User manual

Complete product operation manual: enrolment, console, recipes, backups, recovery, licensing, security, report signing and integrations.

User manual — SEGRD Abzu v2

Document version: 2026-10-18 · Product: Abzu v2 (console + agents + licensing service, integrated `main` branch) · Language: English · Brand: SEGRD

> **How to read this manual.** It describes what the product truly does according to its code and current design. When a capability has a real limit — for example, an executor that only detects but does not execute, or a test against physical hardware still pending — the manual says so in the same section, with the label **Verified limit**. Nothing that follows promises a function that is not in the code. If a screen or message in your installation differs, what your installation shows and what is recorded in its audit log prevail.

Contents

1. Introduction and product forms
2. Requirements, installation and agent enrolment
3. The console: navigation and panels
4. Agents and unified inventory
5. Recipes: create, version, reconcile and capture
6. Recipe catalog
7. Backups
8. LTO tape
9. Disaster recovery (DRP)
10. Licensing and paid VM function
11. Security, roles and compliance
12. Report signing and "View evidence" validation
13. Integrations and notifications
14. Public API, SDKs and Terraform
15. Multi-site console federation
16. Observability: anomalies, predictions and KPIs
17. Mobile app (PWA)
18. Troubleshooting and error messages
19. Operating best practices

20. Glossary

1. Introduction and product forms

1.1 What it is

SEGRD Abzu is an operational continuity platform: a **central console** that governs a fleet of servers through **lightweight agents** installed on each server or cluster. Its pieces are:

- **Versioned recipes** (`abzu.recipe/v1`): the desired state of each application — services, images, volumes, networks, policies — declared in YAML. The recipe is the source of truth; the artifacts applied (Compose in Docker, manifests in Kubernetes) are generated from it and are not edited by hand.
- **Continuous anti-drift reconciliation**: each agent compares the real state against the desired state and converges or alerts according to the recipe policy.
- **Verified backups**: backup of multi-engine databases and volumes, with encryption in stream, retention and verification by real restoration in an isolated environment (sandbox).
- **Disaster recovery (DRP)**: versioned runbooks that rebuild in order — network → volumes → data → applications — with human approval in the baseline and, in v2, policy-governed autonomy.
- **Per-agent licensing**: each enrolled agent consumes a quota (seat); containers per agent are unlimited.

The access model is deliberately different from panels that keep administration credentials for all servers in a single place: **there are no centralised SSH keys and no inbound port is opened** on managed servers. The agent always initiates the connection toward the console (phone-home) and authenticates with its own certificate (mTLS). If an agent is compromised, the scope is that server, not the fleet.

1.2 The two forms of the product

The core is the same (agent, recipe engine, runbooks, backups, Spanish/English i18n). Packaging, identity and licensing change:

Aspect	Commercial form	Government form (turnkey)
Where it lives	Module integrable into the SEGRD platform	Self-contained installation on the institution's network, without depending on any cloud
Identity	The SEGRD platform is the identity provider (OIDC / native server-to-server login). No duplicate local users are created	Local users and roles + SSO with Entra ID (Microsoft 365) and Google Workspace + optional LDAP/AD
Licence	Annual subscription per agent; online validation against the technical licensing point	Perpetual per agent + optional annual support; signed file validated **offline**

Aspect	Commercial form	Government form (turnkey)
Licence assignment	Governed by the vendor's licence and platform panel (see §10). Abzu does not maintain a parallel assignment register	The same rule: assignment governs; the signed offline licence materialises it in the installation
Updates	Signed online channel	Signed bundle importable offline
Telemetry	Aggregated, with option per tenant	None leaves the installation except with the institution's explicit authorisation
Base recipes	Recipe catalog (see §6)	Included library + signed offline catalog bundle
Notifications	Email and event webhooks toward the platform, plus the integrations of §13	Email via Exchange Online and alerts to Teams, plus the integrations of §13 the institution configures
Compliance	Framework by market (see §11.7)	In the Dominican Republic: NORTIC. Outside the DR: never NORTIC (see §11.7)

Throughout the manual, when a procedure differs by form, you will see two subsections: **Commercial** and **Government**. When you do not see them, the procedure is identical.

1.3 Principles the user will notice

1. **Fail-closed.** If a signature does not verify, a licence cannot be validated outside its grace, a policy is ambiguous or a channel fails, the system **does not execute** the action and escalates. This is explained in operating language in each module: what stops, what keeps running and what you must do.
2. **What already runs never stops because of a control decision.** Licence expiry, revocation of an agent or blocking of a paid function stop what is new (enrolments, deployments, paid actions); they never shut down running workloads or touch client data.
3. **Absolute white-label.** The console, the catalog, the mobile app and this manual use the SEGRD brand and the brand each installation configures. Third-party names only appear as **your configuration** (for example, the notification channel you connect), never as the product's voice.
4. **No visible secrets.** No screen shows secret values, passwords or keys. Recipes only contain references (`${secrets.<name>}`); the interface shows `••••••` even to administrators.
5. **Everything is recorded.** Enrolments, deployments, convergences, approvals, DRP decisions, accesses and exports are written to the audit register with a hash chain (each entry links to the previous one; altering or deleting an intermediate entry breaks the chain and is detected).

1.4 Roles of whoever uses this manual

Role in the console	What they normally do
Operator	

Role in the console	What they normally do
	Operates day to day: views agents and recipes, deploys, backs up, restores with the corresponding approvals
Approver	Approves DRP executions, policies and debug sessions; may also operate per their assignment
Auditor	Read-only with access to audit, recertifications and evidence export
Read-only	Consults panels and inventory; does not execute actions
Installation administrator	In addition to the above: configurations, report signing, integrations, recertifications and validation

The exact permission matrix and deny-by-default are in §11.1. There is also a vendor support account with fully recorded total access (see §11.3): it is not a role assignable from your console.

---

2. Requirements, installation and agent enrolment

2.1 Agent requirements

- **System:** Linux x86_64 or ARM64. The agent is a single static binary in Go (without interpreter or runtime dependencies on the target server).
- **Compilation (only if you compile it yourself):** Go 1.24 or higher. The resulting binary needs nothing else.
- **Network:** only outbound HTTPS toward the console. **No** inbound port is required. If your network passes through a restrictive corporate proxy that does not admit the persistent channel, the agent uses a fallback mechanism (HTTPS long-poll) with the same message model and lower performance.
- **To manage Docker containers:** access to the engine socket (`/var/run/docker.sock`). Without a socket, the agent degrades honestly: it reports an empty container inventory and keeps working.
- **For Kubernetes:** the agent is deployed as a Deployment with a dedicated service account and minimum permissions scoped to the managed namespaces — never cluster administrator.
- **For hypervisors, tape and backups:** additional requirements per function in §4.4, §7 and §8.

2.2 What enrolment is

Enrolling is registering a server: the console registers the agent, issues it its own mTLS certificate (30-day validity, automatic renewal at 70% of its life, without intervention) and the agent opens its permanent channel. From that moment the console counts a seat for that agent.

There are two registration credentials:

Credential	Recommended use	Behaviour
Single-use enrolment token		

Credential	Recommended use	Behaviour
	Individual registration and sensitive registrations. **This is the recommended option**	Single use, **15-minute** validity, bound to a tenant and a label (e.g. `produccion`). When used it is **consumed**: it cannot be used a second time
Fleet key (reusable onboarding key)	Bulk registrations of a fleet	Admits N uses, with labels and expiry date; it is **revocable with one click** from the console

In addition, enrolment incorporates the **server fingerprint**: machine fingerprint and, if the agent runs in a public cloud, the **cloud identity** detected from its metadata service (provider, instance ID and region/zone). On a local server with no reachable cloud metadata, the "no cloud" result is valid and is not an error. The fingerprint binds the seat to that machine: cloning an agent does not duplicate seats — the console detects the duplicate fingerprint and marks it for review.

