Swarm-Reitti Bridge

Overview

Swarm-Reitti Bridge (v1.1.0) is a custom-built Node.js service that automatically syncs Foursquare/Swarm check-ins into two downstream services:

- **Reitti** — self-hosted location history tracker, receiving check-ins in OwnTracks format
- **AdventureLog** — self-hosted travel journal, receiving check-ins as Locations, Visits, and World Travel region marks

It was built specifically for this homelab as a bridge between the Foursquare API and these self-hosted services — keeping all location history under local control.

The service runs at `https://swarm.jeeves5454.ddns.net` and is externally reachable so the OAuth callback from Foursquare can complete.

> **This service was built in this homelab environment.** Its source code lives at `/home/jeeves/docker/code-server/projects/swarm-reitti-bridge/` and the production deployment at `/home/jeeves/docker/swarmreitti/swarm-reitti-bridge/`. The container image (`swarm-reitti-bridge`) is built locally via `docker compose build` and is not published to any registry.

Background: Why Polling Mode?

This bridge was originally designed to use Foursquare **push notifications** — Foursquare would POST to the bridge's `/push` endpoint on every check-in, enabling near-instant syncing. That architecture worked correctly.

In **late 2024 / early 2025**, Foursquare changed their API pricing and moved push notifications behind a paid credit system. Attempting to subscribe to push notifications returns:

```
{ "errorType": "credits_exhausted", "code": 402 }
```

The bridge was updated to **polling mode** as the solution. Instead of waiting for Foursquare to push, it actively polls the Foursquare API every 15 minutes for new check-ins.

Aspect	Push (legacy, broken)	Polling (current)
Trigger	Foursquare → Bridge (instant)	Bridge → Foursquare (timer)
Latency	< 1 second	Up to 15 minutes
API credits required	Yes (paid)	No (free tier)
API calls/day	~0 idle	96 (10% of free quota)
Cost	Requires payment	$0.00

The 15-minute maximum latency is entirely acceptable for personal location history tracking.

How It Works

High-Level Flow

```
User checks in on Swarm
         ↓
Foursquare API stores check-in
         ↓
Bridge polls every 15 minutes (or immediately on container start)
GET /v2/users/self/checkins?oauth_token=...
         ↓
Bridge identifies new check-ins (deduplication via lastCheckinIds)
         ↓
┌──────────────────────────────────────────────────────┐
│  For each new check-in (oldest-first):               │
│                                                      │
│  1. Convert to OwnTracks format                      │
│     POST to Reitti ingest API                        │
│     Authorization: Bearer <REITTI_API_TOKEN>         │
│                    ↓                                 │
│             Check-in on Reitti map                   │
│                                                      │
│  2. (if AdventureLog configured)                     │
│     Find or create AL Category (Foursquare → emoji)  │
```

```
│     Find or create AL Location (dedup by name + coords) │
│     Create AL Visit (dedup: one per location per day)   │
│     Mark AL World Travel region as visited              │
│                     ↓                                   │
│          Location + Visit in AdventureLog               │
│          Region marked on world map                     │
└─────────────────────────────────────────────────────────┘

          ↓
State saved to disk (lastCheckinId, lastCheckinTimestamp)
```

AdventureLog failures are **non-fatal** — Reitti sync completes regardless.

Deduplication

The bridge tracks `lastCheckinIds` per user — a map of `userId → most recently processed check-in ID`. On each poll:

1. **Quick check** — fetches the single most recent check-in from Foursquare. If its ID matches `lastCheckinIds[userId]`, the poll exits immediately (already up to date).
2. **Gap fetch** — if different, fetches all check-ins within a 2-week window (paginated, 50 per page), sorted oldest-first so Reitti receives events in chronological order.

Startup Behaviour

On container start, the bridge immediately polls for any users already stored in `state.json` — it does not wait for the first 15-minute interval. This ensures check-ins that occurred while the container was down are processed within seconds of restart.

State Persistence

Three state objects are saved to `/app/data/state.json`:

Key	Contents
`userTokens`	OAuth access tokens per userId (from Foursquare)
`lastCheckinTimestamps`	Last seen check-in Unix timestamp per userId
`lastCheckinIds`	Last processed check-in ID per userId

Writes are atomic (write to `.tmp`, then rename) to avoid corruption on crash.

> **Permission note:** `state.json` is owned by root (written inside the container). Use `sudo` when editing on the host.

