ARES
Server Guardian

ARES SUITE

ARES Server Guardian

Architecture

Detects attacks on your servers and blocks them at the firewall.

VERSION	PRODUCTION	DATE
1.6.0	guardian.server1.nacino.net	19 November 2026
AUTHOR	DEMO	CLASSIFICATION
Alfredo Nacino	guardian-demo.nacino.net	Internal

Contents

01 Purpose and scope

02 System context

03 Building blocks

04 Software components

05 Data

06 Security

07 Deployment

08 Decisions

09 Risks and constraints

1. Purpose and scope

ARES Server Guardian is a host intrusion-prevention service for Linux servers. It turns log lines and live connection counts into security events, decides per address when an attack is under way, and enforces bans and rate limits in the kernel firewall of the same server. This document covers version 1.6.0: the service, its two run modes (full install and agent), the link to ARES Central and how it is deployed in the `prod` and `pre` environments. ARES Central itself is a separate application with its own documents.

2. System context

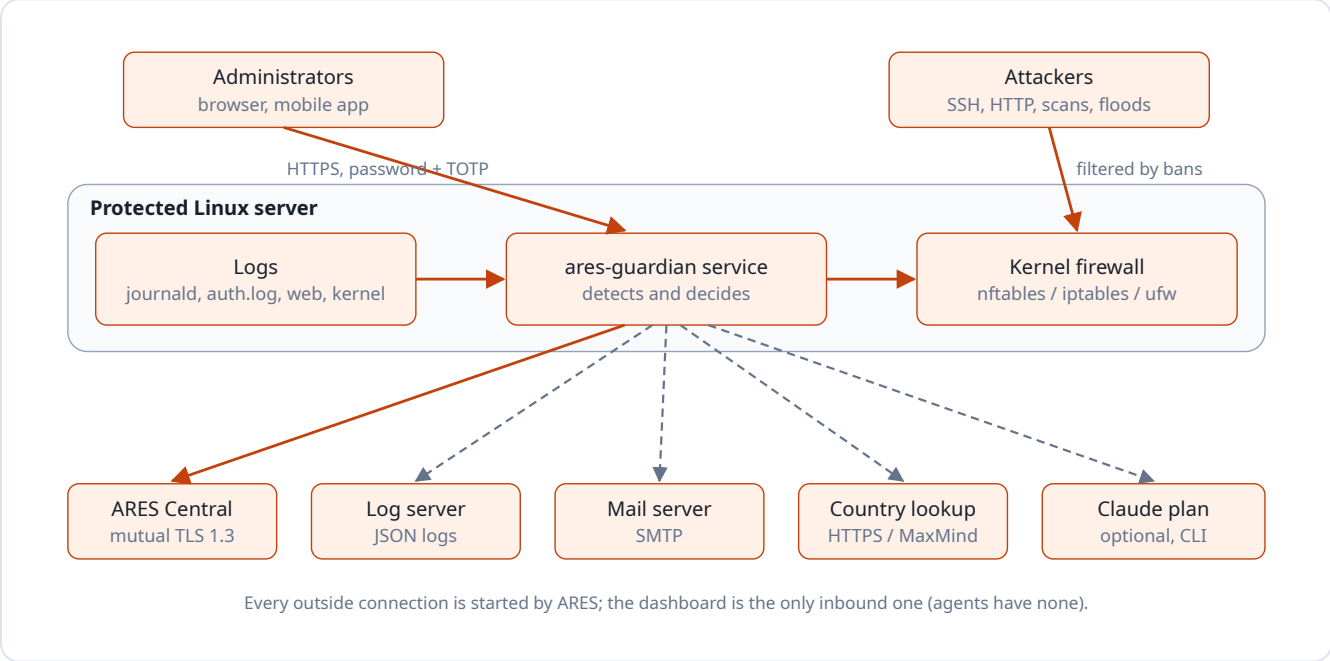


Figure 1. ARES Server Guardian and the systems around it

The service reads only what the server already logs. Every outside connection is started by ARES; the only inbound one is the dashboard, which an agent does not have.