2.3 Procedure: enrol a server with a single-use token

1. In the console, go to **Agents** and choose **Generate enrolment token**. Indicate the tenant (if your account sees more than one) and the label. The console shows the token **only once**: copy it at that moment.
2. On the target server, install the agent binary and copy the example configuration to `/etc/abzu/agent.yaml`:

```
sudo mkdir -p /etc/abzu /var/lib/abzu-agent
sudo cp agent.yaml.example /etc/abzu/agent.yaml
sudo chmod 600 /etc/abzu/agent.yaml
```

1. Edit `/etc/abzu/agent.yaml` and complete at minimum:
    1. `console_url`: base URL of your console, without trailing slash.
    2. `agent_name`: readable name of the server (empty = the machine name is used).
    3. `token`: the token from step 1. **This file must remain with permission 0600** and must never be versioned with real values.
    4. `data_dir`: by default `/var/lib/abzu-agent`.
    5. `insecure: false`. The value `true` disables TLS verification and exists **only for local tests** against consoles without a valid certificate. It is prohibited in production and the agent itself marks it as such in code, logs and documentation.
6. Start the agent:

```
# Direct (test):
./abzu-agent --config /etc/abzu/agent.yaml

# As a service:
sudo cp abzu-agent /usr/local/bin/abzu-agent
```

```
sudo cp abzu-agent.service /etc/systemd/system/abzu-agent.service
sudo systemctl daemon-reload
sudo systemctl enable --now abzu-agent
sudo journalctl -u abzu-agent -f
```

1. What happens inside (what you should see):
    1. The agent **generates its key pair on the server itself**. The private key never leaves it.
    2. It calls `POST {console_url}/v1/enroll` presenting the token, the fingerprint and its version, pinning the console certificate fingerprint (pinning): if that certificate changes without an announced rotation, the agent will refuse to connect.
    3. The console validates that the token is current and unused, registers the agent and returns its mTLS client certificate. The token is consumed.
    4. The agent opens its channel with that certificate. From then on, **all** communication is mTLS.
5. Verify in the console that the agent appears in **Agents** with active status and first heartbeat received, and on the server with:

```
abzu-agent status --json --config /etc/abzu/agent.yaml
```

The output shows **real** data read at that moment (containers from the socket, CPU/memory from `/proc`): `agent_id` persisted at enrolment, machine name, version, health and containers. In local demo mode (see §2.5) `agent_id` comes empty.

If something fails: expired token (more than 15 minutes passed) or already used → generate a new one; do not retry with the same one. Certificate error with `insecure: false` → check that `console_url` points to the correct console and that its certificate is valid; do not "solve" the problem by setting `insecure: true` in production. Full detail in §18.

2.4 Procedure: bulk registrations with a fleet key

1. In **Agents**, create a **fleet key**: define labels, number of uses or expiry according to your internal procedure.
2. Distribute the installer with the key through your usual deployment channel. Each server follows steps 2–5 of §2.3, substituting the token with the key.
3. At the end of the registration window, **revoke the key** from the console. Already enrolled agents are not affected: their identity is their certificate, not the key.

2.5 Local demo mode and offline mode

- **Local demo mode:** if the configuration has no token, the console does not respond or the enrolment endpoint does not exist, the agent **does not crash**: it says so clearly in its log and continues writing each heartbeat to the local log instead of sending it. It serves to test inventory without a console. It is not a production mode.

- **Without connection to the console (operational offline mode):** the agent continues applying the **last known desired state** (saved locally), queues jobs and events in persistent local storage and delivers them **in order** on reconnecting. The licence enters the corresponding grace period (see §10) with a visible countdown. When grace is exhausted, new deployments are blocked; what already runs **never stops**.

2.6 Revoke an agent

1. In **Agents**, open the agent sheet and choose **Revoke**.
2. Effect: its certificate enters the revocation list that travels in the heartbeat (less than 60 seconds), its channel closes and its pending jobs freeze. The revoked agent **cannot re-enrol on its own**: it needs a new token.
3. What revocation does **not** do: it does not stop the containers or machines that were already running on that server. Cutting control is not cutting data; removing workloads is a separate, explicit operational decision.

2.7 Agent update

The console announces available versions through the agent's own channel. The agent downloads the update through that channel, **verifies its signature** and the manifest sum before applying it, applies it with atomic replacement and, if the subsequent health check fails, **automatically rolls back** to the previous version. The console can pin a version per agent or per tenant. An update whose signature does not verify is not applied: fail-closed.

---

3. The console: navigation and panels

3.1 Access and sign-in

Commercial. The entry screen is the **platform native login**: you write your email and password **in the console itself**, without being redirected to another login screen. Behind it, the console server validates those credentials against the platform in a server-to-server call. Depending on your account, the same screen may ask you for an additional step without taking you out of it:

Additional step	When it appears	What to do
TOTP code	Your account has a second factor by authenticator	Write the code and continue
Code by email	A new context is detected (IP/ device)	Write the code you received by email
Account selection	Your email has several accounts with access	Choose the account in the selector the console itself shows
Password change required	The platform requires renewing it first	Follow the indication the screen shows; until completed, no session is created

Government. Login with local installation users and, if your institution configured it, access buttons with your institutional provider (Microsoft 365 / Google) or LDAP/AD directory. Local administrator accounts are subject to the rules of your installation (see the honest note in §11.6).

Passkey. If you registered an access key (fingerprint, face or security key), you can enter with it from the same screen. Register the first one from the security module of your profile, already inside the console. The ceremony requires a secure context (HTTPS); clone detection is active: if a key is duplicated, the system detects it and blocks it.

Session. The session has a maximum duration and inactivity timeout according to your installation configuration (in the reference installation: 4 hours in total and 60 minutes of inactivity). The **Sign out** button invalidates the session on the server, not only in the browser. After several consecutive failed attempts, sign-in is temporarily blocked and responds with rate limiting (error 429, see §18): wait the indicated time, do not insist.

Tenant selection. Your account may have scope over one or several tenants (organisations/institutions). Every resource — agents, recipes, secrets, backups, policies — belongs to a tenant and **does not cross** to another: isolation is applied in queries, not only by hiding buttons. A nonexistent tenant identifier shows an honest 404 error; a resource from another tenant is also answered as not found, so as not to reveal its existence (the public API uses this same criterion, §14).

3.2 Console map

Zone	What it is for	Section
Panel (dashboard)	General status in real time: fleet, containers, backups, recent activity	§3.3
Agents	Agent register: status, version, last heartbeat, applied revision; enrolment and revocation	§2, §4
Inventory	Unified view of nodes: containers, pods, VMs and hypervisor containers	§4
Recipes	Versioned desired state: create, edit, diff, rollback, apply	§5
Catalog	Recipes publishable by institutions: publish, review, install, offline bundles	§6
Backups	Policies, restore points, verification, restoration, reports	§7, §8
DRP	Runbooks, v2 autonomy policies, evaluations, vetoes, drills	§9
Licences / Functions	Licence status, quota, paid functions and their status	§10
Federation	Federated sites, cross-site destinations and DR plans between datacenters (if your installation is central)	§15
Validation	Validation runs with their "View evidence" report	§12.3
Evidence / Compliance	Evidence export labelled by framework	§11.7
Recertification	Access review campaigns	§11.5

Zone	What it is for	Section
Integrations	Notification channels, webhook subscriptions and deliveries	§13
Observability	Anomalies, predictions, recommendations and KPIs/KRIs	§16
Audit	Event log with hash chain, exportable	§11.7
Configurations	Installation settings, including **Report signing**	§12.1
Security (profile)	Your access keys (passkeys) and your security context	§3.1, §11
Mobile	The same account in the PWA: status, alerts and approvals	§17

Not all zones appear in all installations: RBAC (§11.1) hides and, above all, **denies** what your role does not cover. That a button is not seen is not the protection; the protection is that the server responds 403 even if the route is invoked directly.

3.3 Real-time panels

The general panel and detail views are fed by the agent heartbeat (every 30 seconds) and console events:

- **Topology and counters:** servers → containers/pods/VMs, with staggered appearance of cards and animated counters. Data refreshes on its own (in the reference installation, the general panel every ~20 seconds).
- **Timeline:** deployments, convergences, backups, restorations and drills as events with their progress and outcome.
- **RPO heat map:** per service, how much data time would be lost today if it failed — the visual input for backup decisions.
- **Presentation mode:** a switch populates the console with animated demonstration data **without touching real data**, for presentations. Deactivate it to return to real data; no presentation-mode action operates on the fleet.

> **Performance and accessibility:** animations use only transforms and opacity, long lists are virtualised and the console respects the system preference to reduce motion (`prefers-reduced-motion`). The console is keyboard-operable; the accessibility target for the government form is level AA (see §11.7).

If a panel shows "no data", it means exactly that: the source has no records. The console **does not invent figures**: a KPI without a source, a catalog aggregate without measurements or a backup without a manifest are presented as explicit absence, never as a disguised zero.

---

4. Agents and unified inventory

4.1 The heartbeat

Every 30 seconds (configurable between 15 and 60) each agent reports: its identity and version, the recipe revision applied per workload, a summary of inventory (containers/services, images, volumes, namespaces — summaries with hashes, not full dumps), node health (CPU, memory, disk, reach to the container engine, depth of its local queue) and, in response, receives the status of its licence. That heartbeat is also the channel through which the revocation list and update announcements travel.

States of an agent in the console: **active**, **degraded** (responds, but something fails — e.g. no reach to the engine), **offline** (does not report; its workloads keep running, see §2.5) and **revoked**.

