets/db_password
# docker compose -f compose.yaml -f compose.mariadb.yaml up -d
# To use the company's own MySQL or MariaDB instead, leave out the db
service and point SINO_HUB_DATABASE_URL at it
# (jdbc:mysql://... or jdbc:mariadb://...).
services:
  hub:
    environment:
      SINO_HUB_DATABASE_URL: jdbc:mariadb://db:3306/sino_hub
      SINO_HUB_DATABASE_USER: sino_hub
      SINO_HUB_DATABASE_PASSWORD_FILE: /run/secrets/sino_db_password
    secrets: [sino_master_key, sino_db_password]
    depends_on:
      db:
        condition: service_healthy

  db:
    image: mariadb:11.4
    restart: unless-stopped
    command: ["--character-set-server=utf8mb4", "--collation-
server=utf8mb4_bin"]
    environment:
      MARIADB_DATABASE: sino_hub
      MARIADB_USER: sino_hub
      MARIADB_PASSWORD_FILE: /run/secrets/sino_db_password
      MARIADB_RANDOM_ROOT_PASSWORD: "1"
    secrets: [sino_db_password]
    volumes:
      - db-data:/var/lib/mysql
    healthcheck:
      test: ["CMD", "healthcheck.sh", "--connect", "--
innodb_initialized"]
      interval: 5s
      retries: 20

secrets:
```

```
sino_db_password:  
  file: ./secrets/db_password
```

```
volumes:  
  db-data:
```