3. Building blocks

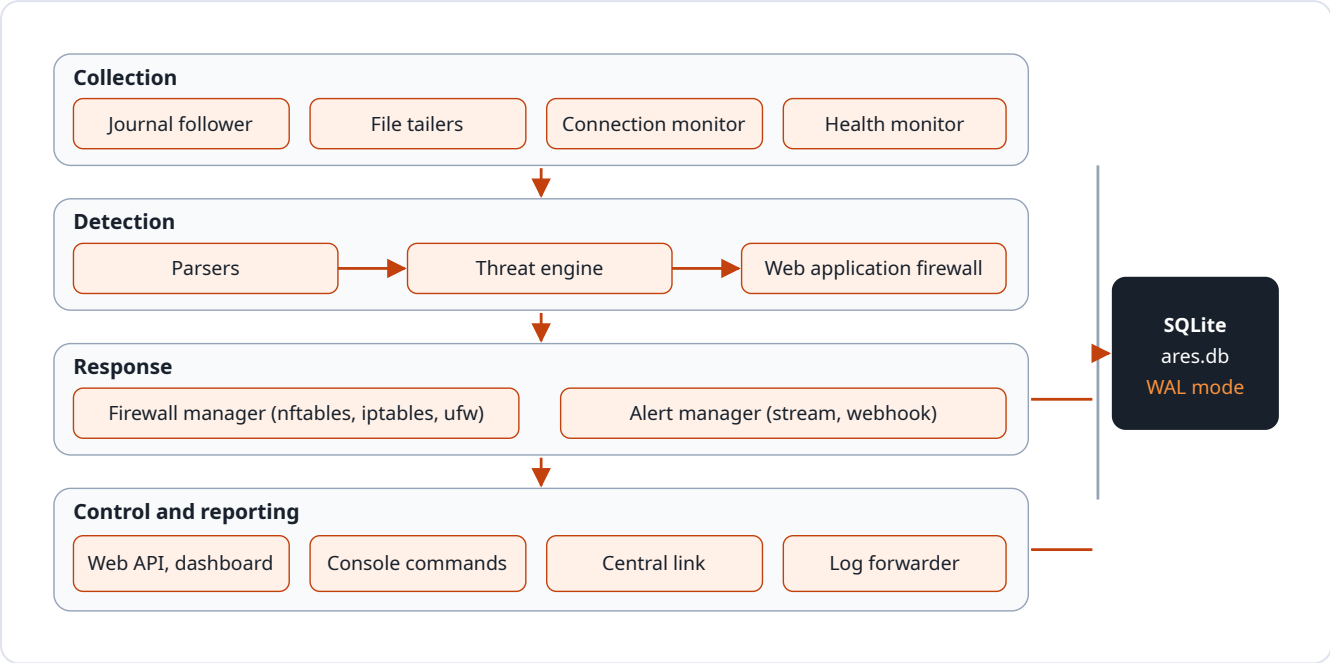


Figure 2. Building blocks inside the service

Block	Job
Collection	Follow journald (sshd, kernel) and log files, sample connections and system health.
Detection	Parse lines into typed events; count them per detector and address in sliding windows; classify web requests against signatures.
Response	Keep the desired firewall rules in the database and render them to the backend; create, de-duplicate and push alerts.
Control and reporting	Dashboard and REST API, console commands, the link to ARES Central, forwarding to a log server, reports and AI analysis.

4. Software components

Component	Module	Job
Service container	`guardian.py`	Builds every component, starts the workers, runs the scheduler (expire bans, refresh DNS names, prune, daily report, backups).
Agent runner	`agent.py`	Agent mode: runs the service with no web app until SIGTERM.
Log followers	`monitors.py`, `parsers.py`	journald and file tailing; SSH, web access and kernel line parsers.

Component	Module	Job
Connection and health monitors	`monitors.py`	Connection and SYN floods; CPU, memory, disk, load, processes.
Threat engine	`engine.py`	Detectors, incidents, attacker profiles, brute-force guard, repeat offenders.
Web application firewall	`waf.py`	Ten signature groups and custom rules per site and path.
Firewall manager	`firewall.py`	Rules in the database, rendered atomically to nftables (`inet ares_guardian`), iptables (`ARES` , `ARES6`) or ufw.
Alerts	`alerts.py`	Alerts, live stream, webhook.
Web API and dashboard	`app.py` , `api/` , `static/`	FastAPI app: REST API under `/api/v1` , WebSocket stream, single-page dashboard, `/healthz` .
Access gate	`access.py` , `api/gate.py`	Allowed addresses and remote access links, checked before any route.
Central link	`central.py`	Mutual-TLS WebSocket to ARES Central; reports and signed commands.
Logs, backups, AI	`logview.py` , `applog.py` , `logship.py` , `backup.py` , `ai.py` , `claudecode.py`	Log viewer and tracing, forwarding, backups, AI analysis.
Console	`__main__.py`	`serve` , `agent` , `status` , rescue commands (`unban` , `whitelist` , `flush-firewall`), `central join` .