Data Mapping

Reitti (OwnTracks Format)

OwnTracks Field	Source	Notes
`_type`	Hardcoded: `"location"`	OwnTracks type identifier
`lat`	`checkin.venue.location.lat`	Venue latitude
`lon`	`checkin.venue.location.lng`	Venue longitude (`lng` → `lon`)
`tst`	`checkin.createdAt`	Unix timestamp of check-in
`tid`	First 2 chars of username, uppercased	OwnTracks tracker ID
`acc`	Hardcoded: `10`	Accuracy in metres
`alt`	Hardcoded: `0`	Altitude (not available from Foursquare)
`batt`	Hardcoded: `100`	Battery (not applicable)
`vel`	Hardcoded: `0`	Velocity (not applicable)
`t`	Hardcoded: `"c"`	OwnTracks trigger type: check-in
`desc`	`checkin.venue.name` (max 200 chars)	Venue name
`addr`	Address + City + State + Country (max 300 chars)	Formatted address string

```
POST <REITTI_API_URL>
Authorization: Bearer <REITTI_API_TOKEN>
Content-Type: application/json
```

AdventureLog Integration

Category Mapping

- The **primary** Foursquare category maps to an AdventureLog category (created if absent)

- Category icons are resolved from a static Foursquare → emoji lookup table (`adventurelog-categories.js`); unknown categories fall back to 🏷️
- **Secondary** Foursquare categories become AdventureLog `tags`

Location Deduplication

Locations are matched by `venue name (lowercase) + coordinates rounded to 2 decimal places` (≈1.1 km tolerance). If a match exists, it is reused without modification. If not, a new location is created. Coordinates sent to AdventureLog are rounded to 6 decimal places (AdventureLog enforces a 9-total-digit limit on lat/lng fields).

Visit Deduplication

One visit is created per location per local calendar day. The local date is derived using the venue's timezone (resolved from lat/lng via `geo-tz`). A check-in shout and any companion names (`with` array) are combined into the visit notes field.

World Travel (Region Marking)

After each visit is created, the bridge resolves the check-in's country and state/province to an ISO 3166-2 region code via AdventureLog's `GET /api/regions` list, then POSTs to `/api/visitedregion`. Already-visited regions are cached per poll cycle and skipped silently. A small country name map handles known Foursquare → AdventureLog naming mismatches (e.g. `México` → `Mexico`). Unresolvable regions are logged as warnings.

> **Note:** Creating Locations and Visits via the API does **not** automatically update the World Travel world map — these are separate systems in AdventureLog. The bridge handles both independently.

OAuth Setup

Foursquare OAuth 2.0 is used to grant the bridge permission to read check-in data. This is a one-time setup that persists in the state file.

Initial Setup Steps

1. **Visit the auth URL:**

```
https://swarm.jeeves5454.ddns.net/auth
```

2. **The bridge redirects to Foursquare** with the configured `client_id` and `redirect_uri`.
3. **Authenticate** with your Foursquare account and authorise the app.
4. **Foursquare redirects back** to `https://swarm.jeeves5454.ddns.net/callback` with an authorisation code.
5. **The bridge exchanges** the code for an access token, saves it to state, and begins polling immediately.

> **Important:** Re-authenticating via `/auth` resets the sync baseline to the most recent check-in at that moment — older unsynced check-ins will be skipped. Only re-authenticate if starting fresh or after a token expiry. If you need to reprocess past check-ins, patch `state.json` directly (see Operations below).

Foursquare App Configuration

Field	Value
Redirect URI	`https://swarm.jeeves5454.ddns.net/callback`
Push API URL	(not required — polling mode)

Access and Endpoints

External URL: `https://swarm.jeeves5454.ddns.net` **Certificate:** Let's Encrypt (letsencrypt resolver)
Port: 3000 (internal)

API Endpoints

Method	Path	Purpose
`GET`	`/`	Status dashboard — service cards, recent check-ins, recent errors
`GET`	`/auth`	Start Foursquare OAuth flow (rate-limited: 10/15 min)
`GET`	`/callback`	OAuth callback (Foursquare redirects here)
`GET`	`/health`	JSON health — `ok`, per-service status, last sync times
`POST`	`/push`	Legacy webhook endpoint (inactive in polling mode)

The `/health` endpoint returns structured JSON including an `ok: true/false` field suitable for Uptime Kuma's JSON Query monitor:

```
{
  "ok": true,
  "status": "healthy",
  "connectedUsers": 1,
  "reitti": { "configured": true, "lastSuccessAt": "...", "lastSuccessVenue": "..." },
  "adventurelog": { "configured": true, "lastSuccessAt": "...", "lastSuccessVenue": "..." },
  "lastPollAt": "..."
}
```

