Enterprise Deployment Guide

Elis AI Technical Documentation

Generated: 2026-05-19 12:44 UTC

Repository: Deploy

Deployment Notes

Updated March 25, 2026.

This document intentionally excludes live secrets. Store production credentials in Azure Container Apps secrets, GitLab CI variables, or a dedicated secret manager instead of committing them to source control.

Immediate Follow-Up

- Rotate any credentials that were previously written into this file.
- Confirm the rotated values are updated in every deployed environment before the next release.

Pending Migration — token_transactions (March 19, 2026)

The `token_transactions` table had a stale `CHECK` constraint (`token_transactions_tx_type_check`) that only allowed: `free_daily`, `miner_earned`, `package_purchase`, `api_spend`, `chat_spend`, `admin_grant`, `refund`. The code now also uses `subscription_grant` and `stripe_topup`, which were blocked by this constraint.

This is now handled by the auto-discovery migration system (see below). No manual SQL is required — just deploy the new backend image and restart.

Token ledger / chat settlement (March 25, 2026)

Chat billing settlement writes `miner_earned` rows that must satisfy the same `token_transactions` columns as `chat_spend` (including `balance_after` when that column exists). Migration `0014_token_transactions_ledger_columns` ensures:

- `balance_after` and `reference_id` exist when an older database predates them.
- `balance_after` is **backfilled** as a running sum of `amount` per `user_id` (ordered by `created_at`, `id`) **only where it was NULL**, so existing rows are not overwritten.
- `reference_id` is filled from `reference_key` where `reference_id` is still null.
- `NOT NULL` is applied to `balance_after` only after every row has a value (avoids breaking partially migrated databases).

Data on Azure is preserved (no `DELETE` / `TRUNCATE`). Deploy the backend image and restart; migrations run automatically. If `0014` exits early because some rows still have `balance_after IS NULL`, fix data manually or re-run after cleanup — the app still works when `balance_after` is absent or nullable, but production schemas that require `NOT NULL` should complete the backfill.

Database Migrations

The project uses a lightweight auto-discovery migration runner in `backend/migrations/`. Migrations run automatically on backend startup via `init_schema()` and are tracked in the `schema_migrations` table so each one executes at most once.

How it works

1. Place a numbered Python file in `backend/migrations/` (e.g. `0002_add_foo_column.py`).
2. The file must expose a `run(cur)` function that receives an open psycopg2 cursor.
3. On startup, the runner sorts all `[0-9]*.py` files, skips already-applied ones, and executes the rest in order.
4. Applied migrations are recorded in `schema_migrations(name, applied_at)`.

Adding a new migration

```
# backend/migrations/0002_describe_the_change.py
"""Brief description of what this migration does."""

def run(cur) -> None:
    cur.execute("ALTER TABLE ... ")
```

Current migrations

Name	Description
`backfill_trace_json`	(legacy) Convert TracePacket repr strings to JSON
`0001_fix_token_transactions_constraint`	Drop stale tx_type CHECK, ensure reference_key + metadata_json columns
`0014_token_transactions_ledger_columns`	Add/backfill `balance_after` + `reference_id` on `token_transactions` (chat settlement)
`0036_automation_chunks`	Create `automation_chunks` pgvector table with HNSW index for widget report vector storage
`0037` – `0044`	Crawl/pagination policy, embedding dimensions, user ML system (see `backend/migrations/`)

Note (WS2-WS5): No new migrations added. Tier gating, shared pool, and token tracking use existing tables with metadata-level changes only (`metadata_json` fields on `automation_chunks`, new

`tx_type` values in `token_transactions`).

Azure deployment notes

- Migrations auto-apply on restart — push the new image and restart the container app.
- To verify: `SELECT * FROM schema_migrations ORDER BY applied_at;`
- If the backend cannot be restarted, run the migration SQL manually against the Azure PostgreSQL instance. The SQL for each migration is in its file under `backend/migrations/`.
- **Template for env vars** (no secrets committed): see `config/deployment.env.template`.

On-prem bundle publication

The downloadable on-prem bundle is now built and published separately from the backend image.

- Build the archive only: `scripts/build_onprem_bundle.ps1`
- Build, upload to Azure Blob Storage, and update backend env vars: `scripts/publish_onprem_bundle.ps1`
- `deploy.ps1 -Target backend` will publish the on-prem bundle automatically when `ONPREM_BUNDLE_BLOB_CONNECTION_STRING` is set in the shell and `-SkipOnPremBundlePublish` is not used.

Required env var for publishing:

- `ONPREM_BUNDLE_BLOB_CONNECTION_STRING`

Required env vars for hosted issuance of signed on-prem activation codes:

- `LICENSE_PUBLIC_KEY_PEM`
- `LICENSE_SIGNING_PRIVATE_KEY_PEM`

Optional env vars:

- `ONPREM_BUNDLE_BLOB_CONTAINER`
- `ONPREM_BUNDLE_BLOB_NAME`
- `ONPREM_BUNDLE_VERSION`
- `ONPREM_BUNDLE_CHECKSUM_SHA256`
- `ONPREM_BUNDLE_PUBLISHED_AT`

On-prem bootstrap env vars are separate from hosted Azure deployment and should not be set on the shared production apps:

- `ONSITE_BOOTSTRAP_TOKEN`
- `ONSITE`
- `NEXT_PUBLIC_ONSITE`

These are bundle/runtime settings for customer-managed onsite installs only. They are not required for `deploy.ps1` Azure Container Apps deploys and should remain unset on the hosted backend/frontend unless you are intentionally deploying a dedicated onsite-style environment.

If `LICENSE_PUBLIC_KEY_PEM` and `LICENSE_SIGNING_PRIVATE_KEY_PEM` are missing on the hosted backend, on-prem checkout stays degraded by design:

- readiness exposes `onprem_license_signing_not_configured`
- hosted billing shows on-prem checkout as not ready
- activation payload issuance returns `pending_signing_key` until the signing keys are configured

Production env parity check (April 7, 2026)

Validated via Azure CLI (`az containerapp show`).

- Resource group: `amodelis-prod`
- App: `backend`
- Revision: `backend--0000171`
- Image: `[private container registry]/backend:v1.0.88`

All concurrency-related env vars confirmed present:

Variable	Value on Azure	Status
`GUNICORN_WORKERS`	`2`	✅ Correct for 2 vCPU
`GUNICORN_WORKER_CONNECTIONS`	`2000`	✅
`GUNICORN_TIMEOUT`	`600`	✅
`ORCH_THREAD_POOL_WORKERS`	`8`	✅
`ORCH_STREAM_RUN_TIMEOUT_SECONDS`	`300`	✅
`ORCH_STATUS_POLLING_TIMEOUT_GRACE_S`	`120`	✅
`ORCH_TOOL_LOOP_MAX_LATENCY_MS`	`300000`	✅
`ORCH_MAX_RESPONSE_TIME_SECONDS`	`300`	✅
`ORCH_MINER_PROXY_TIMEOUT_SECONDS`	`240`	✅
`ORCH_IGNORE_USER_PER_REQUEST_TOKEN_LIMIT`	`1`	✅
`MINER_HTTP_SYNC_WAIT_S`	`90`	✅
`MINER_QUEUE_SYNC_WAIT_S`	`60`	✅
`MINER_INTERNAL_MODE`	`1`	✅
`DB_POOL_MIN`	`3`	✅
`DB_POOL_MAX`	`12`	✅ Sized for Postgres B1ms
`DAILY_FREE_TOKEN_AMOUNT`	`500000`	✅
`LOG_LEVEL`	`ERROR`	✅
`ORCH_STREAM_HARD_TIMEOUT_ENABLED`	`1`	✅

Variable	Value on Azure	Status
`BROKER_ENABLED`	`1`	✅ Broker (narrator) framework owns the user-facing stream and per-conversation task queue
`BROKER_RUNNING_SLOT_CAP`	`3`	✅ Max concurrent tasks per conversation before broker prompts user to reorder/cancel

Verify current env vars:

```
az containerapp show --name backend --resource-group amodelis-prod \
  --query "properties.template.containers[0].env[].{name: name, value: value}" -o table
```

Automation Vector Store (April 7, 2026)

Widget report data is now chunked, embedded, and stored in a pgvector table (`automation_chunks`) for semantic recall in chat and cross-widget pipelines.

Activation

The feature is **off by default**. Set this env var on the backend to enable:

Variable	Value	Purpose
`AUTOMATION_VECTORSTORE_ENABLED`	`1`	Enable ingestion of widget report data into `automation_chunks` and recall in chat/pipelines. Default `0` (disabled).

