Deploying Chariot with Docker

This page explains how to deploy Chariot using the official Docker image. By the end, you will have a running Chariot instance and understand how to configure it for your environment — from a quick local test to a production deployment with TLS and managed licensing.

Prerequisites

Before you start, make sure you have:

- **Docker Engine 20.10+** (or Docker Desktop) installed and running
- **Docker Compose v2** if you plan to use a Compose file (recommended for anything beyond a quick test)
- Network access to pull from Docker Hub (`hub.docker.com/r/cirruslink/chariot`)
- A valid **Chariot license key or license server address** (required for production; the image starts without one for evaluation purposes)
- Outbound internet access from the container host if you are using **online license activation**

How configuration works

All configuration is applied **once, automatically, on the first startup** of the container. The `configure.sh` script runs in the background alongside the Chariot process and works through a defined sequence of steps via the Chariot REST API.

A sentinel file at `/Chariot/config-state/state.json` inside the container records which steps have completed. On subsequent restarts, completed steps are skipped — **changing an environment variable after first run will have no effect** unless you reset the relevant step in the sentinel file. See Re-applying configuration for how to do this.

Configuration is applied in this order:

1. Accept EULA
2. Restore backup (if `BACKUP_FILE` is set)
3. Set admin password
4. Activate license
5. Apply MQTT server configuration
6. Upload IBSnow certificates
7. Create IBSnow servers
8. Apply IBSnow configuration
9. Create MQTT users
10. Configure TLS/SSL
11. Apply system configuration (server name, ports)

> **Backup first, variables win.** When a backup is restored, it runs before other configuration steps. Environment variables applied afterward take precedence — including `ADMIN_PASSWORD`. If your backup contains a different admin password, set `ADMIN_PASSWORD` to ensure you can log in after restore.

> `ACCEPT_EULA=true` is required for configuration. If this variable is not set to `true`, the script exits immediately and no configuration is applied — the container starts with factory defaults only.

Quick start: running Chariot locally

The following command starts a Chariot instance suitable for local testing. It is **not recommended for production** — see Production deployment for a more complete setup.

```
docker run -d \
  --name chariot \
  -p 8080:8080 \
  -p 1883:1883 \
  -e ACCEPT_EULA=true \
  -e ADMIN_PASSWORD=mypassword \
  cirruslink/chariot:latest
```

Once the container starts, wait approximately 60 seconds for Chariot to initialize, then open `http://localhost:8080` in your browser. Log in with:

- **Username:** `admin`
- **Password:** the value you set for `ADMIN_PASSWORD`

> ⚠ **If you do not set `ADMIN_PASSWORD`**, the default credentials (`admin` / `password`) remain active. Change this on any network-accessible instance before connecting devices.

To follow the configuration log as it runs:

```
docker logs -f chariot
```

Configuration log lines are prefixed with `[configure.sh]`. A successful run ends with: `[configure.sh] Configuration complete. Logged out.`

Production deployment

For production, use Docker Compose with an `.env` file to keep configuration out of your Compose file. The example below uses the structure from the official `.env.example`.

Step 1 — Create your `.env` file:

```
cp .env.example .env
```

Edit `.env` and populate at minimum:

```
IMAGE_TAG=latest
ACCEPT_EULA=true
ADMIN_PASSWORD=your-strong-password-here
```

> ⚠ **Never commit your `.env` file to version control** if it contains passwords or license keys.

Step 2 — Create your `docker-compose.yml`:

```
services:
  chariot:
    image: cirruslink/chariot:${IMAGE_TAG}
    ports:
      - "8080:8080"   # HTTP Web UI and REST API
      - "8443:8443"   # HTTPS Web UI and REST API
      - "1883:1883"   # MQTT
      - "8883:8883"   # MQTT over TLS
      - "8090:8090"   # MQTT over WebSocket
      - "8091:8091"   # MQTT over Secure WebSocket
    environment:
      ACCEPT_EULA: "${ACCEPT_EULA}"
      ADMIN_PASSWORD: "${ADMIN_PASSWORD}"
      LICENSE_TYPE: "${LICENSE_TYPE:-}"
      LICENSE_KEY: "${LICENSE_KEY:-}"
      LICENSE_SERVER: "${LICENSE_SERVER:-}"
      LICENSE_PRODUCT: "${LICENSE_PRODUCT:-}"
      SERVER_CONFIG: "${SERVER_CONFIG:-}"
      MQTT_USERS: "${MQTT_USERS:-}"
      SETUP_SSL: "${SETUP_SSL:-}"
      TLS_PRIVATE_KEY: "${TLS_PRIVATE_KEY:-}"
      TLS_CERTIFICATE: "${TLS_CERTIFICATE:-}"
      TLS_CA_CHAIN: "${TLS_CA_CHAIN:-}"
    restart: unless-stopped
```