4.2 Unified inventory

The inventory view brings together in a single table of nodes, with the same operating experience:

- **Docker** containers reported by the agent (name, image, status).
- **Pods** and **Kubernetes** workloads of managed clusters.
- **Virtual machines (VM) and hypervisor containers (LXC)** of connected hypervisors (§4.4), labelled with their hypervisor and host.

Each node shows its origin (which agent or hypervisor reports it). What a source cannot give is shown as absent — for example, the IP addresses of a VM whose guest agent is not installed — instead of being estimated.

4.3 Operations on containers and pods

From the node sheet, the operator can execute the typed operations the recipe and their role allow (apply the desired revision, start/stop/restart where the surface enables it). The agent **does not execute an arbitrary shell**: only operations derived from recipes and typed jobs, against an allowed list. Every operation remains in audit with actor, target and outcome.

4.4 Hypervisors and virtual machines — paid function

VM/LXC management is a **paid function** (its licence status is explained in §10.4; in trial everything works, with a conversion modal that does not block). With it active, the installation discovers and operates VMs/LXC from the following hypervisors. **Current scope:** the management available today is **Proxmox VE**; **VMware, Hyper-V, Xen and generic KVM/libvirt** form part of the committed hypervisor programme (roadmap), and the entries in this table describe the design of each integration:

Hypervisor	How it connects	What it does	Verified limits
Proxmox VE	Cluster REST API (API token or user ticket)	Inventory of QEMU VMs **and LXC**	The power-off action is a **hard shutdown**; the guest system's orderly shutdown is a separate action. The IPs of a QEMU VM depend on it

Hypervisor	How it connects	What it does	Verified limits
		containers, status, power on/off	having a guest agent installed. **Unlike other backup products, here LXC do enter the inventory and the backup (§7.8)**
VMware	**vCenter** API (ESXi is closed: it does not admit an agent on the host)	Basic VM inventory and power on/off by API	No IPs in the basic list (the listing API does not bring them). VMs may also carry an agent inside each VM
Hyper-V	Agent/API on the Windows host (PowerShell)	Inventory and power on/off	**Only on Windows hosts** with the Hyper-V module. On any other system the provider is reported as unavailable with its reason — honest error, never invented data. The node identifier is the VM name
KVM / libvirt (generic)	`virsh` tool on the host	Inventory (`list` / `dominfo` / `domifaddr`) and power on/off	If `virsh` does not exist on the host, the provider is reported as unsupported

General procedure:

1. In the hypervisor sheet, register the connection (API address and credential reference). **Credentials are stored encrypted (AES-256-GCM)** in an agent store of its own, referenced by identifier: never in cleartext in configuration, logs or on screen. The store master key arrives via environment or via a file with permission 0600; a file with more open permissions is rejected.
2. The agent discovers the nodes and presents them in the unified inventory with their type (`vm` / `lxc`), host, hypervisor, status, CPU, memory and IPs when the source gives them.
3. Power actions (on/off) pass through the **entitlement gate**: with the function blocked, inventory continues to be seen but actions are denied with their reason (§10.4). Where an operation is not possible by API on a hypervisor, the console shows it as a limitation; it does not simulate it.

> **Verified limit — Podman:** the agent **detects** Podman sockets (with and without root) and reports it for diagnostic purposes, but the container executor uses a fixed socket and **does not execute on Podman** in this version. Do not present Podman as a supported engine: it is informative detection.

4.5 Agents in the cloud

The same binary runs on AWS, Azure and GCP without opening inbound ports (outbound HTTPS exists in every cloud). At enrolment, cloud metadata detection (§2.2) enriches the fingerprint with provider, instance and region. Where the cloud provides it, cloud volume snapshots can be used as an additional backup destination and secrets can be **referenced** from the cloud secret store without copying their values to the console. Cost labels per workload, when the cloud exposes them, are associated with the corresponding workload.

> **Verified limit:** metadata detection uses short attempts (≤ 1 s per provider, in order AWS → Azure → GCP). In networks with a mandatory proxy toward metadata addresses, detection may not see the cloud and the agent will be treated as local. That empty result is safe: it does not block enrolment.

---

5. Recipes: create, version, reconcile and capture

5.1 The format on one page

A recipe is a YAML `abzu.recipe/v1` that declares the desired state:

```
apiVersion: abzu.recipe/v1
kind: Recipe            # Recipe | Runbook
metadata:
  name: tienda-web      # lowercase, numbers and hyphens; max. 63
  version: 1.4.0        # semantic versioning
  labels: { entorno: produccion }
spec:
  targets: [docker, kubernetes]   # where it can be deployed
  driftPolicy: auto               # auto | manual | report
  secrets:
    - name: db-password           # only the REFERENCE; the value lives encrypted
  services:
    web:
      image: registro.interno/tienda-web:1.4.0   # explicit tag, never implicit
      replicas: 2
      ports: ["8080:80"]
      env:
        DATABASE_URL: "postgres://tienda:${secrets.db-password}@db:5432/tienda"
      depends_on: [db, cache]
      resources: { cpu: "500m", memory: "512Mi" }
      healthcheck: { http: "/salud", interval: 30s }
    db:
      image: postgres:16
      stateful: true              # with data: automatic convergence prohibited
      driftPolicy: manual
      volumes:
        - name: datos-pg
          path: /var/lib/postgresql/data
          backup: { schedule: "0 2 * * *", retentionDays: 7 }
  networks: [tienda-net]
```

Rules the system enforces (they are not advice):

- **No secret in cleartext.** Only references `${secrets.<name>}`, declared in `spec.secrets`. A recipe with a cleartext value is rejected — also in the catalog (§6).
- **Images with an explicit tag** (or digest `image@sha256:…`). No implicit "latest".
- **Deterministic compilation:** the same recipe and version produce byte-for-byte identical artifacts.
- The complete public specification of the format lives in the product documentation (`docs/spec/abzu.recipe-v1.md` of the repository) and is the normative reference for authors.

5.2 Create and version a recipe

1. In **Recipes → New**, write the YAML or start from a one-click template. The editor validates with the engine's strict parser: errors are shown **in Spanish, with the specific field**, before saving.
2. On saving, the recipe is born with its first **revision**. The desired state is git-backed: **each change is a commit** with author, date and message.
3. To change it, edit and save: a **new, immutable revision** is created; the previous one is not rewritten. The console shows the **visual diff** between revisions.
4. **Rollback in one click:** returning to a previous revision creates a new commit that reapplies it — it never rewrites history. What is undone is as audited as what was done.

5.3 Apply a recipe (deployment)

1. Assign the desired revision to a server or cluster and choose **Apply**.
2. Before touching the engine, **admission** can scan the recipe images for vulnerabilities (§11.6): an image in quarantine under the policy **blocks application without touching the engine** and records it. If the scanner is not available, the default policy is **fail-closed** (it is not applied without scanning) unless your installation has knowingly configured otherwise.
3. The job enters the **persistent queue per agent**: with order, retries with growing wait and dead-letter after exhausting failures. Each job carries an identifier and the agent executes it **only once** (idempotence): if there is no acknowledgement, the job is redelivered on reconnecting, not duplicated.
4. The agent compiles/applies: in Docker via the engine API over its socket (never by shell); in Kubernetes with server-side application and its minimum permissions. Deployment logs and metrics arrive in real time to the console through the agent channel.

5.4 Anti-drift reconciliation

In a continuous cycle (every 60 seconds, and instantly through watches in Kubernetes) the agent compares the real state (canonical hash per workload) against the desired state and acts according to policy:

Policy	Behaviour	When it is used
`auto`	Converges on its own to the desired state and records it	By default in services **without** state

Policy	Behaviour	When it is used
`manual`	Alerts in the console and **waits for approval**	**Mandatory in everything that has data** (`stateful: true`): the agent never touches data on its own, even if the recipe says otherwise
`report`	Only records the deviation	Audits or change freezes

Every convergence or deviation remains in audit with the before and the after. A drift in a critical workload can also generate an actionable alert in the mobile app (§17) and in your integrations (§13).

5.5 Reverse capture: from reality to the recipe

When a change was made **outside** the console (the classic 3 a.m. emergency fix), reverse capture prevents it from remaining as eternal, invisible drift:

1. In the agent or workload sheet, choose **Capture real state** (`recipe capture` ; you can scope it by containers or capture all).
2. The agent photographs the real state — live containers/manifests, **non-secret** variables, volumes — and generates a **draft revision** of the recipe. Secrets appear only as masked references.
3. A human reviews the draft (diff against the current revision) and decides to **promote it** to desired state or discard it. Without approval, the draft changes nothing.

Reverse capture is also the **migration engine** for existing installations: it converts workloads that already run into versioned recipes, which is what makes DRP trustworthy (§9): the recipe once again represents reality.