What it does when enabled

1. **Ingestion**: After every successful widget run, report findings, metrics, source evidence, and table rows are chunked (512 tokens, 64 overlap), embedded via the existing `EmbeddingService` , and bulk-inserted into `automation_chunks` .
2. **Chat recall**: `recall_artifacts_for_prompt()` searches `automation_chunks` by cosine similarity (0.10 boost, below company docs' 0.15). Each recalled chunk gets a freshness tag (`[FRESH]` , `[RECENT]` , `[AGING]` , `[STALE]`) based on `collected_at` .
3. **Cross-widget recall**: Downstream widgets with `upstream_widget_id` or `source_config.recall_from_widgets` automatically receive upstream data in their prompt, gated by `max_upstream_age_hours` (default: `stale_after_hours` or 48h).

4. **Retention**: `delete_expired_widget_reports()` cascade-deletes associated chunks. Per-widget, only the latest 2 runs' chunks are kept to prevent index bloat.

Migration

Migration `0036_automation_chunks` runs automatically on startup. It creates:
- `automation_chunks` table with `embedding vector(1536)`, metadata columns (`widget_id`, `run_id`, `report_id`, `company_id`, `chunk_type`, `source_urls`, `entity_tags`, `metric_tags`, `metric_value`, `collected_at`)
- HNSW index on `embedding` for cosine distance search
- 7 supplementary indexes for filtering by company, widget, report, run, collected_at, and compound (company_id, chunk_type)

Verify post-deploy:

```
SELECT * FROM schema_migrations WHERE name = '0036_automation_chunks';
SELECT count(*) FROM automation_chunks;  -- should be 0 until first widget runs complete
```

Resource impact

- **Embedding cost**: ~$0.04/day at embedding API rates for 500 daily widget runs × 20 chunks.
- **Storage**: ~1 KB per chunk (content + vector embedding). 10K chunks ≈ 10 MB. Negligible on current production Postgres storage.
- **Latency**: Ingestion is async (runs after report creation, does not block the widget run response). Recall adds one pgvector query (~5-15ms with HNSW index) to the existing recall pipeline.

Widget configuration (optional, for cross-widget recall)

Widget field	Type	Purpose
`upstream_widget_id`	`string`	Widget whose latest report data is injected into this widget's prompt
`source_config.recall_from_widgets`	`string[]`	Additional widget IDs to recall data from
`source_config.max_upstream_age_hours`	`int`	Max age of upstream chunks to use (default: `stale_after_hours` or 48)

Rollback

To disable without data loss, unset `AUTOMATION_VECTORSTORE_ENABLED` (or set to `0`). Ingestion and recall are fully gated — existing chunks remain in the table but are not queried. To remove the table

entirely:

```
DROP TABLE IF EXISTS automation_chunks;
DELETE FROM schema_migrations WHERE name = '0036_automation_chunks';
```

Tier Gating, Shared Data Pool & Token Tracking (April 9, 2026)

This release adds plan-based feature gating, cross-company data sharing, and token usage tracking. **No new database migrations are required** — all features use existing tables (`automation_chunks` , `token_transactions` , `company_subscriptions`) with metadata-level changes only.

What changed

Area	Backend files	Frontend files
Tier gating	`agents/plan_gate.py` (new), `billing.py` , `automations_api.py` , `agents/automations_service.py`	`components/automations/widget-editor.tsx` , `(app)/billing/page.tsx` , `lib/types.ts`
Shared data pool	`agents/shared_pool.py` (new), `agents/collection_store.py` , `agents/db.py` , `agents/adaptive_persistence.py`	—
Token tracking	`agents/orchestration.py` , `agents/embeddings.py` , `agents/automations_service.py` , `company_api.py` , `agents/db.py`	`(app)/companies/[id]/page.tsx` , `lib/company-api.ts`
Content pages	—	`app/pricing/page.tsx` (new), `app/tutorials/user/data-collection/page.tsx` (new), `app/page.tsx` , `app/docs/page.tsx`
Real-time extraction	`agents/collection_extract.py` (new), `agents/tools/tool_calling_service.py`	—

Tier gating details

Plan features are defined in `billing.py` under each plan's `features` dict and `FREE_TIER_FEATURES` . The `plan_gate.py` service resolves a company's active plan and returns feature limits used at five enforcement points:

1. **Widget creation limit** — `automations_api.py` checks `max_widgets`
2. **Deep collection access** — `automations_service.py` checks `deep_collection`
3. **Widget visibility** — Public widgets require `shared_pool_access`
4. **Crawl page limit** — Page budget per widget run gated by plan tier
5. **Billing page** — Frontend shows plan-specific feature bullets

No env vars required. Gating reads directly from `company_subscriptions` + `billing.py` plan definitions.

Shared data pool details

Widgets marked `visibility=public` with `source_class=public_web` contribute chunks to the shared pool. Other companies can recall these chunks during chat via `shared_pool.py` → `search_shared_pool_chunks()` in `db.py`. Chunks from the requester's own company are excluded. PII fields (`source_urls`, `entity_tags`) are redacted before delivery.

- Daily query limit: 50 per company (configurable in `shared_pool.py`)
- Contributor credits: `shared_data_credit` tx_type in `token_transactions`
- No new tables — queries the existing `automation_chunks` table filtering on `metadata_json` fields

Token tracking details

Three new `tx_type` values in `token_transactions`:
- `automation_spend` — logged after each automation widget run
- `embedding_spend` — logged per embedding batch
- `shared_data_credit` — credited to data contributors

The `0001` migration already dropped the `tx_type` CHECK constraint, so these values require no schema changes.

New API endpoints (role-gated to `dev+`):
- `GET /api/companies/<id>/usage?days=30` — aggregate token usage summary
- `GET /api/companies/<id>/usage/members?days=30` — per-member breakdown

Verify post-deploy

```
-- Confirm tx_type CHECK is dropped (should return 0 rows)
SELECT conname FROM pg_constraint
WHERE conrelid = 'token_transactions'::regclass AND contype = 'c';

-- Confirm automation_chunks has metadata fields
```

```
SELECT DISTINCT metadata_json->>'source_class', metadata_json->>'widget_visibility'
FROM automation_chunks LIMIT 10;
```

Rollback

All WS2-WS5 features are code-only (no schema migrations). To roll back, deploy the previous backend + frontend image tags. No database changes to revert.

Production Endpoints

- **Frontend**: `https://tryelisai.com` (also `https://www.tryelisai.com`)
- **Backend API**: `https://api.tryelisai.com`

Legacy Azure URLs (redirect to custom domain):
- Backend: `https://backend.calmocean-1927d5f8.eastus.azurecontainerapps.io`
- Frontend: `https://frontend.calmocean-1927d5f8.eastus.azurecontainerapps.io` (301 → tryelisai.com)

Custom Domain

- **Domain**: `tryelisai.com` (registered via Azure App Service Domains)
- **DNS Zone**: Azure DNS in resource group `amodelis-prod`
- **Container Apps Environment**: `amodelis-env`
- **Environment Static IP**: `52.188.74.22`

DNS Records

Type	Name	Value
A	@	52.188.74.22
CNAME	www	frontend.calmocean-1927d5f8.eastus.azurecontainerapps.io
CNAME	api	backend.calmocean-1927d5f8.eastus.azurecontainerapps.io
TXT	asuid	7722DCC994F5A7F153C7DC7BB498B1F3F8A06136A614C6E571EF4DAE049EAAD2
TXT	asuid.www	(same verification ID)
TXT	asuid.api	(same verification ID)

Hostname Bindings

Hostname	Container App	Certificate
tryelisai.com	frontend	Managed (auto-renewed)
www.tryelisai.com	frontend	Managed (auto-renewed)
api.tryelisai.com	backend	Managed (auto-renewed)

The old Azure frontend URL is redirected to `https://tryelisai.com` via Next.js middleware (`frontend/middleware.ts`).