Every collector and the Central link run as threads in one process. Detection is synchronous per event, so a ban is applied within the same second the threshold is reached.

5. Data

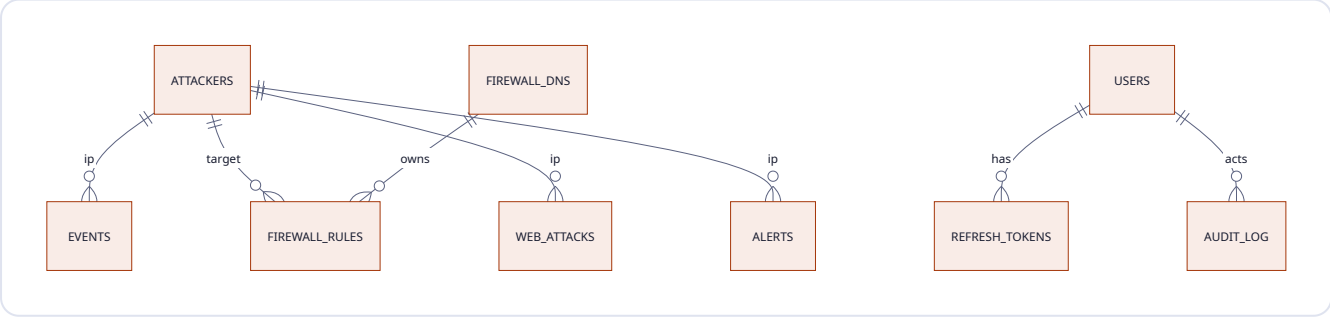


Figure 3. Main data entities

All state is in one SQLite database, `ares.db`, in WAL mode, in the data directory (`/var/lib/ares-guardian`, or `DATA_DIR`). Its 21 tables hold events, attackers, firewall rules and the whitelist, alerts, health samples, reports, AI analyses and usage, users, tokens and devices, the audit log, allowed addresses and links, web attacks, WAF rules, DNS names and a key-value store for settings. Beside it are `secret.key` (token signing), `central/` (this server's key and certificate, Central's CA) and `backups/`. Retention is 1 to 6 months for security logs and 14 days for health samples.

6. Security

Threat	Control
Locking out the administrators	The whitelist, private and loopback addresses and ARES Central's address are never banned; new installs start in dry-run; `flush-firewall` and `access disable` work from the console.
Stolen dashboard password	Mandatory TOTP, lockout after failures, short-lived access tokens, rotating refresh tokens, optional allowed-address list.
Forged command from "Central"	Central's private CA is pinned by fingerprint in the join code; every command is Ed25519-signed, names one server, expires in 5 minutes and runs once.
A compromised Central	Commands are a fixed list of firewall and alert operations; no shell or file access; a server can be made read-only.
Secrets in logs	Link and WebSocket tokens are redacted from access logs; passwords and tokens are never logged.
Attacker-written text reaching the AI	Claude Code runs with no tools, settings or MCP servers in an empty folder; it can only answer.

The service unit runs as root with `ProtectSystem=strict`, write access only to `/var/lib/ares-guardian`, `NoNewPrivileges`, a private `/tmp` and the capabilities it needs (`CAP_NET_ADMIN`, `CAP_NET_RAW`, `CAP_KILL`, `CAP_SYS_PTRACE`, `CAP_DAC_READ_SEARCH`).