5.6 Recipe secrets

- At rest: encrypted per installation. In transit: only through the mTLS channel and already encrypted. In use: the agent decrypts them **in memory** with its identity key and injects them; they are never written to disk in cleartext nor do they appear in logs or screens.
- **Rotating a secret** = creating a new revision of the recipe that references it; the agent applies it with a rolling restart of the affected services.
- Not even an administrator can view a value from the console: the interface and the API always mask it.

---

6. Recipe catalog

The catalog is the mechanism for an institution to **publish** a polished recipe and for others to **install** it with guarantees. It is not a star shop: its signals are operational (§6.4).

6.1 Publish a recipe

1. In **Catalog → New entry**, provide: publisher (your stable identifier), name and semantic version of the publication, changelog in clear text, the complete YAML of the recipe and the **declared**

compatibility: targets (`docker` / `kubernetes`), database engines it supports and minimum agent version.

2. Each change publishes a **new, immutable entry** with its own version; an already published entry is not edited.
3. The entry is **signed with the publisher's key** (ed25519 digital signature over canonical content that includes name, version, the YAML fingerprint, the changelog fingerprint and compatibility). The signature also covers compatibility: nobody can alter it without invalidating it.

> **The publisher is your key.** Your public identity is the fingerprint (SHA-256) of your public key, which must be pinned through independent channels so other installations recognise it. Keep your private key as you would keep any signing key: without it you will not be able to publish new versions as that publisher.

6.2 Review: from draft to approved

```
draft ──submit──► automatic security review
                        ├─ missing or invalid signature ──► rejected (reason
recorded)
                        ├─ secrets in cleartext ──────────► rejected (reason
recorded)
                        └─ images registered for agent admission
                                  │
                                  ▼
                             under review ──► human approval or rejection WITH
REASON
```

- All states and reasons are **persisted and audited**.
- Vulnerability scanning of images also occurs in the **agent on applying** (§5.3): an approved entry with a quarantined image **is not installed**.
- Only an entry that is **approved + with verified signature + without secrets in cleartext** is installable.

6.3 Install from the catalog

1. In **Catalog**, open the approved entry and choose **Install**, indicating the destination.
2. **Compatibility is applied, not only shown**: if the destination is not among the declared targets, the DB engine does not match or the agent version is lower than the declared minimum, the console **blocks the download** and responds with the reason, without delivering the YAML.
3. The agent repeats verification **before** installing (installation gate, fail-closed), in this conceptual order: valid structure and versioning → **valid publisher signature against the trusted keys configured in that agent** (no signature, invalid signature, untrusted publisher or **zero trusted keys configured** = rejection) → no secrets in cleartext → compatibility. Only after that gate does installation follow the normal recipe flow (§5.3), including image admission.

4. Configure in each agent the catalog trusted keys directory (`catalog_trusted_keys_dir` in `agent.yaml` , one public key file per publisher). **Empty means that agent installs nothing from the catalog.** It is deliberate.

6.4 Operational signals (honest ratings)

Per entry, the catalog can show **operational, anonymised** aggregates: active installations, rate of successful restore drills and last verification. If an installation has no real data for an aggregate, it shows **"no data"**. Publishing figures without measuring is prohibited by the catalog specification.

6.5 Offline catalog bundle (government)

For installations without a network:

1. **Export:** in **Catalog → Export**, the console generates a **signed bundle** (format `abzu-catalog-bundle/v1` , a transportable file) with the publishers (and their keys) and the entries with their individual signature.
2. Transport the bundle through your approved channel to the isolated installation, together with — **through a separate channel** — the fingerprint of the bundle issuer's key.
3. **Import:** in **Catalog → Import**, import is **fail-closed and atomic**: (1) the issuer fingerprint must match the one pinned through the separate channel; (2) the bundle signature must verify; (3) the signature of **each** entry must verify against its publisher and none may contain secrets in cleartext. **Any failure = nothing is imported.**

---

7. Backups

7.1 General model

The backup module lives in the **agent**; the console schedules and displays, and **never touches database credentials**. Principles that apply to everything in this chapter:

- **Every backup travels in stream, without temporary files on the host disk:** native tool → **AES-256-GCM encryption in stream** → upload in parts to the destination. The memory it uses is bounded to one part plus one encryption block.
- **Manifest per backup:** next to each object its JSON manifest is kept with engine and version, database, times, sizes (encrypted and cleartext), SHA-256 fingerprint of the encrypted artifact, format, row count per table and, after verification, the **measured restore time** (its real RTO).
- **The recipe declares what to back up, never with what key.** Credentials are resolved by the agent inside the database container itself, from its environment.
- **Non-disableable encryption in the government form:** also for volume snapshots (§7.7). A backup marked as compromised **is marked and alerted; it is never mutated or deleted** (§7.6).

7.2 Database engines

Engine	Tool	Format	Incrementals / PITR
PostgreSQL	`pg_dump -Fc` (custom format) executed inside the DB container	`pgdump-custom`	Base with physical copy + archived WAL segments (see §7.4)
MySQL / MariaDB	`mysqldump --single-transaction` with routines, triggers and events	`mysqldump`	Base with binlog position captured + closed binlogs (see §7.4)
SQLite	WAL consistency check + dump (`.dump`)	`sqlite-dump`	Always full (real SQLites are small; incremental adds nothing)
SQL Server (MSSQL)	`BACKUP DATABASE` / `BACKUP LOG` via the container's own tool	`mssql-bak` / `mssql-trn`	Log backups = the native incremental; restoration with recovery (see §7.4)

Tool resolution follows this order: (1) inside the DB container, if it carries the client (official images do); (2) auxiliary tools container; (3) host binary. Detection confirms the engine version and chooses compatible options. In the recipe, the database is declared like this:

```
databases:
  - name: lbc_db
    engine: postgres
    container: lbc_db
    schedule: "0 2 * * *"
    retention: { daily: 7, weekly: 4, monthly: 6 }
    destination: repositorio-objetos
    verify:
      sandbox: true
      portals: ["https://app.cliente.com/health"]
```

7.3 Create a backup policy

1. In **Backups → Policies**, create the policy: workload/database, engine, destination, schedule (cron expression) and **GFS retention** (grandfather-father-son: dailies, weeklies, monthlies and annuals retained).
2. **Sealed GFS point:** a point created on a Sunday, day 1 or 1 January is classified and **sealed** until its full period expires: short retention **does not prune it earlier**, even if it would be due by days. If the destination has immutability (§7.5), the seal rests on it.
3. **Backup window:** declare the allowed window (e.g. `22:00-06:00` ; allows crossing midnight). Outside the window, the job **does not start**: it waits and reports "waiting for window". Services operating 24/7 are not saturated by an out-of-hours backup.

4. **"Back up now":** from the workload sheet, trigger an immediate backup without waiting for the schedule (it is the manual way to force the job, equivalent to skipping the window explicitly and with audit).
5. **Chaining:** a declarative chain can order the steps of a single run — backup → copy to second destination → verification → integrity check — with failure propagation: if a step fails, the run reports it and does not fake success for the following ones.
6. The policy is part of the desired state: if someone changes the schedule by hand, reconciliation restores it and audits it (anti-drift also for backups).

7.4 Incrementals and point-in-time recovery (PITR)

On the same driver interface of §7.2, without changing it:

Engine	Base	Incremental	Restore to an instant
PostgreSQL	Physical base copy with WAL in stream	WAL segments archived after forcing their rotation. Requires in the DB: archiving enabled and sufficient WAL level; if not met, the job says so and does not promise incrementals	In sandbox: base data + WAL + target by **date/time or by position (LSN)**; on reaching it, the engine is promoted
MySQL/ MariaDB	Dump with the binlog position captured from the stream itself	Closed binlogs not covered, after rotating them	Restore the base and replay binlogs up to the requested **date/time or position**
SQL Server	Full `BACKUP DATABASE` (guarantees full recovery model) and capture of its LSN	`BACKUP LOG`	Restore the base without recovering, apply logs up to the instant (`STOPAT`) and close with recovery
SQLite	—	— (always full)	To the last backup

PITR procedure from the console:

1. In **Backups**, open the workload and its **chain**: you will see the base and the chained incrementals (each link is a normal manifest with its parent and the chain block: type, chain identifier, sequence and position per engine).
2. Choose **Restore to an instant** and indicate the target date/time (and the position/LSN if your procedure uses it).
3. The system restores **first in sandbox** (§7.6) and, if you provide a **probe** — a read-only query —, executes it in the sandbox and shows you the rows that existed at that instant: it is the evidence that the chosen point is the correct one, before touching production.
4. **Chain-aware retention:** GFS pruning never leaves a broken chain: if a link is retained, its base and all intermediate incrementals are retained; a chain with no retained descendants is pruned whole.

