BOL - Beginning of Life Simulator

System Architecture Document

Generated: 2026-05-12 00:52
Version 1.0.0

1. System Overview

BOL (Beginning of Life) is a full-stack Python application that simulates prebiotic chemistry and the emergence of protocells. It operates as a single-server web application with no external database dependencies. All persistent data is stored as JSON files on the local filesystem.

Key characteristics: self-contained (no external databases or microservices), single-process (Flask + simulation thread), file-based persistence (JSON), real-time polling-based updates.

2. Deployment Topology - Production

Layer 1: Client (Browser/PWA) - Dashboard, Microscope, AI Lab, Projects. Uses Chart.js, Three.js, vanilla JS.

Layer 2: Nginx Reverse Proxy - SSL termination (Let's Encrypt), bol.drel.us -> 127.0.0.1:5000, security headers (CSP, HSTS, X-Frame-Options), static file serving, 10 MB upload limit.

Layer 3: Gunicorn WSGI Server - 3 worker processes, 120s timeout, managed by systemd, auto-restart on failure (RestartSec=5).

Layer 4: Flask Application - 30+ HTML templates, 40+ REST API routes, authentication, security middleware, PDF generation.

Layer 5: Simulation Engine - background thread, 27 reactions, micelle/vesicle assembly, RNA replication, metrics collection.

Layer 6: File System Storage - profiles/*.json, projects/*.json, scenarios/*.json, training/modules.json, bugs/bugs.json, .secret_key, output/.

Target: GCP Compute Engine e2-small (2 vCPU, 2 GB RAM), Ubuntu 24.04 LTS.

2.1 Local Development

Browser -> http://127.0.0.1:5000 -> Flask dev server (single thread). No Nginx, no SSL, no Gunicorn. SESSION_COOKIE_SECURE=False.

2.2 Network Configuration

Port 22 (SSH): admin only, key-based auth. Port 80 (HTTP): public, redirects to 443. Port 443 (HTTPS): public, Nginx -> Gunicorn. Port 5000 (Gunicorn): localhost only.

3. Security Architecture

NIST SP 800-53 controls: AC-7 (login rate limiting, 10/5min per IP), IA-5 (PBKDF2-HMAC-SHA256, 260K iterations, min 8 chars + complexity), SC-8 (TLS 1.2+ via Let's Encrypt, HSTS 1-year), SC-13 (PBKDF2 passwords, secrets.token_hex, timing-safe compare), SI-10 (allowlist validation on all user inputs), SI-11 (generic error messages to users), AC-3 (resource ownership, admin-only endpoints), AU-2 (Gunicorn access/error logs).

CISA compliance: BOD 18-01 (HTTPS enforcement, HSTS, Secure cookies), BOD 22-01 (dependencies

pinned to recent versions).

3.1 Security Headers

Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-inline' CDN...; style-src 'self' 'unsafe-inline' CDN...; font-src 'self' CDN...; img-src 'self' data:; connect-src 'self'; frame-ancestors 'none'.

Strict-Transport-Security: max-age=31536000; includeSubDomains. X-Content-Type-Options: nosniff. X-Frame-Options: SAMEORIGIN. Referrer-Policy: strict-origin-when-cross-origin. Permissions-Policy: camera=(), microphone=(), geolocation=(), payment=(). X-XSS-Protection: 1; mode=block.

3.2 Authentication & Sessions

Password storage: PBKDF2-HMAC-SHA256, 260,000 iterations, 32-byte random salt per user. Password policy: minimum 8 characters, must contain uppercase, lowercase, and digit.

Session cookies: HttpOnly (no JS access), SameSite=Lax (CSRF mitigation), Secure flag (production only). 8-hour session lifetime. Secret key persisted in .secret_key file (gitignored).

Rate limiting: 10 login attempts per 5-minute window per IP address. Returns 429 Too Many Requests on excess. Uses in-memory tracking with automatic pruning.

3.3 Input Validation & Output Encoding

Server-side: usernames validated against [a-zA-Z0-9_-] allowlist. Project IDs, lesson IDs, and filenames sanitised before filesystem access.

Template engine: Jinja2 auto-escaping enabled globally. No |safe filter usage. Client-side: escHtml(), _esc() helper functions and textContent for safe DOM updates.

Command injection prevention: shlex.split() with shell=False for subprocess. SSRF prevention: URL scheme validation (HTTP/HTTPS only).

4. Technology Stack

Backend: Python 3.12+, Flask >=3.0, Gunicorn, Nginx, NumPy >=1.24, SciPy >=1.10, Matplotlib >=3.7, Vispy >=0.14.

Frontend: Jinja2 templates, vanilla JavaScript, Chart.js (CDN), Three.js (CDN), Font Awesome 6 (CDN), CSS3 custom properties, Canvas 2D, Web Speech API, Service Worker.

Infrastructure: GCP Compute Engine e2-small, Ubuntu 24.04 LTS, systemd, Let's Encrypt + certbot, ufw firewall, local filesystem (JSON files).

5. Data Flow Architecture

Request flow: Browser -> HTTPS -> Nginx -> HTTP -> Gunicorn -> Flask Route. Page routes render Jinja2 templates to HTML. API routes return JSON.

Simulation flow: Config -> Simulation.init() -> Step Loop (background thread): diffuse -> energy update -> chemistry -> assembly -> replication -> metrics. Dashboard polls GET /api/state (1s) and GET /api/history (2s).

Auth flow: POST /api/profile/login -> rate limit check -> load profile JSON -> PBKDF2 hash with stored salt -> timing-safe compare -> set session cookie.

6. Operational Procedures

Service management: systemctl status/restart/stop bol. Logs: journalctl -u bol -f, /var/log/bol-access.log, /var/log/bol-error.log. Nginx: nginx -t, systemctl reload nginx.

Deployment update: (1) Run tests locally, (2) deploy_to_server.ps1 rsync, (3) systemctl restart bol, (4) verify HTTPS endpoint, (5) check logs.

SSL renewal: certbot configured for auto-renewal. Manual: certbot renew.

Backup: Git for code. Periodic backup of profiles/, projects/, bugs/, training/, .secret_key.

7. Scalability Considerations

Current capacity: ~50 concurrent browser sessions (3 Gunicorn workers), ~10,000 molecules per simulation, thousands of JSON files feasible on SSD.

Scaling options: vertical (larger GCP instance), more Gunicorn workers, database migration (SQLite/PostgreSQL if file I/O bottlenecks), CDN for static assets (CloudFlare or GCP Cloud CDN).