Required Runtime Secrets

Backend:

- `DATABASE_URL`
- `REDIS_URL`
- `JWT_SECRET`
- `OPENAI_API_KEY`
- `ANTHROPIC_API_KEY` — Claude models (sk-ant-…)
- `GOOGLE_API_KEY` — Gemini models
- `GROQ_API_KEY` — Groq-hosted Llama / Mixtral / DeepSeek (gsk_…)

- `MISTRAL_API_KEY` — Mistral cloud models
- `CEREBRAS_API_KEY` — Cerebras Wafer-Scale models (csk-…)
- `STRIPE_SECRET_KEY`
- `STRIPE_PUBLISHABLE_KEY`
- `APP_BASE_URL`
- `FRONTEND_BASE_URL`
- `DAILY_FREE_TOKEN_AMOUNT` (use `500000` for production; `1000` **is a common typo** and would show as ~1.0K in the UI — the backend now rejects sub-10K values unless `ALLOW_LOW_DAILY_FREE_TOKENS=1`)
- `AUTH_REQUIRE_EMAIL_VERIFICATION`
- `AUTH_REQUIRE_EMAIL_2FA`
- `AZURE_COMMUNICATION_SERVICES_CONNECTION_STRING`
- `AZURE_COMMUNICATION_EMAIL_SENDER`
- `LICENSE_PUBLIC_KEY_PEM` when issuing signed on-prem activation codes
- `LICENSE_SIGNING_PRIVATE_KEY_PEM` when issuing signed on-prem activation codes

Frontend build/runtime:

- `INTERNAL_API_URL`
- `NEXT_PUBLIC_API_URL`

Backend environment variables (miners, CORS, browser service)

Set these on the **backend** Container App (or in CI) when you use external miners, browser-based fetching, or custom domains:

Variable	Purpose
`CORS_ALLOWED_ORIGINS`	Comma-separated browser origins allowed to call the API. **Production:** include `https://tryelisai.com`, `https://www.tryelisai.com`, and any staging origins.
`MINER_INTERNAL_MODE`	`1` / `true` = backend uses built-in virtual miners (OpenAI keys on the backend). `0` / `false` = dispatch jobs to registered external miners. **Production with community miners:** use `0`.
`MINER_INTERNAL_OWNER_USER_ID`	Optional explicit user id to attribute internal-miner earnings when `MINER_INTERNAL_MODE` is on.
`MINER_INTERNAL_OWNER_EMAIL`	Optional email lookup for the same (if user id not set).
`ROUTER_ALLOW_EXPLORATION_IN_PROD`	Set to `1` to allow active router policies with `exploration.enabled=true` in production (`APP_ENV=prod`).
`BROWSER_SERVICE_URL`	Base URL of the headless browser microservice (e.g. `http://browser:9020` in compose). **Unset** = browser fetch features disabled.
`BROWSER_SERVICE_AUTH_TOKEN`	Shared secret; backend sends `Authorization: Bearer …` to the browser service. Must match the token configured on the browser Container App.
`BROWSER_SERVICE_POOL_SIZE`	Concurrent sessions to pool (default `6` in `docker-compose.yml`). Must not exceed `BROWSER_SESSION_MAX_COUNT` on the browser service (default `12`).
`BROWSER_SESSION_MAX_COUNT`	Server-side ceiling for open Chrome sessions on the browser container (default `12` in `browser_service/.env`). Keep above `BROWSER_SERVICE_POOL_SIZE`.
`ORCH_WEB_FETCH_PARALLELISM`	Concurrent browser fetch calls per orchestration run (default `6`). Keep in sync with `BROWSER_SERVICE_POOL_SIZE`.
`BROWSER_SERVICE_POOL_WAIT_SECONDS`	Max wait for a free browser (default `5`).

Variable	Purpose
`BROWSER_SERVICE_FETCH_WAIT_SECONDS`	Post-navigation settle time for `/fetch` (default `0.4`).
`BROKER_ENABLED`	`1` (default in `deploy.ps1` and `docker-compose.yml`) routes every chat turn through the broker (narrator), which owns the user-facing stream, dispatches background `Task` records via `task_registry`, and emits `broker_question` chunks when the conversation queue cap is reached. Set `0` to fall back to the legacy `run()` pipeline path.
`BROKER_RUNNING_SLOT_CAP`	Max concurrent running tasks per conversation before the broker prompts the user with reorder/cancel choices. Default `3`.
`BROKER_TASK_BACKEND`	Optional. `memory` forces the in-process `TaskRegistry` even when `REDIS_URL` is set (useful for isolated debugging). Leave unset in production so the Redis-backed registry is used and task state survives across Gunicorn workers.
`BROKER_TASK_TTL_SECONDS`	TTL for per-conversation task lists in Redis. Default `86400` (24 h).

> **Note on deploy automation:** `deploy.ps1` now sets `BROKER_ENABLED` and `BROKER_RUNNING_SLOT_CAP` on the backend Container App on every deploy. Override by exporting the env var in the deploy shell before running the script (e.g. `$env:BROKER_ENABLED = "0"`).

Model provider API keys

`deploy.ps1` also pushes the following provider keys to the backend Container App on every backend deploy, but **only when they are present in the deploy shell** (it never blanks an existing Azure value):

Variable	Provider	Format
`ANTHROPIC_API_KEY`	Anthropic Claude	`sk-ant-…`
`GOOGLE_API_KEY`	Google Gemini	(Google API key)
`GROQ_API_KEY`	Groq (Llama / Mixtral / DeepSeek)	`gsk_…`
`MISTRAL_API_KEY`	Mistral cloud	(Mistral key)
`CEREBRAS_API_KEY`	Cerebras Wafer-Scale	`csk-…`

These act as the platform BYOK fallback used by `backend/agents/model_clients.py` (ModelGateway) and `backend/internal_miner.py` (auto-registration of internal miners) when an organization has not configured its own keys via the BYOK panel.

To deploy with all keys, export them before running `deploy.ps1`:

```
$env:ANTHROPIC_API_KEY = "sk-ant-…"
$env:GOOGLE_API_KEY    = "AIza…"
$env:GROQ_API_KEY      = "gsk_…"
$env:MISTRAL_API_KEY   = "…"
$env:CEREBRAS_API_KEY  = "csk-…"
.\deploy.ps1 -Target backend
```

On-prem only variables

Do not configure these on the shared Azure production apps unless you are intentionally running an onsite deployment profile:

Variable	Purpose
`ONSITE`	Enables onsite runtime posture and related guards/flows. Leave unset or `0` on hosted Azure.
`NEXT_PUBLIC_ONSITE`	Frontend onsite mode flag for downloadable bundle/runtime. Not used for hosted Azure frontend builds.
`ONSITE_BOOTSTRAP_TOKEN`	One-time first-admin bootstrap token for onsite bundles. Never required for hosted Azure deploys.

The browser service container should set **`BROWSER_SERVICE_ALLOWED_INTERNAL_HOSTS`** if you navigate to private hostnames (defaults include `frontend`, `localhost`, `host.docker.internal`).

Registry / CI:

- `GITLAB_TOKEN`

Azure Email Auth

The app now supports Azure Communication Services Email for both account verification and emailed second-factor codes.

Set these backend secrets to enable it:

- `AZURE_COMMUNICATION_SERVICES_CONNECTION_STRING`
- `AZURE_COMMUNICATION_EMAIL_SENDER`
- `FRONTEND_BASE_URL`

Set these backend flags to enforce it:

- `AUTH_REQUIRE_EMAIL_VERIFICATION=1`
- `AUTH_REQUIRE_EMAIL_2FA=1`

Behavior:

- Signup creates the user and sends a verification link to `/verify-email`.
- Login checks email verification first, then sends a one-time sign-in code by email before issuing the JWT.
- When the flags are unset, the previous email/password flow stays active.

Compliance Environment Variables (April 11, 2026)

The `ComplianceMonitor` service runs every 6 hours and evaluates runtime posture against HIPAA, ITAR, NERC CIP, FISMA, and PCI-DSS requirements. Alerts are surfaced on the system-admin compliance dashboard and per-org compliance tabs. To pass all compliance checks on Azure production, set the following env vars on the **backend** Container App.

Required for all deployments

Variable	Value	Framework	Purpose
`JWT_EXPIRY_HOURS`	`24`	HIPAA	Session token lifetime must be ≤ 24 hours. Default is 72 — triggers a **high** session_timeout alert.
`AUTH_REQUIRE_EMAIL_2FA`	`1`	HIPAA	Enforce email-based MFA on login. Without this, a **medium** mfa alert fires.
`AUTH_IDLE_TIMEOUT_MINUTES`	`15`	HIPAA	Auto-logout after idle period. Must be > 0 and ≤ 15. Missing or > 15 triggers a **medium** idle_timeout alert.
`DB_ENCRYPTION_KEY`	*(base64 Fernet key)*	HIPAA	Encryption key for PHI fields at rest. Missing triggers a **critica

[...content truncated for whitepaper synthesis...]

Repository: Deploy Dev

Dev Azure Deployment Notes

Scope

This document describes the isolated **dev** deployment in Azure. It is separate from production and safe for testing latest code.

Dev Architecture

- **Resource group**: `amodelis-dev`
- **Container Apps environment**: `amodelis-dev-env` (East US)
- **Backend app**: `backend-dev` (external ingress, port 8000)
- **Frontend app**: `frontend-dev` (external ingress, port 3000)
- **Browser service**: `browser-dev` (internal ingress, port 9020)
- **Postgres**: Azure Database for PostgreSQL Flexible Server `elis-dev-pg` (East US 2)
- **Redis**: Azure Cache for Redis `elis-dev-redis` (East US)
- **Registry**: `[private container registry]`

Why this topology

- Container Apps TCP ingress is not a reliable fit for Postgres in this project.
- Managed Postgres + managed Redis gives stable connectivity and simpler operations.
- Dev and prod are isolated by resource group, app names, and managed services.