7.5 Destinations, second copy and immutability (3-2-1 rule)

A backup with a single destination meets no serious continuity rule. The module works with interchangeable destinations — jobs, manifests and retention do not know the concrete type:

- **S3-compatible object:** the usual destination.
- **Local file / offline medium:** a directory on a disconnectable medium = the **offline / air-gap** copy against ransomware.
- **Tape:** see §8.
- **Federated site:** the storage of another federation site (§15).

Second copy (backup copy). Configure in the policy a second destination, immediate (when the point is created) or scheduled. The copy replicates artifact + manifest **in stream, verifying the fingerprint during the copy**: if the hash does not match, the copy is aborted and the partial is deleted — a corrupt "copy" with the appearance of a good one never remains. The copy is recorded in the manifest, with its destination and its own retention.

Immutability (Object Lock). So that not even an attacker with agent credentials can delete a sealed point:

1. The destination bucket must be **born** with Object Lock enabled (it cannot be enabled afterwards on an existing bucket). The service creates it that way when the policy requests it.
2. On writing the point, retention is applied to it: **GOVERNANCE mode by default**; **COMPLIANCE configurable** for critical points. The retention date is the end of the point's GFS seal, or the period the policy sets.
3. **Read this before choosing a mode:** in a bucket with versioning, a simple deletion only creates a delete marker — the retained version remains readable and its fingerprint matches the manifest — and **permanent deletion of the protected version fails** ("the object is WORM-protected"). But GOVERNANCE **does admit an exception** for credentials with a special bypass permission (e.g. the storage administrator). That is why the mandatory pattern is the **hardened repository**:

- Bucket **dedicated per agent/tenant**, created with Object Lock.
- Credentials **single-use per agent**, with minimum permission only over their bucket (put/read/list) and **without** bypass or administration permission. The storage root credentials are **never** used by an agent.
- Object Lock retention ≥ GFS retention of the sealed point.
- The second copy lives in another site/bucket with its own credentials: compromising an agent does not expose the copy.
- For critical sealed points, use **COMPLIANCE**, which admits no bypass.

Tiering (levels). An age policy can move or copy old points from the hot destination to the cold one: copy mode (they remain in both) or move mode (deleted from hot **only after re-reading the object in cold and verifying its fingerprint**). Sealed points or points with current Object Lock are **skipped and reported**, not forced.

7.6 Sandbox verification and "View evidence"

A backup that is never restored is a promise, not a backup. Abzu verification truly restores:

1. The decrypted content is scanned with the antivirus available on the agent host. If the scanner is not there, the report says so (`skipped`), without pretending it was scanned. If it detects malware: the point is marked **compromised** (§7.6.1), an alert is raised and the original **is not deleted or mutated**.
2. A **disposable sandbox with no network toward production** is raised with the source container image.
3. It waits for the sandbox engine to respond healthy.
4. The artifact is restored and **rows are counted table by table**: equality against the manifest must be **exact**.
5. The time from when the sandbox starts until the verified count is measured: that is the **measured RTO** that remains in the manifest.
6. In extended verification the following are added: HTTP tests on the portals declared in the recipe (expected code, content and latency), **real screenshot** of the portal and the automatic health report that accompanies the backup.
7. The sandbox is always destroyed, whatever happens.

SQLite verifies by restoring into a temporary file on the host (its natural sandbox: no network or container) and deleting it on finishing.

Verification as policy. Verification is scheduled **per workload** (frequency and scope: restoration + counts, HTTP, antivirus), not only as a global drill. Its result feeds the "View evidence" report for that run (§12.3) and the verified restores KPI (§16).

Scheduled integrity check. In addition to restoring, a periodic job **re-reads and re-hashes** each artifact in stream and contrasts it with its manifest, without modifying anything: it reports each point as intact, corrupt or absent. It is the cheap complement that detects silent corruption over time.

Periodic backup report. From the manifests, the console generates per tenant/workload the coverage report: last backup and its status, RPO met against the objective, infected points, copies and integrity of the last check, in readable format (Markdown/JSON). It is the periodic evidence management and auditors request (see §11.7).

7.6.1 Infected points and safe restoration

- If a verification marks a point as **infected**, the status remains in its manifest and **propagates to all its destinations/copies**. The console shows it and DRP respects it.
- Restoring an infected point is **blocked by default**: it requires explicit risk approval (in the agent operation, the corresponding express confirmation). The system records who accepted that risk.
- Before restoring to production, **safe restoration** repeats the pre-check on the chosen point just before executing it — verification may have run days ago —: integrity against the manifest and re-scan. If the pre-check fails, it is not restored.

7.7 Volumes, host files and global point catalog

- **Volume snapshots:** the content of a volume (or of any directory) is archived in stream, **always encrypted** (in government it is not disableable) and uploaded with its manifest, which includes the **per-file index** (path, size and fingerprint of each file in cleartext).
- **Restore a single file:** from the manifest, choose the file and **Restore file**: the system traverses the decrypted stream and extracts **only that file**, verifying its fingerprint, without restoring the whole volume or downloading the artifact to disk. Honest cost note: the traversal is sequential, so a file at the end of a huge snapshot takes proportionally.
- **Backup of host paths:** declare a list of paths (files/directories of the host or of a shared resource) and they are backed up with the same pipeline and the same per-file restore.
- **Global point catalog:** all manifests (databases and volumes, from all destinations) are indexed in a catalog searchable by workload, dates, status (`ok` / `infected` / `sealed`), engine and type. From the catalog you reach the concrete point — and, if it lives on tape, which cartridge to mount (§8).

7.8 Backup of hypervisor VMs and of the console itself

- **Proxmox VMs (QEMU and LXC):** with the hypervisor function active (§4.4), Proxmox VMs and LXC containers are protected via its API (snapshot + export). The **LXC included** are a real differentiator against products that only back up VMs.
- **VMs of other hypervisors with CBT** (VMware/Hyper-V): classified as a later iteration (v1.1 in gap planning); meanwhile, a legacy VM without a recipe is protected with the **agent inside the VM** (its databases via §7.2 and its paths via §7.7). Do not promise CBT where the code does not carry it.
- **The console itself backs itself up** with the same engine (its database, configuration and recipes) and can be restored from that backup. Without this, the product's DRP would be theatre.
- **Bandwidth limit:** with an off-site destination, the agent can limit upload speed by schedule (limitation by token bucket in the uploader), so a backup does not saturate production.

7.9 Continuous backup (CDC)

For workloads whose target RPO is **minutes or seconds**, continuous backup sends **log segments** (WAL in PostgreSQL, binlogs in MySQL/MariaDB) as they close, instead of waiting for the next scheduled incremental:

1. The shipping service discovers new segments from the last confirmed position.
2. It uploads each segment **in order** to the destination (CDC does not know the transport: object, file or tape).
3. It **verifies** the upload by re-reading the object and comparing its fingerprint.
4. **Only after verifying does it confirm the position** (persisted atomically, segment by segment). A failure leaves the position at the last verified segment: on retrying, the current segment is re-uploaded, **without gaps**.

5. The metric that governs is the **lag**: seconds between "now" and the end of the last confirmed segment = your **real RPO in seconds**. If the shipping service falls, RPO grows silently: **monitor and alert the lag**.

CDC does not restore: it guarantees the links. Each segment corresponds to a link in the §7.4 chain, and PITR is the same — the system validates that base + segments form an ordered, restorable chain; if the chain is broken, **it says so** instead of promising an impossible PITR.

For which workloads yes: transactional databases where losing 15–60 minutes costs more than operating the loop (payments, orders, inventories) and with moderate log volume — a large database with little writing is the ideal case. **For which no:** massive batch writing (the log grows like the data and CDC degenerates into copying everything, continuously), small or low-criticality databases with an acceptable 1-hour RPO (incrementals suffice), and environments with nobody to attend the lag alert.

> **Verified limit — CDC:** outside this scope remain WAL streaming via the replication protocol and row-by-row capture of the logical replication type: they are another product, not continuous backup. And the end-to-end test against real engines (WAL archiving of a live PostgreSQL and binlog rotation of a live MySQL, with timed PITR) **remains pending an environment with both engines**: current tests are unit tests with synthetic segments and a local file destination (discovery, order, confirmation only after verified upload, resumption without gaps after an injected failure and lag calculation). Do not present CDC as verified against live engines until that test exists.

---

8. LTO tape

8.1 What there is and what there is not

- **Committed roadmap (v1.1 line):** backup to tape **basic with generic driver**: Linux tape server, simple pools, simple tape GFS, catalog by barcode and restoration with staging to disk.
- **Committed roadmap (v2):** the complete infrastructure built **on top of** the above: GFS pools with the 5 media sets, vaulting with custody and evidence, multistream and WORM pools (§8.4).
- Tape enters as **one more destination** (§7.5): jobs, manifests and retention do not know the concrete destination type.