7. Deployment

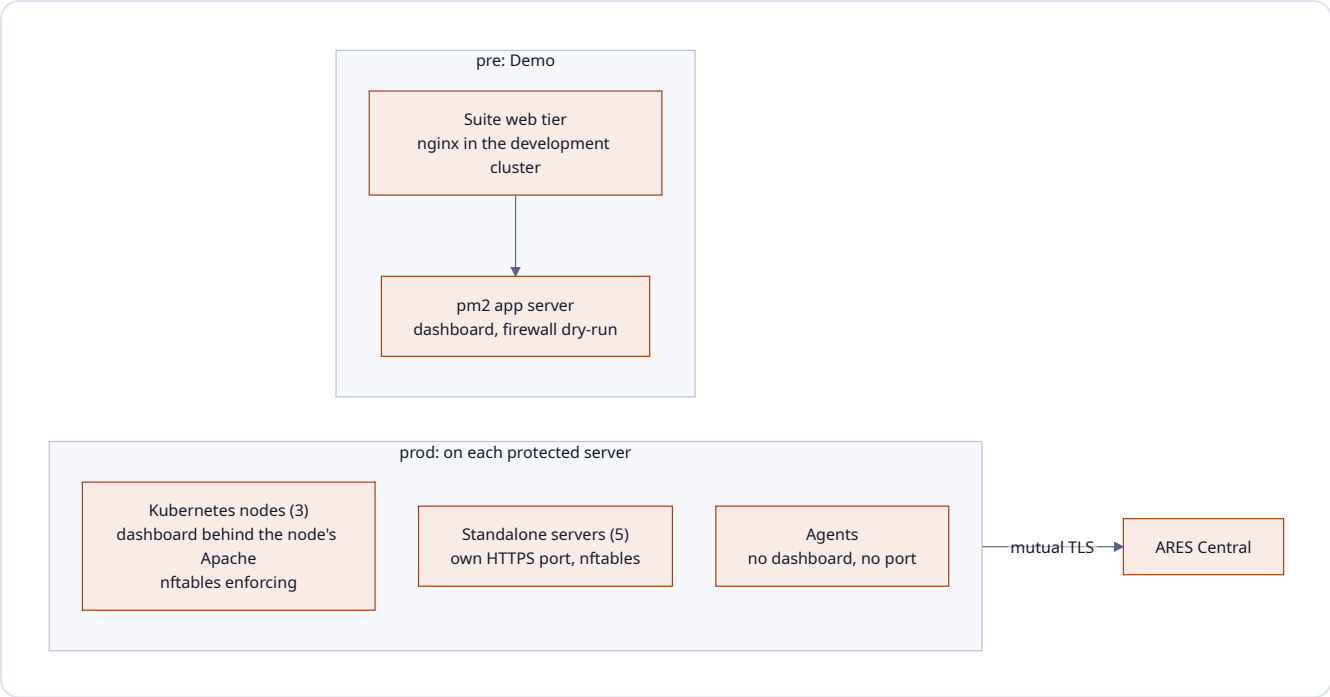


Figure 4. Environments

- **prod**: ARES runs on the host it protects as the `ares-guardian` systemd service, installed from the release bundle by `scripts/deploy-remote.sh` (servers in a local host list) or `scripts/install-remote.sh` (servers named on the command line). A firewall is never deployed as a cluster-wide workload: only the servers chosen get one.
- **pre**: `suite deploy pre` puts the dashboard on the pm2 app server with `deploy/demo.toml` (unprivileged, dry-run, not joined to Central) behind the Suite web tier at https://guardian-demo.nacino.net.

8. Decisions

Decision Run on the protected host, not in the cluster. The firewall, the logs and the processes ARES needs belong to the host; a pod would need the host's network and PID namespaces anyway.

Decision: Dial out to Central. No management port is opened on any server, and servers behind NAT or dynamic addresses join without router changes.

Decision: Agent mode is a setting, not a second product. `[server] web = false` runs the same code without the web app, so agents get every detection and firewall change with each release.

Decision: Rules live in the database and are rendered to the backend. A restart or a backend change re-applies the same rules; the kernel is never the source of truth.

9. Risks and constraints

- **Root access**: ARES needs firewall rights; a bug in rule rendering can block traffic. Mitigated by dry-run, the whitelist, atomic nftables updates and `flush-firewall`.
- **Log-based detection**: attacks that write no log line are seen only through connection counts.
- **Single SQLite writer**: fine for one server's events; very high event rates (large floods) are reduced by counting in memory and storing only security events.
- **Agents depend on Central**: an agent without a working Central link still blocks attacks, but nobody sees them until the link returns.