Environment variables

Set automatically via Azure CLI (`deploy-dev.ps1`)

The script runs `az containerapp create` / `az containerapp update` and merges env vars (existing keys not listed are left in place).

Backend (`backend-dev`) — every deploy (existing app)

- `REDIS_URL` — from Azure Redis (`rediss://…:6380/0`).
- **Concurrency / pools** (defaults in script; change `$DevGunicornWorkers` , `$DevGunicornWorkerConnections` , `$DevGunicornTimeout` , `$DevOrchThreadPoolWorkers` , `$DevDbPoolMin` , `$DevDbPoolMax` near the top of `deploy-dev.ps1`):

- `GUNICORN_WORKERS`
- `GUNICORN_WORKER_CONNECTIONS`
- `GUNICORN_TIMEOUT`
- `ORCH_THREAD_POOL_WORKERS` — cap on concurrent blocking orchestration threads **per worker process**
- `DB_POOL_MIN` / `DB_POOL_MAX` — psycopg2 pool **per backend process** (size against Postgres `max_connections`)

Backend — after FQDN is known

- `APP_BASE_URL` — `https://<backend-dev-fqdn>`
- `CORS_ALLOWED_ORIGINS` — `*` until frontend exists, then updated when frontend deploys

Backend — after frontend deploy

- `FRONTEND_BASE_URL` — `https://<frontend-dev-fqdn>`
- `CORS_ALLOWED_ORIGINS` — `https://<frontend-dev-fqdn>,*`
- Concurrency vars above are **re-applied** on this update so they are not lost.

Backend — first-time `containerapp create` only (plus the keys above)

Variable	Purpose (dev defaults in script)
`FLASK_DEBUG`	`1`
`ORCH_MODEL_BACKEND`	`openai`
`ORCH_OPENAI_MODEL`	`gpt-5-mini`
`ORCH_ENCRYPTION_ENABLED`	`0`
`ORCH_STREAM_RUN_TIMEOUT_SECONDS`	`120`
`ORCH_ADAPTIVE_STORE_ENABLED`	`1`
`ORCH_CONTEXT_COMPACTION_ENABLED`	`1`
`ORCH_MODEL_MINERS`	`1`
`MINER_ENROLLMENT_SECRET`	dev placeholder
`ELIS_MINER_CODE_SECRET`	dev placeholder
`ELIS_KEK_HEX`	dev placeholder
`MAINTENANCE_MODE`	`0`
`AUTH_REQUIRE_EMAIL_VERIFICATION`	`0`
`AUTH_REQUIRE_EMAIL_2FA`	`0`
`BROWSER_SERVICE_URL`	internal browser app URL
`BROWSER_SERVICE_AUTH_TOKEN`	dev token
`BROWSER_SERVICE_POOL_SIZE`	`2`
`BROWSER_SERVICE_FETCH_WAIT_SECONDS`	`0.4`
`BLOG_SCHEDULER_ENABLED`	`0`
`ROUTER_ALLOW_EXPLORATION_IN_PROD`	`1`

Frontend (`frontend-dev`) — Azure CLI

- `INTERNAL_API_URL` — `https://<backend-dev-fqdn>` (must match image build rewrite target; rebuild image if backend URL changes)

- `NEXT_PUBLIC_API_URL` — same (baked at build time for browser SSE; script sets runtime env too for consistency)

Frontend — Docker build (not Container App secrets; passed as `docker build --build-arg`)

- `INTERNAL_API_URL` / `NEXT_PUBLIC_API_URL` — resolved backend HTTPS URL
- `NEXT_PUBLIC_E2E_BRIDGE=1` — optional Selenium hook for dev images only

Set manually (secrets / DBA) — Azure CLI examples

Run after first deploy or when rotating secrets. Each command **merges** the listed vars with existing configuration.

```
az login
# Resource group and app names match dev topology
RG=amodelis-dev

# Required for API + DB
az containerapp update --name backend-dev --resource-group "$RG" \
  --set-env-vars "DATABASE_URL=postgresql://USER:PASS@elis-dev-pg.postgres.database.azure.com:5432/elis?
sslmode=require"

az containerapp update --name backend-dev --resource-group "$RG" \
  --set-env-vars "OPENAI_API_KEY=<your-key>"

# Strongly recommended
az containerapp update --name backend-dev --resource-group "$RG" \
  --set-env-vars "JWT_SECRET=<long-random-secret>"
```

Notes:

- `DATABASE_URL` must use the Azure Postgres FQDN and `sslmode=require`.
- `REDIS_URL` is maintained by `deploy-dev.ps1`; do not hand-edit unless you know the primary key / host changed.
- For **higher concurrency**, raise `$DevOrchThreadPoolWorkers` and `$DevDbPoolMax` in `deploy-dev.ps1`, increase `GUNICORN_WORKERS`, scale **backend-dev** replicas/CPU in Azure, and ensure Postgres **`max_connections`** ≥ (roughly) `replicas × GUNICORN_WORKERS × DB_POOL_MAX` plus overhead.

Optional / production-style extras

See `config/deployment.env.template` for names such as `STRIPE_*`, `DB_ENCRYPTION_KEY`, `DAILY_FREE_TOKEN_AMOUNT`, etc. Set with the same `az containerapp update --set-env-vars` pattern when needed.

Deploying Dev

Use `deploy-dev.ps1` from repo root.

Typical flows:
- First-time full setup:
- `./deploy-dev.ps1`
- Code-only redeploy:
- `./deploy-dev.ps1 -SkipInfra`
- Backend-only:
- `./deploy-dev.ps1 -Target backend -SkipInfra`
- Infra-only:
- `./deploy-dev.ps1 -Target infra`

Validation Checklist

1. Health check:
   - `GET https://backend-dev.livelydesert-e7e7480b.eastus.azurecontainerapps.io/health`
2. Auth smoke:
   - `POST /api/auth/signup` returns `201`
   - `POST /api/auth/login` returns `200`
3. Session smoke:
   - `POST /api/conversations` returns `201`
   - `GET /api/conversations` returns `200`
4. Frontend check:
   - `https://frontend-dev.livelydesert-e7e7480b.eastus.azurecontainerapps.io` returns `200`

Known Issues and Fixes Applied

Dev chat / SSE failing in the browser (resolved in `deploy-dev.ps1`)

The Next.js frontend **bakes** `NEXT_PUBLIC_API_URL` and `INTERNAL_API_URL` (API rewrite target) at **Docker image build** time. The deploy script previously passed `NEXT_PUBLIC_API_URL=TBD`, so the browser streamed chat to `https://TBD/api/...` and failed.

Fix: `deploy-dev.ps1` now resolves the backend Container App ingress FQDN (or the documented dev default) and passes the same HTTPS origin for both build args. Override anytime with:

`$env:DEV_BACKEND_PUBLIC_URL = "https://<your-backend-fqdn>"`

Then rebuild and redeploy the frontend image.

`deploy-dev.ps1` exiting early on `az containerapp update` (resolved)

On Windows PowerShell 5, Azure CLI progress on stderr can interact badly with `$ErrorActionPreference = 'Stop'`. The script now wraps those updates in `Invoke-DeployAz`, which temporarily sets `Continue` and checks `$LASTEXITCODE`. PowerShell 7+ also sets `$PSNativeCommandUseErrorActionPreference = $false` when available.

Fresh DB init ordering issues (resolved)

On fresh Postgres, schema init previously failed due to table ordering:
- `company_agents` referenced `companies` before `companies` existed.
- `ALTER TABLE conversations ...` executed before `conversations` existed.
- `_ensure_column()` attempted `ALTER TABLE` for tables not yet created.

Fixes now in code:
- Pre-create `companies` before `company_agents` FK creation.
- Pre-create `conversations` before `ALTER TABLE conversations`.
- Harden `_ensure_column()` to no-op if target table does not exist yet.

Operational Notes

- If using mutable tags like `dev-latest`, force a new Container App revision when needed by updating a harmless env var (for example `DEPLOY_TS`).
- Verify latest revision is ready after deploy before running E2E tests.
- Keep production untouched: do not deploy to `amodelis-prod` when validating dev.

Repository: Cluster Deploy

Plan: Dedicated Cluster Provisioning and Deploy Failure Hardening

Harden dedicated cluster lifecycle so create, repair, retry-provision, redeploy, and deployment-script cluster rollouts can recover from partial resource state and partial build/update failures. Every cluster update action must end with env apply plus verify after resource existence confirmation.