8.2 The generic driver

The agent speaks to the tape through a single, sequential interface (load/unload by barcode or by slot, rewind, search by file marks, write/read at the current position, erase, changer status and inventory):

- With **real hardware** it uses the system's standard tools: the drive (`/dev/nst*`) and the changer (`/dev/sg*`). Without hardware, it reports it with its reason: **it never invents a unit**.
- With the **virtual library on disk** (a directory = a library; each cartridge = a subdirectory with its barcode, its data and its marks index) development, testing and demonstration are done **without hardware**, with **the same state machine** as real hardware: writing or reading without loading =

error; loading a second cartridge without unloading = error; loading a nonexistent barcode = error; searching out of range = error; writing after rewinding truncates the rest.

Pool, catalog and retention (simple):

- The policy assigns a cartridge or a set of barcodes (the pool). If the loaded cartridge **is not from the pool, writing fails loudly**: nothing about writing to the wrong cartridge in silence. Reading does change cartridge: the catalog governs.
- The **catalog by barcode** (persisted) records for each object: key, barcode, file number on the tape, size, fingerprint, write date and retention end. **It is critical data**: without a catalog, tape is an irrecoverable black box — back it up as such.
- **Simple retention:** overwriting a key with current retention = error; **recycling** a cartridge (the only "deletion" that exists on a sequential medium) with current points = error. There is no individual deletion of an object on tape, by the nature of the medium.
- **Restoring from tape** = reading back to the requested destination (staging to disk first), verifiable by fingerprint against the catalog.

8.3 When the generic is not enough: custom driver via secure channel

On detecting the unit, the agent classifies it:

- The **generic suffices** if all three are met: (1) the drive exposes itself as a standard SCSI tape; (2) the changer, if it exists, exposes itself as a standard changer operable with system tools; (3) the unit is not in the table of those requiring manufacturer-specific commands.
- An **uncatalogued** unit is not automatically declared "custom": it remains as **"requires evaluation"**.
- If a unit requires its own driver, the console presents the notice **"A driver is required for your unit"** and the flow is:

1. **Requirement (what travels to the vendor):** the agent generates the requirement file with **only unit data**: type, manufacturer, model, path and reason; the serial number always goes **redacted/partial** (first and last characters, the rest hidden). No credentials, no client data, no backup content. The file carries its **fingerprint**, which allows detecting any subsequent alteration. If no unit requires a custom driver, the command **does not generate an empty requirement**: it says so and ends with an error.
2. **Construction and delivery:** the vendor builds the driver for that unit and delivers it as a **signed package** through the **secure channel**: a file with the driver manifest (name, version, compatible units and the fingerprint of **each** file) and its separate **digital signature**.
3. **Verified installation — fail-closed, in this order and all mandatory:**
    1. The **vendor's public key** configured or embedded in the agent must exist. Without a key → rejection.
    2. Valid package structure (only regular files, no escaping paths, no duplicates, with manifest and signature).
    3. **Valid signature** over the exact bytes of the manifest. If not → rejection.

4. **Exact fingerprints:** the file set is exactly the declared one and each fingerprint/size matches. If not → rejection.
5. **Compatibility:** some compatibility label covers the unit detected on that unit. If not → rejection.
6. Record: files are copied to the local driver registry, the index is updated and the attempt is noted in its local audit register.

7. Bad signature, altered hash or incompatible unit = **total rejection**: nothing is installed, not even partially, and the attempt is recorded with its reason. Installed drivers are listed, their attempts consulted and they can be **removed** — all recorded.

The agent's tape status is always exposed in its status snapshot (`tape`): availability and reason, units detected with their classification, loaded barcode, whether a custom driver is required (with its note for the console), installed drivers and pending requirements. Without hardware, the sheet shows the honest reason ("no tape hardware on this host") instead of zeros.

> **Verified limit — real hardware:** validation against a **physical LTO library** (real load/search choreography, times, drive cleaning, physical barcodes) is **pending availability of the library**. The hardware code is tested with a fake executor and with the real parser of changer output, and end-to-end with real objects was verified against the virtual library. Do not promise support for any concrete model beyond the classification in the unit table until that validation exists.

8.4 Complete tape infrastructure (v2)

Built on §8.2–§8.3, without rewriting them:

- **GFS pools with 5 media sets:** each day is classified into its set — the highest level governs: 1 January → **annual**; day 1 of quarter → **quarterly**; day 1 of month → **monthly**; Sunday → **weekly**; rest → **daily**. Default retentions: dailies 7 days, weeklies 5 weeks, monthlies 12 months, quarterlies 2 years, annuals 7 years. The manager registers media by barcode, assigns the set's medium on writing, extends the medium's retention with each point, seals full media and locates each point (which barcode to mount and in what position).
- **Vaulting with custody and evidence:** exporting a medium requires destination and actor, marks it as **exported** (unrecyclable and unassignable while it is out) and records the movement in a **custody register append-only with hash chain**: who hands over and who receives, when, which barcodes and to what destination. Return and verification in custody are movements of the same register. A manipulated register is **rejected on opening**: custody is evidence, not a note.
- **Multistream:** several parallel streams, one per drive, each with its own manifest (drive, barcode, position, size, fingerprint) and **verification on close** (re-read + fingerprint before deeming the stream good). If one stream fails, the job returns an error and its manifest reports it, but the other streams remain intact and verified: there is no shared state between drives.
- **WORM pools:** with WORM active, retention is **absolute and fail-closed**: recycling or reusing a medium before its retention expires is rejected **even when forcing it**. In non-WORM pools, forcing is the operator's deliberate path to recycle early (recorded human decision); without forcing, retention is also respected.

- **Traceability of verification:** a point that lives on tape is a normal manifest, so the same sandbox verification of §7.6 covers it without new code: the operator resolves the artifact from tape (locate the point → mount the barcode → read) and verifies the same as any other destination.

---

9. Disaster recovery (DRP)

9.1 The runbook: rebuilding is re-applying

A runbook is a special versioned recipe (`kind: Runbook`) that describes the canonical reconstruction order:

1. **Network:** networks and namespaces, with their policies.
2. **Volumes:** create empty volumes with the same names and classes.
3. **Data:** restore volume snapshots and, for databases, restore from the backup module (§7) choosing the point — including a PITR point (§7.4).
4. **Applications:** deploy services in topological order according to their dependencies, waiting for each layer to respond healthy before continuing.

As the recipe always reflects reality (reconciliation + reverse capture, §5), recovering is not improvising: it is re-applying. Design objectives by tier:

Tier	Target RPO	Target RTO	Notes
Commercial Essential	24 h	4 h	Daily snapshots, assisted manual restore
Commercial Professional	1 h	1 h	Hourly snapshots, semi-automatic runbook
Government	Contractual (design: 15 min)	Contractual (design: 30 min)	Semi-automatic or autonomous by policy (§9.3) + evidenced monthly drills

Without a promise of zero loss: the design offers RPO through frequent snapshots and through CDC (§7.9), not synchronous replication.

9.2 Semi-automatic execution (v1/v1.2 baseline)

Detection and preparation are automatic (most recent verified point, destination checks, ordered plan). **Execution requires human authorisation**, through two channels:

- **Console: Execute DRP** button with double confirmation. Who authorised is recorded.
- **Voice call:** if nobody responds in the console, the system **calls the on-call responsible** (pre-registered numbers): the voice agent explains what was detected and what will be recovered, and requests authorisation by voice ("say I authorise") or by keypad (key 1).

Non-negotiable security rules for voice:

- The call alone **does not suffice**: pre-registered number **+ PIN** spoken or by keypad.
- The recording and transcription remain in the audit register as evidence.

- The webhook that delivers the call result verifies its **signature in fail-closed mode**: without a valid signature, no authorisation is accepted. (This was a known breach of the inherited platform — an unsigned webhook that accepted openly — and is recorded as a requirement to verify before using voice for real DRP.)
- If nobody in the chain answers after the configured attempts: **nothing with data is executed**; it escalates and alerts through all channels (console, Teams, email, SMS and the integrations of §13).

Separation of duties (SoD): whoever configured a runbook **cannot be its sole approver**. Execution requires an approver distinct from the configurator.

9.3 Autonomous policies v2: governed autonomy

In v2, each workload or tier can declare its **autonomy level**. The human ceases to be the 3 a.m. bottleneck and becomes the **legislator**: they define, sign and can revoke the rules within which the system decides on its own.