Configuration

Environment Variables

Variable	Value / Notes
`FOURSQUARE_CLIENT_ID`	**REDACTED** (Foursquare developer console)
`FOURSQUARE_CLIENT_SECRET`	**REDACTED**
`FOURSQUARE_REDIRECT_URI`	`https://swarm.jeeves5454.ddns.net/callback`
`REITTI_API_URL`	`https://reitti.jeeves5454.ddns.net/api/v1/ingest/owntracks`
`REITTI_API_TOKEN`	**REDACTED** (Reitti API bearer token)
`PUSH_SECRET`	**REDACTED** (session signing secret)
`ADVENTURELOG_API_URL`	`https://travel.jeeves5454.ddns.net`
`ADVENTURELOG_API_KEY`	**REDACTED** (AdventureLog API key)
`UPTIME_KUMA_PUSH_URL`	**REDACTED** (Uptime Kuma push monitor URL, no query params)
`POLLING_INTERVAL_MINUTES`	`15` (default)
`PORT`	`3000`
`NODE_ENV`	`production`

Omitting both `ADVENTURELOG_API_URL` and `ADVENTURELOG_API_KEY` disables AdventureLog sync entirely. Omitting `UPTIME_KUMA_PUSH_URL` disables push heartbeats.

Traefik Labels

```
traefik.http.routers.swarm-bridge.rule: Host(`swarm.jeeves5454.ddns.net`)
traefik.http.routers.swarm-bridge.entrypoints: websecure
traefik.http.routers.swarm-bridge.tls.certresolver: letsencrypt
traefik.http.services.swarm-bridge.loadbalancer.server.port: 3000
```

Volumes / Bind Mounts

Host Path	Container Path	Purpose
`/home/jeeves/docker/swarmreitti/swarm-reitti-bridge/data`	`/app/data`	State file (`state.json`) persisted across restarts

Monitoring (Uptime Kuma)

Three complementary monitors are recommended:

Monitor Type	Target	What it catches
Docker Container	Container name `swarm-reitti-bridge`	Container crash / OOM kill
HTTP(s)	`https://swarm.jeeves5454.ddns.net/health`	App unresponsive
HTTP(s) JSON Query	`https://swarm.jeeves5454.ddns.net/health` → `$.ok`	Reitti/AL sync errors, no users
Push	Uptime Kuma push monitor URL	Poll loop stopped, token expired

The push monitor receives `?status=up` after every successful poll and `?status=down&msg=<reason>` on failure. Configure the heartbeat interval to `POLLING_INTERVAL_MINUTES + 2` minutes (e.g. 17 minutes for a 15-minute poll interval).

Logging

The bridge uses structured JSON logging to stdout (Dozzle-compatible):

```
{"ts":"2026-06-23T12:00:00.000Z","level":"INFO","cat":"POLL","msg":"Scheduled
poll","userCount":1}
{"ts":"2026-06-23T12:00:01.000Z","level":"INFO","cat":"CHECKIN","msg":"Sent to
Reitti","venue":"Tim Hortons"}
{"ts":"2026-06-23T12:00:02.000Z","level":"INFO","cat":"AL","msg":"Location
created","venue":"Tim Hortons","id":"..."}
{"ts":"2026-06-23T12:00:02.000Z","level":"INFO","cat":"AL","msg":"Visit
created","locationId":"...","date":"2026-06-23"}
{"ts":"2026-06-23T12:00:02.000Z","level":"INFO","cat":"AL","msg":"Region marked as
visited","regionId":"CA-ON"}
{"ts":"2026-06-23T12:00:03.000Z","level":"INFO","cat":"STATE","msg":"State
saved","usersCount":1}
```

Log categories: `SERVER`, `POLL`, `CHECKIN`, `AL`, `AUTH`, `STATE`.

`WARN` and `ERROR` entries are also captured in an in-memory ring buffer (last 20) and displayed on the status dashboard at `/`.

View live logs:

```
docker logs -f swarm-reitti-bridge
```

Operations and Management

Verify the Bridge is Running

```
# Container status
docker ps | grep swarm-reitti-bridge

# Health check (includes ok field and per-service status)
curl -s https://swarm.jeeves5454.ddns.net/health | jq .

# Status dashboard
open https://swarm.jeeves5454.ddns.net/
```

Re-authenticate with Foursquare

If the OAuth token expires or becomes invalid (bridge logs show `Token expired` WARN):

1. Visit `https://swarm.jeeves5454.ddns.net/auth`
2. Log in and authorise the app
3. The bridge saves the new token and polls immediately

> Do **not** re-authenticate to retry past check-ins — it will move the sync baseline forward and those check-ins will be permanently skipped.