Step 3 — Start the service:

```
docker compose up -d
```

Port reference

Port	Protocol	Purpose
8080	HTTP	Web UI and REST API
8443	HTTPS	Secure Web UI and REST API
1883	MQTT	Standard MQTT listener
8883	MQTTS	MQTT over TLS
8090	WS	MQTT over WebSocket
8091	WSS	MQTT over Secure WebSocket

You only need to expose the ports relevant to your deployment. If you are not using WebSocket clients, for example, omit ports 8090 and 8091.

> ⚠ **If you change a listener port via `SERVER_CONFIG`**, update your `ports` mapping in the Compose file to match the new container-side port. For example, if you set `"port": 9999` in `SERVER_CONFIG`, your mapping should be `"1883:9999"` — not `"1883:1883"`.

Environment variable reference

General

Variable	Required	Default	Description
`ACCEPT_EULA`	**Yes**	—	Must be `true if configuration is desired`. No configuration is applied if unset or set to any other value.
`ADMIN_PASSWORD`	Recommended	`password`	Password for the `admin` account.

MQTT users

Use `MQTT_USERS` to provision MQTT client accounts on first run. You can provide user definitions inline as a JSON array, or as a path to a JSON file mounted into the container.

Option 1 — Inline JSON (suitable for a small number of users):

```
MQTT_USERS='[{"username":"myuser","password":"mypassword","acl":{"subscribeTopics":["#"],"publishTopics":
["#"]}}]'
```

Option 2 — JSON file (recommended when managing multiple users or keeping credentials out of your Compose file):

```
MQTT_USERS=/config/mqtt-users.json
```

Mount the file in your Compose service:

```
volumes:
  - ./mqtt-users.json:/config/mqtt-users.json:ro
```

JSON structure:

```
[
  {
    "username": "myuser",
    "password": "mypassword",
    "acl": {
      "subscribeTopics": ["devices/#"],
      "publishTopics":  ["devices/#"]
    }
  }
]
```

The `acl` object controls which MQTT topics each user can publish to and subscribe from. Use `["#"]` to grant access to all topics, or restrict access using standard MQTT wildcard patterns (+ for single level, # for multi-level).

Licensing

Variable	Required	Description
`LICENSE_TYPE`	No	`online` or `floating`. Omit to skip license activation.
`LICENSE_KEY`	If `LICENSE_TYPE=online`	License key for online activation. Requires outbound internet access.
`LICENSE_SERVER`	If `LICENSE_TYPE=floating`	Address of the floating license server.
`LICENSE_PRODUCT`	If `LICENSE_TYPE=floating`	Product identifier for the floating license, either `mqtt-server-ibsnow` , `mqtt-server` , or `ibsnow`

Choosing a license type:

- **Online** — Contacts Cirrus Link's license server at startup. Simplest option when outbound internet access is available.
- **Floating** — Connects to a dedicated license server. Use this when managing multiple Chariot instances that share a license pool, or when you want centralized license control regardless of where your instances are deployed.

MQTT server configuration

`SERVER_CONFIG` accepts a JSON object that is merged onto the current MQTT server configuration. Include only the fields you want to override. A full field reference will be available in the Chariot REST API documentation.

```
SERVER_CONFIG='{"allowAnonymous":false,"port":1883,"securePort":8883}'
```

System configuration

Variable	Required	Default	Description
`CHARIOT_SERVER_NAME`	No	—	Display name shown in the Web UI.
`HTTP_PORT`	No	`8080`	HTTP listener port for the Web UI and REST API.
`HTTPS_PORT`	No	`8443`	HTTPS listener port.
`HTTPS_ENABLED`	No	—	Set to `true` to enable HTTPS.

TLS / SSL setup

To enable TLS, mount your certificate files into the container and reference them by their container paths.