Level	Behaviour
`manual`	Like the baseline: nothing executes without human approval
`deferred approval`	It executes if **nobody vetoes it** within the configured window (N minutes). The veto is exercised from the console or from mobile (§17)
`bounded autonomous`	It executes without a human, **only** within the policy guardrails

The ceiling is set by the installation, not the recipe. The effective level is the lesser of the policy level and the maximum allowed by the installation. Without installation guardrail configuration, start-up is `manual` (fail-closed).

9.3.1 Create and activate an autonomous policy

1. In **DRP → Policies**, create the policy: workloads covered (**closed scope**: outside that list, the policy does not exist), level, local time window, maximum acceptable RPO before firing, veto window (if applicable) and the prior snapshot requirement.
2. Saving creates a **new version in draft, without signatures**. Editing an active policy does the same: the current version is not touched until the new one is activated.
3. **Activation with dual signature (strengthened SoD):** activating requires the signature of an administrator **distinct** from whoever created it. Without a second signature, the policy remains an inactive draft and the evaluator treats it as **nonexistent**.
4. The activated policy, its ceiling and the kill switch remain in audit with a hash chain.

9.3.2 Guardrails: what the system checks before acting alone

Before any autonomous action, the evaluator verifies **all** guardrails. It suffices for one to fail for the case to fall to manual (or to rollback, in subsequent verification):

Guardrail	If it fails
Installation **kill switch** activated	Manual: zero autonomy across the whole installation
Policy not active or without valid dual signature	Manual
Level requested above the installation ceiling	It is **capped** to the ceiling (it is not rejected: it degrades to the allowed level)
Workload outside the policy's closed scope	Manual
Outside the local time window	Manual
Observed RPO greater than the policy maximum, or **unknown**	Manual
Prior snapshot absent (when the policy requires it)	Manual
Post-restoration verification failed or **absent**	Automatic **rollback**

Each decision is persisted with the result **per guardrail** and hash-chained: afterwards it can be reconstructed why the system acted, waited or did not act.

Double verification of the decision. The console decides (there live the signed policy, the ceiling, the kill switch, the veto window and the audit) and the **agent re-verifies** the autonomous order with the same pure function before executing: if its re-verification does not reproduce "execute", the agent **refuses** and the case falls to manual. Console and agent must agree for something to execute without humans.

Veto window. In `deferred approval`, evaluation remains waiting until the end of the window, as persisted state: vetoing within the window stops it; expired without a veto, it is released to execute.

Post-verification and rollback. After restoring, the system requires **double verification**: health **and** consistency. If either fails — or is not recorded — evaluation is marked as "rollback required" and rollback is triggered. An absent verification counts as failed: fail-closed.

Kill switch. In **DRP → Installation guardrails**, an administrator can activate the global switch: immediate effect, zero autonomous actions across the whole installation until deactivated. Manual flows are not affected. Activating and deactivating it remains in audit.

9.4 Drills and their report

- **Scheduled drill** (monthly by default, configurable): the system raises a sandbox environment, restores the last copy of each workload marked for DRP, runs its health checks and issues the **drill report**: success/failure per workload and times measured against the tier RTO/RPO. A restore that is never tested is a restore that does not exist: the report is its auditable evidence.
- **Drill under the autonomous policy (v2):** the drill runs through the same pipeline (sandbox, without touching production) and its report adds **what the policy would have done** given the same signal versus what was executed, with a divergence mark if they differ. Thus autonomy is continuously tested and does not debut in the real disaster.

- Drill reports belong to the family of signable reports with "View evidence" (§12).

---

10. Licensing and paid VM function

10.1 The model on one page

- **Unit = agent.** Each enrolled agent consumes a quota; containers per agent are unlimited.
- **Assignment governs, and lives outside Abzu.** Which account/tenant has which platform, plan, quota and functions is governed by the vendor's **licence and platform panel**. Abzu **does not create a parallel assignment register**: it validates against what is assigned there.
- **The technical validation point** is the licensing service: automatic registration of installations, quota heartbeat, issuance fed by the vendor's assignment and signed offline licences for government. Its web portal retains purchases and reports and shows status by installation identifier (read-only view; sensitive operations require the installation token, never the identifier alone).
- **Soft block, always.** A trial or subscription expired outside its grace, or quota exhausted: no new servers enter nor is anything new deployed, but **nothing already running is shut down or touched**.

10.2 Zero-touch registration and trial

1. On being installed, the installation **registers itself** (without vendor intervention): it provides name, edition (`comercial` / `gobierno`), contact and its fingerprint, and receives its installation identifier and a **token delivered only once** (the service keeps only its fingerprint; present it at each heartbeat as a credential).
2. Registration opens a **30-day / 3-server trial**, without a card.
3. At each heartbeat, the installation reports its active servers. Response: status, quota, used/available, remaining days and whether it can add servers and deploy.
4. **Server that stops reporting:** its quota is released after **7 days of grace**. If it reports again within grace, it is reactivated and grace is cancelled.
5. **Quota exhausted:** what exists continues; new servers are rejected **softly** (they remain listed as rejected in the response, with the rest operating) until quota is expanded.
6. **Expired subscription:** 7 days of licence grace and afterwards expired status — always a soft block. The **perpetual licence never expires** (remaining days = no expiry); expiry of optional support does **not** block it.

Licence states you will see in the console and in the portal: `active` (current) · `trial` (in trial) · `grace` (expired ≤ 7 days ago) · `soft_blocked` (trial expired) · `expired` (subscription expired outside grace).

10.3 Request expansion and how it is activated

1. From the console or the portal, register a **purchase/expansion request** (type, quota requested, contact, notes). The request **is only registered: it does not charge** nor request payment data.
2. Activation arrives when payment (commercial) or award (government) is confirmed in the vendor's panel: its confirmation feeds technical issuance — through the governed internal path or through the payment webhook with **fail-closed HMAC signature** and **idempotent** (a duplicate event does not re-issue) — and the licence is issued without manual intervention.
3. Without a webhook secret configured, that endpoint remains **disabled** (not "open"): fail-closed also in issuance.

10.4 Paid VM/hypervisor function

VM/hypervisor management (§4.4) is activated only if the licence **assigned by the vendor** covers it (function `vm_hypervisor`). The console validates that assignment against the technical point and the agent reapplies the gate on actions. States you will see:

State	What you see and what you can do
Available (`entitled`)	The assignment covers the function and the licence is current (or in licence grace: what exists continues, marked as degraded). Discovery and actions allowed. In the government pilot reference installation, the function comes out **available** because its assignment covers it — not because of any special treatment by name: assignment governs
Trial (`trial`)	**Everything works, VMs too.** The console shows the **conversion modal**: it informs that the function is paid and offers to activate it, **without blocking its use**. The trial has no walls; conversion is earned by showing the complete product
No function (`none`) — soft block	The section **is seen** (with its paid-function notice), but **actions are denied** (on web: redirection with notice; on API: `403` with required entitlement code). The rest of the product continues operating; client data is never touched
Validation impossible	Fail-closed **with defined grace**: within grace the last known state is retained, marked as degraded; outside grace or without a valid saved snapshot, the function remains at "no function". The agent applies the same criterion with its own cache and its entitlement grace

Government pays full price. In the government edition there is no trial price or discount for this function: its conversion is always at **full price** (perpetual per agent + annual support), from award. In commercial, trial converts by activation and absence of function, by purchase.

10.5 Offline licence (government without a network)

For installations without internet egress, the licence travels as a **signed file** (ed25519 digital signature):

1. The issuer generates the file for **that** installation: licence identifier, installation identifier, holder, edition, type (`perpetua` / `suscripcion`), server quota, issue dates, expiry (empty in perpetual: **never expires**) and support end.
2. The installation imports it from the console. The console **verifies locally, without a network**: valid signature against the issuer's public key it carries embedded, installation identifier matching its own, and validity. Any alteration of the file (quota, identifier, type) **invalidates the signature** and is rejected: fail-closed on invalid signature, different algorithm, unsupported version, invalid quota or subscription without expiry or already expired.
3. The console revalidates the file **on start-up and every 24 hours**, always without external calls. The issuer's private key is **never** distributed.

10.6 Federated licensing

In a federation (§15), counting aggregates toward the licensing service: each installation produces **signed reports** of its use and the central aggregates them — in government, as a **signed offline aggregate**, verifiable with public keys even if the central is in failover. The licensing portal also gains federated self-service on the **same** vendor assignment: quota purchase per site, aggregated reports per institution and capacity alerts before exhaustion. Never on a parallel register.

---

11. Security, roles and compliance

11.1 Fine-grained RBAC with deny-by-default

Roles: **operator**, **approver**, **auditor** and **read-only**. Each endpoint of the console and of the internal API is mapped to an **action** (`view`, `operate`, `approve`, `audit`, `destroy`, `debug`) and each action to the roles that can execute it. Denial is by default at **three levels**:

1. An endpoint **without an explicit action** in the map → **403**.
2. A user **without a resolvable role** → **403**.
3. An unknown action → **403**.