Steps

1. [x] Phase 1: Authoritative inventory contract (blocks phase 2)
2. [x] Add a reusable Azure inventory helper in backend/services/dedicated_cluster_provisioner.py that reports existence and canonical endpoints for frontend app, backend app, postgres server/db, redis, kbs, managed env, and resource group.
3. [x] Define canonical inventory payload shared by provision, repair, redeploy, and deep-health paths with explicit source markers (azure_discovered, db_persisted, computed).
4. [x] Phase 2: Partial-provision recovery flow (depends on 1; blocks phase 3)
5. [x] Refactor provision_cluster in backend/services/dedicated_cluster_provisioner.py to be per-resource idempotent: create when missing, validate when present, repair when present-but-invalid.
6. [x] Enforce dedicated frontend per cluster by ensuring frontend-{slug} exists and ingress/domain bindings are valid before writing frontend_fqdn.
7. [x] Add per-resource completion markers so retries resume from last successful checkpoint rather than replaying all steps.
8. [x] Phase 3: Env-last reconciliation for all cluster updates (depends on 2; blocks phase 4)
9. [x] Split cluster update into stage A resource reconciliation and stage B env apply plus verify.
10. [x] Gate env apply on confirmed dependency presence (postgres, redis, kbs, backend ingress, frontend ingress as applicable).
11. [x] Run deterministic env verification after each update action (create, repair-service, retry-provision, redeploy): DATABASE_URL, REDIS_URL, KBS_URL, APP_BASE_URL, FRONTEND_BASE_URL.
12. [x] Persist mismatch details to last_provision_error and prevent success status when verification fails.
13. [x] Phase 4: System-admin action contract hardening (depends on 2-3; blocks phase 5)
14. [x] Update force_redeploy_cluster_from_scratch, retry_cluster_provision, and repair_cluster_service in backend/services/system_admin_clusters_service.py so success responses include reconciliation and env verification outcomes.
15. [x] Do not return ok=true for frontend or dns repair when dedicated frontend app still does not exist post-repair.
16. [x] Require post-action metadata refresh from inventory before status transition to active.
17. [x] Phase 5: Truthful health checks and anti-cheat guards (depends on 1-4; blocks phase 6)
18. [x] Remove inferred frontend fallback as authoritative in

backend/services/system_admin_clusters_service.py deep-health paths.
19. [x] Probe only inventory-backed endpoints or clearly mark computed probes as non-authoritative.
20. [x] Add mismatch flags whenever DB metadata references non-existent Azure resources.
21. [x] Phase 6: Deployment pipeline cluster-rollout hardening (depends on 3; parallel with phase 7)
22. [x] Update deploy.ps1 dedicated rollout section to include explicit step outcomes per cluster: backend_update, frontend_update, env_reconcile, env_verify.
23. [x] Add per-cluster failure policy for partial builds and partial updates:
24. [x] shared image build or push failure: fail fast and stop rollout.
25. [x] per-cluster update failure: continue remaining clusters, mark cluster as failed, append to retry manifest.
26. [x] env verify failure after update: mark failed, no success count increment, include failing keys in summary.
27. [x] Add end-of-run machine-readable summary artifact (for example JSON) listing succeeded and failed clusters with reasons and which stage failed.
28. [x] Add optional retry mode input to deploy.ps1 that can re-run only failed clusters from the last summary artifact.
29. [x] Phase 7: Persistence and API schema updates (depends on 2-5; parallel with phase 6)
30. [x] Extend backend/agents/db.py cluster metadata fields if needed for inventory snapshot timestamp, verification status, and last verification error summary.
31. [x] Keep backend/system_admin_api.py handlers thin and return richer service-produced reconciliation and verification payloads.
32. [x] Phase 8: Regression and operational validation (depends on 6-7)
33. [x] Add targeted tests in backend/tests/test_system_admin_clusters_service.py for partial resource scenarios and env verification failures.
34. [x] Add deploy-flow validation checks for partial-cluster failure handling logic in deploy.ps1 (script-level test harness or dry-run path where feasible).
35. [x] Execute one-cluster validation runbook for elis-ai-ed8c0fe5 and one multi-cluster dry run validating partial failure reporting and retry behavior.

Relevant files

- backend/services/dedicated_cluster_provisioner.py — inventory contract, idempotent partial recovery, env-last reconcile and verify.
- backend/services/system_admin_clusters_service.py — action result contract hardening and truthful health probing.
- backend/system_admin_api.py — thin route handlers exposing enriched service outputs.
- backend/agents/db.py — persistence surfaces for reconciliation and verification status.
- deploy.ps1 — dedicated cluster rollout stages, partial-failure handling, retry manifest support.
- backend/tests/test_system_admin_clusters_service.py — partial-provision and env verify

regressions.
- backend/tests — provisioner and deploy-path validation additions.

Verification
1. Run targeted cluster service tests: python -m pytest backend/tests/test_system_admin_clusters_service.py -q
2. Run provisioner-focused tests: python -m pytest backend/tests -k "provisioner and (partial or reconcile or env)" -q
3. Validate system-admin API responses include reconciliation and env verification results after repair, retry-provision, and redeploy.
4. Dry-run or controlled deploy test for deploy.ps1 to confirm summary artifact, accurate success or fail counts, and retry-only-failed flow.
5. Confirm dedicated backend env never points to shared postgres or redis hosts for dedicated clusters.

Decisions
- Frontend architecture decision: dedicated frontend app per cluster is required.
- Mandatory rule: every cluster update action must run post-update env apply plus verify before reporting success.
- Deployment requirement: deployment flow must support dedicated cluster rollout with explicit partial-failure semantics.
- Excluded scope: unrelated router-chain and agent-chain logic outside dedicated cluster lifecycle.
- Guardrail: no inferred endpoint may be treated authoritative without Azure existence confirmation.

1. [x] Add automatic ticket or alert creation for clusters left in failed state after deploy.
2. [x] Add nightly integrity auditor to reconcile inventory and env drift across all active dedicated clusters.

Repository: Dedicated Hosts

Implementation Status (2026-04-10)

This section is the source of truth for completion checkoff.

Completed

- [x] Dedicated cluster billing add-on and setup fee
- [x] Org creation supports deployment type (shared or dedicated)
- [x] Cluster metadata model and status lifecycle
- [x] Dedicated checkout and confirm APIs
- [x] Dedicated cluster status API and frontend dashboard card
- [x] BYOK API keys (backend, frontend, enforcement)
- [x] BYOA OAuth credentials (backend, frontend, provider resolution)
- [x] Org model controls (allowed and blocked models)
- [x] Dedicated miners assignment and dedicated-only enforcement
- [x] Domain allowlist resource locking for web tools
- [x] Plan-gated enterprise UI sections
- [x] Org creation abuse prevention (user org cap)
- [x] Owner lockout recovery (owner transfer and system admin override)
- [x] Tenant-scoped auth guard (JWT company claim enforcement)
- [x] Stripe webhook subscription lifecycle handling
- [x] Grace mode write blocking for dedicated clusters
- [x] Dedicated deploy automation script (`deploy-dedicated.ps1`)
- [x] Dedicated teardown automation script (`teardown-dedicated.ps1`)

Verification Snapshot

- Billing and enterprise suites pass together: 84 passing, 0 failing.
- Coverage includes dedicated billing routes, deployment type validation, miner enforcement, owner transfer, webhook lifecycle transitions (updated, deleted, payment_failed), tenant claim enforcement, grace-mode write blocking, and enterprise settings flows.

Operational Notes

- Full enterprise data export and archival pipeline (scheduled pg_dump plus blob lifecycle) should be tracked as an operations runbook item if you want this automated beyond script-level teardown.
- Grace reminders by email are partially enabled by existing billing lifecycle hooks and email service; production scheduling policy should be finalized in ops.
- Retention policy is now standardized with a 5-year default chain-run window and a dedicated grace/archive queue flow for unpaid orgs. Use `docs/operations/platform-retention-plan.md` as the operational source of truth.