Step 1 — Add your certificate files to the Compose volume:

```
volumes:
  - ./certs:/certs:ro
```

Step 2 — Set the TLS environment variables:

```
environment:
  SETUP_SSL: "true"
  TLS_PRIVATE_KEY: /certs/privkey.pem
  TLS_CERTIFICATE: /certs/cert.pem
  TLS_CA_CHAIN: /certs/chain.pem
```

Variable	Required when	Description
`SETUP_SSL`	—	Set to `true` to enable TLS configuration.
`TLS_PRIVATE_KEY`	`SETUP_SSL=true`	Container path to the private key file (`.pem`).
`TLS_CERTIFICATE`	`SETUP_SSL=true`	Container path to the certificate file (`.pem`).
`TLS_CA_CHAIN`	`SETUP_SSL=true`	Container path to the CA chain file (`.pem`).

> All three TLS variables must be set together. If any are missing, the SSL step will log an error and skip. Once TLS is configured, HTTPS is available on port 8443 and MQTT over TLS on port 8883.

Restoring from a backup

Chariot supports full configuration backups as `.zip` files exported from the Web UI. To restore one on first run:

Step 1 — Mount the backup file:

```
volumes:
  - ./my-backup.zip:/backups/backup.zip:ro
```

Step 2 — Set the environment variable:

```
environment:
  BACKUP_FILE: /backups/backup.zip
```

The backup is restored before any other configuration step. Settings provided via environment variables — including `ADMIN_PASSWORD` — are applied afterward and will override anything in the backup.

After a successful restore, the sentinel records the backup filename and timestamp. You can view this with `docker exec chariot cat /Chariot/config-state/state.json`.

IBSnow (Snowflake IoT Bridge)

IBSnow connects Chariot to Snowflake for IoT data streaming. The variables below configure the integration on first run.

Variable	Description
`IBSNOW_CONFIG`	JSON object merged onto the current IBSnow configuration. Include only the fields you want to override.
`IBSNOW_SERVERS`	JSON array of MQTT server connections to create, or an absolute path to a JSON file.
`IBSNOW_CERTS`	JSON array of absolute container paths to certificate files to upload (e.g. `.p8`, `.pem`).

Configuration example (common fields only — a full field reference will be available in the Chariot REST API documentation):

```
{
  "snowflake_application_enabled": true,
  "ibsnow_instance_name": "prod-1",
  "ibsnow_cloud_region": "us-east-1",
  "streaming_profile_account": "xy12345",
  "streaming_profile_host": "xy12345.snowflakecomputing.com",
  "streaming_profile_database": "cl_bridge_stage_db",
  "streaming_profile_schema": "stage_db",
  "streaming_profile_warehouse": "cl_bridge_ingest_wh",
  "streaming_profile_role": "cl_bridge_process_rl"
}
```

Pass this as a single-line value in your `.env` file:

```
IBSNOW_CONFIG={"snowflake_application_enabled":true,"ibsnow_instance_name":"prod-1","
streaming_profile_account":"xy12345"}
```

`Servers example:`

```
IBSNOW_SERVERS='[{"url":"tcp://localhost:1883","name":"Server 1","subscriptions":"sub1,sub2","verifyHostname":
false,"username":"user1","password":"password","clientId":"my-client"}]'
```

Certificates example:

```
volumes:
  - ./ibsnow-certs:/certs:ro
environment:
  IBSNOW_CERTS: '["/certs/rsa_key.p8","/certs/cert.pem"]'
```

The script verifies that all certificate files exist at their specified paths before attempting upload. If any file is missing, the step is skipped and an error is logged.

Re-applying configuration

Because configuration steps run only once, changing an environment variable and restarting will not re-apply that step. To force a step to run again:

Step 1 — Open a shell into the running container:

```
docker exec -it chariot sh
```

Step 2 — Edit the sentinel file:

```
vi /Chariot/config-state/state.json
```

The file lists every configuration step. Set the `applied` value to `false` for any step you want to re-run. For example, to re-apply MQTT users:

```
"mqttUsers": { "applied": false }
```

Step 3 — Restart the container:

```
docker compose restart chariot
```

The sentinel key names and their corresponding environment variables are:

Sentinel key	Triggered by
`backup`	`BACKUP_FILE`
`adminPassword`	`ADMIN_PASSWORD`
`license`	`LICENSE_TYPE` and related variables
`serverConfig`	`SERVER_CONFIG`
`ibsnowCerts`	`IBSNOW_CERTS`
`ibsnowServers`	`IBSNOW_SERVERS`
`ibsnowConfig`	`IBSNOW_CONFIG`
`mqttUsers`	`MQTT_USERS`
`ssl`	`SETUP_SSL` and `TLS_*` variables
`systemConfig`	`CHARIOT_SERVER_NAME`, `HTTP_PORT`, `HTTPS_PORT`, `HTTPS_ENABLED`

Health check

The image includes a built-in health check that polls the `/eula` REST endpoint every 30 seconds, with a 60-second grace period on startup and a 10-second timeout. After three consecutive failures, Docker marks the container as unhealthy.

To check the current health status:

```
docker inspect --format='{{.State.Health.Status}}' chariot
```

You can also use the same endpoint manually to confirm Chariot is responding:

```
curl -sf http://localhost:8080/eula -H "Accept: application/json;api-version=1.0"
```

A `200` response means Chariot is up. No response or a non-2xx code means it is still starting or has failed.

Troubleshooting

Configuration was not applied

Symptom: Chariot starts, but no users, licenses, or settings were applied. The default `admin`/`password` credentials are still active.

Cause: `ACCEPT_EULA` was not set to `true`. The configure script exits immediately if this check fails, before any other step runs.

Fix: Confirm the variable is set, then remove and recreate the container so the sentinel is fresh:

```
docker rm -f chariot
docker run ... -e ACCEPT_EULA=true ...
```

I changed an environment variable but nothing happened

Cause: Configuration steps are one-shot. Once a step is marked `applied: true` in the sentinel, it will not run again regardless of environment variable changes.

Fix: Reset the relevant step in the sentinel file and restart. See Re-applying configuration.

Can't log in — credentials rejected

Possible causes and fixes:

Default password still active. If `ADMIN_PASSWORD` was not set, the password is `password`.

Backup overrode the password. If `BACKUP_FILE` was used and `ADMIN_PASSWORD` was not set, the password from the backup is now active. Set `ADMIN_PASSWORD`, reset the `adminPassword` sentinel step, and restart.

Admin password step failed. Check the logs for an error on the password step:

```
docker logs chariot | grep -i "admin password"
```

MQTT clients cannot connect

Work through these checks in order:

- Confirm the correct port is exposed: `docker inspect chariot | grep -A 10 Ports`
- If `allowAnonymous` is `false` in `SERVER_CONFIG`, clients must provide credentials. Verify the username and password match an entry in `MQTT_USERS`.
- If you changed a listener port in `SERVER_CONFIG`, ensure the `ports` mapping in your Compose file reflects the updated container-side port.
- For TLS connections (port 8883), verify the client trusts the CA that signed the certificate. Check the SSL step completed: `docker exec chariot cat /Chariot/config-state/state.json | grep ssl`

TLS setup failed or HTTPS is not available

Symptom: Logs show an error on the SSL step. Ports 8443 or 8883 are not responding.

Common causes:

- One or more TLS variables (`TLS_PRIVATE_KEY`, `TLS_CERTIFICATE`, `TLS_CA_CHAIN`) are missing. All three are required when `SETUP_SSL=true`. The script logs: `SETUP_SSL=true requires TLS_PRIVATE_KEY, TLS_CERTIFICATE, and TLS_CA_CHAIN to be set`
- A certificate file path is wrong or the volume is not mounted correctly. The script logs: `TLS file not found: /path/to/file`

Fix: Verify the volume mount and container paths:

```
docker exec chariot ls /certs
```

Confirm the file names match the values in `TLS_PRIVATE_KEY`, `TLS_CERTIFICATE`, and `TLS_CA_CHAIN`, reset the `ssl` sentinel step, and restart.

A configuration step failed but the container is still running

Cause: Configuration steps are independent — a failure in one step does not stop the others. Chariot will continue starting up and log a summary of any steps that failed.

Fix: Check the end of the configuration log for a summary of failed steps:

```
docker logs chariot | grep -A 10 "following steps failed"
```

Each failed step is listed by name. Cross-reference with the sections above to diagnose the specific cause.