Reprocess Past Check-ins

To force the bridge to reprocess a specific check-in (e.g. after fixing an AL sync error):

```
cd /home/jeeves/docker/swarmreitti/swarm-reitti-bridge

# View current state
sudo cat data/state.json

# Roll back to just before a specific check-in Unix timestamp
sudo python3 -c "
import json
with open('data/state.json') as f:
    s = json.load(f)
s['lastCheckinIds']['<userId>'] = None
s['lastCheckinTimestamps']['<userId>'] = <tst - 1>
with open('data/state.json', 'w') as f:
    json.dump(s, f, indent=2)
print('Done')
"

docker compose restart swarm-reitti-bridge
```

The startup poll will immediately pick up the check-in on restart.

Adjust Polling Interval

Edit `POLLING_INTERVAL_MINUTES` in the compose environment and rebuild:

Interval	API calls/day	Use case
`5`	288	More responsive (30% quota)
`15`	96	**Recommended** (10% quota)
`30`	48	Most conservative (5% quota)

Rebuild the Container

```
cd /home/jeeves/docker/swarmreitti/swarm-reitti-bridge
docker compose up -d --build
docker logs swarm-reitti-bridge --follow
```

Troubleshooting

Check-ins not appearing in Reitti

1. Check the status dashboard at `/` — the Reitti card shows last push time and any error message
2. Check logs:
   ```
   docker logs swarm-reitti-bridge | grep -E '"cat":"CHECKIN"|"level":"ERROR"'
   ```
3. Verify token is connected: `curl -s https://swarm.jeeves5454.ddns.net/health | jq .connectedUsers` — if `0`, re-authenticate via `/auth`
4. Test Reitti API directly:
   ```
   curl -s -o /dev/null -w "%{http_code}" -X POST \
     https://reitti.jeeves5454.ddns.net/api/v1/ingest/owntracks \
     -H "Authorization: Bearer <token>" \
     -H "Content-Type: application/json" \
     -d '{"_type":"location","lat":43.65,"lon":-79.37,"tst":1782088106}'
   ```
   Expected: `200`. If `401`, the token in `.env` is wrong.

Check-ins not appearing in AdventureLog

1. Check the status dashboard `/` — the AdventureLog card shows last sync and any error

2. Verify AdventureLog API key is valid:

   ```
   curl -s "https://travel.jeeves5454.ddns.net/api/locations?limit=1" \
     -H "X-API-Key: <key>"
   ```

   Expected: JSON with `count` field. If `403`, the key is wrong.
3. Common error — **trailing slash causes 308 redirect with lost body**: the bridge strips trailing slashes automatically; if seeing 400s verify the AL URL has no trailing slash in `ADVENTURELOG_API_URL`

World Travel region not updating

- The bridge only marks regions when `venue.location.state` and `venue.location.country` are both present in the Foursquare data. Venues with missing location data are skipped.
- If a region resolves but the country/state name doesn't match AdventureLog's English names, a `WARN` log is emitted: `Could not resolve region`. Add a mapping to `FOURSQUARE_COUNTRY_MAP` in `index.js` if needed.
- `GET /api/visitedregion` can confirm which regions are already marked.

OAuth callback fails (redirect URI mismatch)

- Ensure `FOURSQUARE_REDIRECT_URI` in env exactly matches the redirect URI registered in the Foursquare developer console: `https://swarm.jeeves5454.ddns.net/callback`

Bridge container exits immediately

- Check logs: `docker logs swarm-reitti-bridge`
- Most common cause: missing required environment variable. The app logs all missing vars and exits with code 1.

State file corruption

```
sudo rm /home/jeeves/docker/swarmreitti/swarm-reitti-bridge/data/state.json
docker restart swarm-reitti-bridge
# Then re-authenticate via /auth
```

Source Code

```
/home/jeeves/docker/code-server/projects/swarm-reitti-bridge/   ← development
/home/jeeves/docker/swarmreitti/swarm-reitti-bridge/            ← production deployment
├── index.js                    ← Main application
├── adventurelog-categories.js  ← Foursquare category → emoji icon map
├── Dockerfile
├── docker-compose.yml
├── package.json                ← version 1.1.0
├── tests/
│   ├── integration.test.js
│   └── unit.test.js
└── data/
    └── state.json              ← Runtime state (OAuth tokens, last check-in IDs)
```

Last Updated: 2026-06-23

Revision #3
Created 2026-06-17 04:45:57 UTC by Admin
Updated 2026-06-24 05:34:15 UTC by Admin