Docs: On-Prem Quickstart

Elis AI On-Prem Quick Start

What you receive

- Docker Compose application bundle with prebuilt protected images
- one-command env initialization scripts
- request-code based licensing flow for an installation-bound activation code
- deployment and operations docs

Required dependencies

Install these before starting the platform:

- Docker Desktop for Windows or macOS: https://www.docker.com/products/docker-desktop/
- Docker Engine for Linux: https://docs.docker.com/engine/install/
- Docker Compose plugin: https://docs.docker.com/compose/install/
- PowerShell for the Windows init helper: https://learn.microsoft.com/powershell/
- Python 3 for the Linux/macOS init helper: https://www.python.org/downloads/

Deploy

1. Extract the bundle.
2. Initialize the bundle. This generates unique local secrets, writes the required `.env` files, prints or returns a one-time deployment bootstrap token, and loads the protected Docker images:
   - Windows PowerShell:
     - powershell -ExecutionPolicy Bypass -File .\scripts\init_onprem_bundle.ps1
     - Linux/macOS:
     - bash ./scripts/init_onprem_bundle.sh
3. Review the generated `.env` files only if you need non-default hostnames, ports, reverse-proxy URLs, customer-managed OAuth, or an external OpenAI-compatible model backend.
4. Keep all runtime URLs local to the onsite environment. Do not point `APP_BASE_URL`, `FRONTEND_BASE_URL`, `INTERNAL_API_URL`, `NEXT_PUBLIC_API_URL`, or `COMMUNITY_SITE_URL` at `tryelisai.com` or any Azure Container Apps hostname.

5. Leave hosted billing and Azure integration vars empty on the onsite runtime. The onsite stack should not carry Stripe keys, Azure WAF/Log Analytics IDs, or the on-prem bundle blob connection string.
6. Start the stack:
   - docker compose up -d
7. Save the generated `ONSITE_BOOTSTRAP_TOKEN` securely. It is required exactly once to create the initial onsite system admin account.
8. Open the onsite app and create the first account using the deployment bootstrap token. That first successful claim becomes the local system admin for the deployment and permanently consumes the bootstrap path.
9. Sign in as that system admin and generate a request code from the system admin license screen.
10. Open the hosted Billing page for the same organization, paste the request code or installation ID, and generate an installation-bound activation code.
11. Install that activation code on the onsite system admin license screen.
12. Verify the API is healthy:
    - curl http://localhost:8000/health

The shipped compose file does not mount the source tree. The on-prem backend image is bytecode-only, the onsite miner image is compiled/obfuscated, and the frontend image strips source maps during the release build.

Required standalone settings

- `ONSITE=1` on the backend runtime
- `NEXT_PUBLIC_ONSITE=1` in the frontend runtime
- unique `JWT_SECRET`
- unique `DB_ENCRYPTION_KEY`
- unique `ELIS_KEK_HEX`
- unique `BROWSER_SERVICE_AUTH_TOKEN`
- unique `ONSITE_BOOTSTRAP_TOKEN`
- unique Postgres password in `DATABASE_URL` / `PG_ADMIN_URL`

The init helper generates these automatically. The backend will still refuse to start if an onsite install is configured with known dev/shared defaults or with hosted TryElisAI/Azure URLs.

First admin bootstrap

- The bundle ships with no default admin credentials.
- The init helper generates `ONSITE_BOOTSTRAP_TOKEN` per deployment.
- The first successful account creation or bootstrap claim using that token becomes the initial local system admin.
- After that succeeds once, the bootstrap path is permanently blocked and the deployment records who claimed it and when.

URL and service settings

- `APP_BASE_URL` should be the backend origin that operators use inside the onsite environment. Default: `http://localhost:8000`.
- `FRONTEND_BASE_URL` should be the frontend origin for the onsite install. Default: `http://localhost:3000`.
- `INTERNAL_API_URL` should stay on the internal Docker network. Default: `http://backend:8000`.
- `NEXT_PUBLIC_API_URL` should be the browser-reachable backend origin. Default: `http://localhost:8000`.
- `KBS_URL` should stay `http://kbs:8500` unless you intentionally move the KBS service.
- `BROWSER_SERVICE_URL` should stay `http://browser:9020` unless you intentionally move the browser service.
- `COMMUNITY_SITE_URL` should stay empty on the onsite runtime.

Runtime defaults baked into the bundle

- `AUTH_REQUIRE_EMAIL_2FA=0` is the onsite backend default.
- Set `AUTH_REQUIRE_EMAIL_2FA=1` only for customer deployments that explicitly require emailed MFA and have Azure or SMTP email delivery configured.
- The default exposed ports are `3000` for the frontend and `8000` for the backend.
- `docker-compose.yml` in the bundle already maps the frontend `NEXT_PUBLIC_ONSITE=1` and backend `ONSITE=1` flags.

Model backend choice

On-prem installs can use either local Ollama or external OpenAI-compatible APIs.

- `ORCH_MODEL_BACKEND=ollama`

- Default bundle mode.
- Uses the bundled protected local miner with `ORCH_OLLAMA_URL=http://miner-qwen-1:11434`.
- Set `NO_EXTERNAL_EGRESS=1` if you want a stricter no-egress / airgapped posture.
- `ORCH_MODEL_BACKEND=openai`
- Supported for non-airgapped onsite installs.
- Set `OPENAI_API_KEY` and keep `NO_EXTERNAL_EGRESS=0`.
- The onsite runtime still must not point at hosted TryElisAI or Azure platform endpoints.

Minimal manual settings

The default bundle is intended to boot locally without hand-editing every service.
Most installs only need manual changes when one of these is true:

- you want non-localhost frontend or backend URLs
- you want customer-managed Google or Microsoft OAuth
- you want custom port bindings or reverse-proxy integration
- you want to use OpenAI-compatible APIs instead of the default local Ollama-backed miner profile
- you want to scale beyond the default local Ollama-backed miner profile

Optional customer-managed settings

- `NEXT_PUBLIC_GOOGLE_CLIENT_ID` for Google login on the onsite frontend
- `NEXT_PUBLIC_MICROSOFT_AUTH_CLIENT_ID`, `NEXT_PUBLIC_MICROSOFT_AUTH_TENANT_ID`, `MICROSOFT_AUTH_CLIENT_ID`, and `MICROSOFT_AUTH_TENANT_ID` for Microsoft login
- reverse-proxy or DNS changes reflected in `APP_BASE_URL`, `FRONTEND_BASE_URL`, and `NEXT_PUBLIC_API_URL`
- `NO_EXTERNAL_EGRESS=1` if you want a stricter no-egress or air-gapped posture

Values that must remain empty onsite

- `STRIPE_SECRET_KEY`
- `STRIPE_PUBLISHABLE_KEY`
- `AZURE_WAF_POLICY_ID`
- `AZURE_LOG_WORKSPACE_ID`

- `ONPREM_BUNDLE_BLOB_CONNECTION_STRING`

Seat licensing

Seat counts are enforced by the signed activation code issued after checkout.
The runtime ONSITE flag only marks deployment posture and does not grant extra seats.
The activation code is tied to the installation ID from the onsite request code and cannot be replayed on a different install.

Updates

Purchase renewals and additional seat blocks are managed from the Billing page in the hosted control plane.
To add more seats later, buy additional seat blocks for the same organization, generate a fresh request code from the onsite install, and then issue a replacement activation code for that installation. A reinstall is not required.
Each activation code is tied to the installation ID from the onsite request code. Reusing that code on a different installation will fail.
Installing the refreshed activation code replaces the previous onsite seat and expiry limits for that installation.
See the product docs at /docs/onprem-install and the walkthrough at /tutorials/dev/onprem-install for the full visual setup guide.

Docs: Offline Update Guide

Offline Update Guide (Air-Gapped)

Overview

This process updates an isolated deployment with no direct internet egress.

Update Flow

1. Build images in connected environment
   - Build backend, frontend, miner, and required sidecar images.
2. Export images
   - Save images to tar archives:

```
docker save <image:tag> -o image.tar
```

3. Transfer artifacts to offline environment
   - Use approved media transfer process and integrity checksums.
4. Load images in offline environment

```
docker load -i image.tar
```

5. Apply database migrations
   - Start backend with migration runner enabled and verify migration logs.
6. Verify deployment
   - Run health checks and critical smoke tests.
   - Confirm compliance and audit endpoints are reachable.
7. Rollback if needed
   - Execute scripts/rollback.sh with previous image tags and backup path.

Pre-Deployment Checklist

- backup completed and verified
- rollback package prepared
- signed change approval obtained
- release notes and migration notes reviewed

Docs: Miner Setup Guide

Elis Miner Setup Guide

Run your own inference miner and connect it to the Elis backend.

Prerequisites

Requirement	Minimum
Docker	24.x+
RAM	4 GB (OpenAI) / 16 GB (Ollama)
Disk	2 GB (OpenAI) / 20 GB (Ollama, model-dependent)
Network	Outbound HTTPS to the Elis backend and any model provider you use
OpenAI API key	Required when `MINER_BACKEND=openai`

1. Download the miner

The backend serves the current miner packages from `/api/downloads`:

- `elis-miner-linux-x86_64.tar.gz` - Linux Docker image tarball
- `elis-miner-win-x86_64.zip` - Windows standalone package with `run.bat`

Linux example:

```
curl -O https://api.tryelisai.com/api/downloads/elis-miner-linux-x86_64.tar.gz
docker load < elis-miner-linux-x86_64.tar.gz
docker images | grep elis-miner
```

Windows example:

```
Invoke-WebRequest `
  -Uri https://api.tryelisai.com/api/downloads/elis-miner-win-x86_64.zip `
  -OutFile elis-miner-win-x86_64.zip
Expand-Archive elis-miner-win-x86_64.zip -DestinationPath .\elis-miner
```

2. Configure environment variables

Copy the template and fill in your values:

```
cp miner/.env.template miner/.env
```

Required values:

Variable	Required	Description
`BACKEND_URL`	Yes	Elis backend URL, for example `https://api.tryelisai.com`
`MINER_BACKEND`	Yes	`openai`, `ollama`, or `stub`
`MODEL_NAME`	Yes	Supported catalog model name, for example `gpt-5-mini`
`OPENAI_API_KEY`	Yes*	Required when `MINER_BACKEND=openai`

Optional values:

Variable	Required	Description
`MINER_REGISTRATION_TOKEN`	No	Optional dashboard-generated token that reserves the miner for your account before validation
`WORKER_ID`	No	Display name for this miner (defaults to hostname)
`MINER_CALLBACK_URL`	No	Direct HTTP endpoint the backend can call when it can reach the miner
`MINER_STATE_DIR`	No	Directory used to persist the miner ID and pairing key across restarts
`OPENAI_BASE_URL`	No	Custom OpenAI-compatible API URL
`OLLAMA_HOST`	No	Ollama server URL (default: `http://localhost:11434`)
`MINER_ENROLLMENT_SECRET`	No	Legacy shared-secret enrollment path for admin-managed deployments
`MINER_HEARTBEAT_INTERVAL`	No	Heartbeat interval in seconds (default: `15`)
`PORT`	No	Local listen port (default: `8080`)
`KBS_URL`	No	Key Broker Service URL for encrypted inference
`KBS_ATTESTATION_SECRET`	No	HMAC secret for KBS attestation
`OPENAI_MODEL`	No	Legacy alias for `MODEL_NAME`

Notes:

- `MODEL_NAME` is the canonical model setting for every miner backend.
- The backend validates `MODEL_NAME` against the platform catalog during registration.
- If the model is not in the catalog, registration is denied.

3. Choose a supported model

Each miner serves exactly one model. Pick a name that already exists in the Elis catalog.

OpenAI-backed example:

```
MINER_BACKEND=openai
MODEL_NAME=gpt-5-mini
OPENAI_API_KEY=sk-...
```

Ollama example:

```
MINER_BACKEND=ollama
MODEL_NAME=qwen2.5:0.5b
OLLAMA_HOST=http://localhost:11434
```

See config/model_catalog.json or query `GET /api/models` for supported models.

4. Run the miner

Docker

```
docker run -d \
  --name elis-miner \
  --env-file miner/.env \
  -v elis-miner-state:/var/lib/elis-miner \
  -p 8080:8080 \
  elis-miner:latest
```

Use `--gpus all` if your host supports GPU passthrough.

If you are using Docker Compose, persist the state volume so the miner keeps the same ID and pairing key across restarts:

```
services:
  miner:
    image: elis-miner:latest
    env_file: ./miner/.env
```

```
    environment:
      MODEL_NAME: gpt-5-mini
      MINER_BACKEND: openai
      WORKER_ID: my-miner-1
      MINER_CALLBACK_URL: http://miner:8080
    ports:
      - "8080:8080"
    volumes:
      - miner-state:/var/lib/elis-miner
    restart: unless-stopped

volumes:
  miner-state:
```

Windows standalone package

1. Copy `.env.template` to `.env`
2. Fill in the variables above
3. Run `run.bat`

5. Registration and pairing flow

On startup the miner:

1. Calls `POST /api/miners/auto-enroll` with `MODEL_NAME`, its miner-generated pairing key, and any optional registration token
2. Receives a miner ID and session token if the model is accepted
3. Prints the pairing key locally
4. Connects to `/ws/miner` for heartbeats and job dispatch
5. Accepts HTTP dispatch too when `MINER_CALLBACK_URL` is set and reachable

Important behavior:

- Registration tokens are optional. They reserve the miner for a user account but do not activate it.
- The miner-generated pairing key is the final validation step.
- A miner can connect before validation, but it will not receive jobs or earn credit until paired from the dashboard.
- Miner capability metadata is ignored. Supported models and capabilities come from the platform catalog.

6. Finish validation in the dashboard

1. Open the **Miners** page in the Elis dashboard
2. Optionally generate a registration token before starting the miner
3. Start the miner and copy the pairing key it prints on boot
4. Paste that key into the pairing form on the **Miners** page
5. Confirm the miner moves from awaiting validation to an active state

7. Verify health

```
curl http://localhost:8080/health
```

Expected response:

```
{"ok": true, "model": "gpt-5-mini", "backend": "openai"}
```

Also verify the miner appears on the dashboard with:

- the expected `model_name`
- `validation_status=validated`
- a recent heartbeat

8. Docker Compose service rules

When adding miners to a shared Compose stack:

- Set a unique `WORKER_ID`
- Keep `MODEL_NAME` to a single catalog model per miner
- Use a persistent state volume for `/var/lib/elis-miner`
- If you set `MINER_CALLBACK_URL`, use a URL the backend container can actually reach
- Do not rely on miner-supplied capabilities; the backend ignores them

9. Troubleshooting

Symptom	What to check
`AUTO_ENROLL_FAIL status=400`	`MODEL_NAME` is missing or not in the platform catalog
`AUTO_ENROLL_FAIL status=403`	`MINER_ENROLLMENT_SECRET` is wrong for a legacy/admin-managed flow
Miner is online but not getting jobs	Check whether it is still awaiting validation on the dashboard
Miner keeps generating a new pairing key	Persist `MINER_STATE_DIR` or mount `/var/lib/elis-miner`
No HTTP jobs arrive	`MINER_CALLBACK_URL` is missing, unreachable, or points to `localhost` from a remote backend
OpenAI `401 Unauthorized`	`OPENAI_API_KEY` is invalid
Ollama model is missing	Confirm `MODEL_NAME` exists locally or Ollama can pull it from `OLLAMA_HOST`

10. Build the downloadable packages

The `/downloads` page is backed by the archives in `dist/`. Rebuild them with:

```
.\miner\build_downloads.ps1
```

`deploy.ps1` now runs this automatically before backend builds so the deployed `/downloads` page stays in sync with the miner code.

Docs: Disaster Recovery

Disaster Recovery Plan

Objectives

- Recovery Time Objective (RTO): under 4 hours
- Recovery Point Objective (RPO): last successful backup (continuous PITR for managed Postgres where enabled)

Recovery Workflow

1. Incident declaration and scope confirmation
   - Confirm affected components and impact radius.
   - Freeze mutating operations when required.
2. Database recovery
   - Restore latest validated backup:

```
pg_restore -d elis < backup.sql
```

- For managed Postgres with PITR, restore to target timestamp and validate integrity.

3. Redis recovery
   - Restore append-only file (AOF) backup to Redis data volume.
   - Restart Redis and verify keyspace integrity for session/runtime caches.
4. Application layer recovery
   - Deploy known-good backend/frontend/miner images.
   - Validate health endpoints and core auth -> chat -> trace path.
5. DNS and traffic failover
   - Update DNS routing to recovered environment.
   - Verify TLS/cert and ingress behavior post-cutover.
6. Validation and closeout
   - Validate key user workflows.
   - Confirm audit and incident timelines are complete.

Evidence Checklist

- backup identifier and timestamp
- restore command outputs

- health check proof
- post-restore smoke test results

Docs: Operations Orchestration Persistence

Orchestration persistence (operators)

Two independent settings affect how much is written after a model run:

`ORCH_ADAPTIVE_STORE_ENABLED`

When **true** (recommended in production for learning and feedback), the orchestrator persists adaptive data including **`chain_runs`**, step traces, execution records, and artifacts. **Automations user feedback** (thumbs on widget runs) expects a `chain_runs` row keyed by the orchestration **`run_id`** stored on `widget_runs.orchestration_run_id`.

When **false**, little durable chain data exists; widget-run feedback may return `feedback_not_ready`.

`ORCH_AUXILIARY_RUNSTORE_PERSIST`

When **true**, `main.run(..., persist=True)` also appends to **RunStore** (`run_log`, `chain_history`). Useful for ops debugging or analytics that read those tables.

When **false** (default for editor/widget orchestration entry points that pass `persist` from this flag), chat and automations still get **`chain_runs`** as long as adaptive store is enabled; only the generic RunStore append is skipped.

`WIDGET_FORCE_RESEARCH_ROUTE`

When **true**, scheduled/manual widget runs use a tenant context with **`speed_mode=deep`**, **`web_search_allowed=true`**, and **`router_trace.forced_widget_route=true`** after the run for auditability. Does not remove routing entirely; it biases execution toward deeper research.

Docs: Operations Platform Retention Plan

Platform Retention Plan

Updated: 2026-04-11

Policy Baseline

- Chain-run retention default: 1825 days (5 years).
- Scope: all organizations unless they explicitly set a different `chain_run_retention_days` value.
- Grace window for dedicated unpaid lifecycle: 90 days (configurable via `DEDICATED_GRACE_PERIOD_DAYS`).
- Archive format for unpaid dedicated orgs: full PostgreSQL export (`pg_dump`) stored in low-cost blob storage.
- Post-grace default action: decommission dedicated cluster after archive is recorded.

Runtime Enforcement

- Company settings default and validation enforce a positive integer retention window.
- Existing organizations with missing/null retention are backfilled by migration to the platform default.
- Background scheduler deletes expired `chain_runs` and dependent `chain_step_runs` in batches.
- Billing lifecycle marks dedicated clusters as archive-required when subscription state moves to grace (`past_due`, `unpaid`, `canceled`, `incomplete_expired`).

Operations Runbook (Runbook-First Model)

1. Monitor archive queue using `GET /api/system-admin/retention/queue`.
2. Export the dedicated database with `teardown-dedicated.ps1 -ExportDatabase` (or equivalent controlled export process).
3. Upload archive to blob storage and capture URI/checksum.
4. Record completion in platform metadata using:
   - `POST /api/system-admin/retention/queue/{company_id}/archive-record`
   - Body includes `archive_blob_uri`, optional `archive_checksum`, optional `archive_notes`.
5. If export is complete and grace has ended, set `decommission=true` in the archive-record call to mark the cluster decommissioned.

Environment Variables

- `CHAIN_RUN_RETENTION_DEFAULT_DAYS` (default: `1825`)
- `CHAIN_RUN_RETENTION_ENABLED` (default: `1`)
- `CHAIN_RUN_RETENTION_INTERVAL_HOURS` (default: `24`)
- `CHAIN_RUN_RETENTION_BATCH_SIZE` (default: `2000`)
- `DEDICATED_GRACE_PERIOD_DAYS` (default: `90`)

Notes

- This plan intentionally keeps archive execution in an operations runbook instead of in-request backend automation.
- Keeping the archive workflow operator-driven reduces credential/process orchestration risk in application runtime while preserving retrieval capability.

Site: /docs

Documentation

Learn how the Elis AI distributed intelligence network processes your requests.

How Elis AI Works

Elis AI is a distributed intelligence network that breaks complex questions into smaller, focused tasks and routes them to specialized AI agents running across a decentralized pool of compute miners. Instead of relying on a single large model, every request passes through a dynamic pipeline that picks the best-fit agent, dispatches inference to the best available miner, and returns a fully traced answer — so you always know how a response was produced.

🤖 Elis AI Mode

Automatically builds and reuses specialized agents. Each agent runs a multi-step workflow — expertise lookup, context retrieval, execution — using the best available model tier for each step.

🔓 Elis Unlocked

Advanced multi-pass workflow with planning, fact-checking, and iterative refinement. Best for complex tasks that need verification and deep research.

⛏️ Miners

Miners host local LLMs and earn tokens for verified responses. The network supports multiple model tiers for speed and quality tradeoffs.

🛡️ Verification

Every inference step is traced end-to-end. The scheduler selects the best available miner using weighted round-robin, trust tiers, and load awareness — ensuring fair distribution and reliable execution.

Why the network stays sustainable

Elis is built on a token economy where compute contributors are rewarded for useful work. Miners that return high-quality results quickly earn more assignments. Because miners compete regionally, users get served by nearby high-performing nodes — improving latency and experience.

For Consumers

Contribute background compute to earn tokens that offset usage costs.

Background contribution earns tokens.

Tokens are spent on model and tool usage.

Quality verification keeps rewards fair.

For Enterprise

End-to-end encryption, automated PII redaction, audit trails, and self-hosted deployment options. See Enterprise section →

Process Flow

1. User submits prompt → 2. Router chain selects agent → 3. Agent builds workflow → 4. Miners run inference → 5. Quality verification → 6. Traced response returned

Ready to try it?

Create an account and start asking questions — every answer comes with a full trace.

Get Started → View Whitepapers

Site: /tutorials

Guided Walkthroughs
Tutorials shaped like the landing page, but built for real product work.
Step-by-step guides to help you get the most out of Elis AI. Choose the track that matches how you use the platform.
User Guide
For everyone. Learn how to use the platform day to day, set up teams, and manage the resources tied to your workspace.
Sign Up & Log In
Create your account and access the platform.
Open
Starting a Conversation
Send your first prompt and see AI responses stream in.
Open
Document Editor
Writer documents (50 cap), AI rewrite & selection tools, links, merge, history, and side chat.
Open
Managing Messages
Edit, resend, copy, and delete messages.
Open
Creating an Organization
Set up an organization to collaborate with your team.
Open
Managing Members
Invite team members, set roles, and manage access.
Open
Uploading Documents
Upload files for your organization's knowledge base.
Open
Connecting Databases
Link an external database for context-aware queries.
Open
Creating & Managing Projects
Organize work into projects with linked resources.
Open
Viewing the Audit Log
Review all activity within your organization.
Open

Settings & API Keys
Manage your profile and create API keys.
Open
Billing & Tokens
Purchase token packages and view transaction history.
Open
Subscription Management
Cancel or restore personal and organization subscriptions before access changes take effect.
Open
Organization Seat Management
Lower seat blocks safely by choosing which members or pending invites lose access.
Open
Browsing Models
Explore the model directory and compare options.
Open
Accepting Invitations
View and respond to pending organization invites.
Open
Project Conversations
Create and manage conversations within a project context.
Open
Using Dashboards
Create dashboards, iterate drafts, publish versions, and manage favorites.
Open
AI Edit a Dashboard
Edit dashboards in plain English with a live preview, clarify questions, and per-message restore points.
Open
Dashboard Editor Deep Dive
Understand each editor section: planning, AI spec edits, tasks, buckets, datasources, draft thread, and assistants.
Open
Using Automations
Organization-only recurring widgets with mobile support, chart modes, and alert dispatch.
Open
Deep Data Collection
Crawl websites, extract structured records, share public datasets, and earn credits.
Open
Enterprise & Dedicated Hosting

Enterprise
Dedicated infrastructure, model governance, BYOK/BYOA credentials, organization-scoped miners, and lifecycle operations for enterprise deployments.
Dedicated Hosting Upgrade
Activate dedicated deployment, select shared vs. dedicated at org creation, and monitor cluster provisioning lifecycle.
Open
Dedicated User Management
Invite members with cluster-aware links, review pending signup invites, and adjust paid seat capacity without losing track of reserved seats.
Open
Enterprise Controls
Configure model allow/block lists, web domain allowlist, BYOK API keys, and BYOA OAuth credentials.
Open
Provider API Keys (BYOK)
Sign up with OpenAI, Anthropic, Google, Groq, Mistral, and Cerebras, then store each provider key on your organization.
Open
Dedicated Miners
Assign organization-scoped miners, generate registration tokens, and enforce dedicated-only inference.
Open
OAuth & SSO Setup
Configure BYOA OAuth credentials and optional SSO enforcement for enterprise cluster members.
Open
Dedicated Lifecycle Operations
Operate Stripe webhook lifecycle, grace mode, data export, and cluster decommission procedures.
Open
Developer Guide
For admins and developers. Go deeper on miner management, API integration, observability, and operator tooling.
On-Prem Installation
Purchase the annual license, download the Docker bundle, install dependencies, and activate the platform.
Open
Setting Up a Miner
Hosted or self-hosted: BACKEND_URL, MODEL_NAME, pairing key, Docker or Windows.
Open

Managing Miners

Monitor health, filter workers, enable/disable, validate pairing, unclaim when needed.

Open

Using the API

Authenticate and make programmatic requests.

Open

Automations API

Create, run, and monitor recurring research widgets via REST.

Open

Conversation Explorer

Browse and filter conversations with run telemetry.

Open

Agent Directory & Detail

Search agents, view stats, and explore mutations.

Open

Runs Dashboard & Comparison

Filter runs and compare them side by side.

Open

Run Detail Deep Dive

Inspect Overview, Routing, Steps, Context, and Integrity tabs.

Open

Anomaly Dashboard

Filter by anomaly type and severity, then drill into flagged runs.

Open

Downloads

Access compiled packages and Docker images.

Open