Every denial remains in audit. Role resolution follows this order: explicit user role → role of their SSO/LDAP session → assignment by email configured in the installation → inherited map of their team. An invalid explicit role **denies**; it does not silently degrade to a lesser role.

Some usage references: view = the four roles per their scope; operate = operator and approver; approve = approver; audit and export evidence = auditor (and administrator); destroy and debug = only with the additional controls of §11.2 and §11.4.

11.2 Adaptive authentication (step-up)

On **anomalous context**, the console requires **step-up re-authentication** before sensitive actions. Signals (any one triggers; each contributes to the risk score that is recorded): **new IP**, **new device** (browser fingerprint) and **unusual time** (outside the installation's configured window; by default 06:00–22:00 in the installation's reference time zone).

- The first request with a session establishes its baseline of known context; completing a step-up incorporates the current context into that baseline.
- **Sensitive actions with step-up:** destroying recipes, creating or deciding recertifications and activating the debug console.
- Step-up asks for your **password** and, if your account has a confirmed second factor, also the **TOTP code**. The step-up mark is valid for **10 minutes** (configurable).
- Without a valid mark, the sensitive action is denied — on web, by redirecting you to the step-up form retaining your destination; on API, with `403` and code `step_up_required` — **without expelling you**: non-sensitive matters continue working. If the second-factor provider is not available and your account requires it, it is also denied: fail-closed.

11.3 The vendor support account (fully recorded total access)

By permanent vendor rule, the vendor security operations centre account is **super-administrator of the product**: total access to the console, to all tenants and installations and to all functions — including paid and government ones — without licence or role gates limiting it, and recipient of critical alerts (DRP, licensing, security).

What this rule **does** require, the same as any account: authenticating as that account and passing adaptive step-up when applicable (that verifies **identity**, it does not limit role or licence). And each of its accesses through role control remains in the audit register **citing the rule**, like any other event. You can audit those accesses in your own log: total access is also fully recorded access.

11.4 Governed debug console

The debug console is the only surface that comes close to direct execution, and for that reason it is governed:

- **Disabled by default** in the installation configuration. Off in configuration, any live session **dies**.
- **Dual approval:** it is requested and approved by **two distinct approvers**, and never the requester themselves (separation of duties).
- **Mandatory step-up** to activate it (§11.2).
- **Ephemeral session:** 15 minutes by default, without silent renewal. Its state lives in volatile memory: a restart turns it off, it does not revive it.
- **Mandatory full recording:** without recording there is no session. Request, approvals, start and end remain in the transcript and in the audit register.

If your installation has not deliberately enabled it with this procedure, it does not exist: there is no alternative "hidden console".

11.5 Access recertification

Periodic campaigns (semi-annual) to review who has access to what, with evidence:

1. An administrator creates the recertification **campaign**.
2. Each access is decided individually: **certified** (continues) or **revoked** (withdrawn). Nobody can decide on **their own** access (SoD), and creating/deciding campaigns are sensitive actions with step-up (§11.2).
3. The campaign and its decisions are **exported as evidence** for audit.

11.6 Image scanning, Fail2Ban and other protections

- **Image scanning on admission (anti-CVE):** before applying a recipe, its images are scanned for known vulnerabilities. An image in **quarantine** under the policy (by default, correctable critical findings) **blocks** application without touching the engine. The scanner is an interchangeable piece behind an interface.
- **Fail2Ban:** the agent reads the host's **real** Fail2Ban state (jails, current failures and bans with their IPs) and exposes it in the agent status snapshot, **always present**: if the host has no Fail2Ban, the field says "not available" with its reason, to distinguish "without Fail2Ban" from "without data". The recipe can **declare** the expected jails on the host (e.g. SSH and sensitive port protection); the agent contrasts them with reality. The **daily operation** of Fail2Ban (bans, allow lists, decisions) lives in its own console linked from the agent sheet — the same Fail2Ban the agent reports — with its own authentication; Abzu **links it, it does not rewrite it or embed it with credentials**.
- **Updates and artifacts:** agent releases signed with mandatory verification and automatic rollback (§2.7); software bill of materials (SBOM) published per release; offline bundles are verified by signature before being applied (§6.5, §10.5).

> **Honest note — second factor on local accounts:** via SSO/OIDC, the second factor is provided by your institution's identity provider. For the **local accounts** of the government installation, mandatory second factor for administrators is a requirement recorded as pending in the product hardening line: this manual does not claim it as available for local accounts until your installation evidences it. Plan local administrator accounts accordingly (strong passwords, active step-up and recertification up to date).

11.7 Audit, hash chain and evidence export by framework

Audit register. Every security and operation event — enrolments, sign-ins (correct and failed), sensitive accesses, deployments, convergences, captures, approvals and DRP decisions (with their channel: console, voice or mobile), licence changes, recertifications, decisions denied by RBAC and accesses by the support account — is recorded with date and time, actor, tenant and outcome. In the government form the register is 100% local and **exportable to the institution's SIEM** (syslog/CEF). Sensitive changes are stored encrypted at rest.

Hash chain. Several system chains link each entry with the fingerprint of the previous one (decision audit, offline copy catalog, tape custody, DRP evaluations): altering, deleting in the middle or

reordering entries **breaks the chain and is detected on verifying it**; a manipulated register is rejected on opening it.

Evidence export by framework. From **Evidence / Compliance**, an administrator or auditor generates the evidence package for their installation: the facts already recorded (audit, inventory, recertification, recipe revisions) **labelled by control of the framework they choose**, in a JSON manifest with the SHA-256 fingerprint of each fact and a CSV index fact × control, re-verifiable by an external auditor without having to trust the console. The package is completed with the security architecture whitepaper and the SBOM.

The market rule (non-negotiable):

Market	Framework
Dominican Republic (government)	**NORTIC**
United States	**NIST** (SP 800-53 / CSF)
Rest of the world	**ISO/IEC 27001**

NORTIC is **only** the Dominican Republic and **is never marketed outside the DR**. The same fact, three readings: the labelling changes, the evidence is the same.

Honest scope of compliance. The management system — policies, risks, human resources, physical security, the operations centre — is **your institution's** in any market. Abzu **evidences** it, it does not replace it: a control without recorded facts comes out in the package with an empty list, not with fabricated evidence. Say it that way from day one before any auditor: it is what makes everything else credible.

---

12. Report signing and "View evidence" validation

12.1 The internal signing module

The product deliverables — audit, operational validation, backup and DRP reports — are delivered **signed and verifiable**. The signature is computed with an **internal module resident in the platform**: the signature lives inside your installation and **is not consumed by calling any external service**. It works the same in a government installation without internet egress or isolated (air-gapped).

The signature is digital (ed25519 algorithm) over the document and its exact content: document + fingerprint of its content + signer + moment + signature. Altering the document changes its fingerprint and verification fails; altering the signature register does too.

12.2 Load signing data in Configurations

Each installation loads **its own** signing data. Installation administrators only:

1. Go to **Configurations → Report signing**.
2. Complete the **signer identity**: name, position/role, email and organisation. It is the identity that will appear in each signature of this installation.

3. Provide the signing key, through one of these two paths:
    1. **Generate:** the platform generates a new key pair **inside your installation**. The private key remains stored **encrypted at rest**; the public key and its fingerprint remain visible (they are public). Generating replaces the previous identity: **one active identity per installation**.
    2. **Import:** paste your installation's own private key (in the format the form requests). It is validated **before** being saved by deriving its public key and its fingerprint; if it is not valid, it is rejected and nothing is saved.
3. Save. The private key is **never shown again**: neither on this screen nor on any other. Updating the signer's visible data (name, position, email, organisation) does not touch the keys.
4. Without an active signing identity, **nothing is signed**: the attempt to sign a report without a configured identity is rejected with its message (fail-closed). Configure signing **before** your first delivery.

12.3 Sign and deliver a report. "View evidence"

Each validation run (operational, backup or DRP) has its web report:

1. In **Validation**, each run in the list shows its type, scope, date and overall outcome, and the **"View evidence"** action.
2. **"View evidence"** opens the report for that run: header (installation/tenant, date, scope, overall outcome), the **list of tests applied with their outcome** (passed / failed / omitted / error, with their detail) and a **strip of images** with the **real capture** of each test that generates one (the same capture backup verification produces, §7.6). Without filler images: the test that generates no capture shows its **textual evidence** in the same report.
3. Runs reach this list through two paths: manual registration of a run with its tests (without false captures: what is registered by hand brings its textual evidence) or **import of the health report** generated by agent verification, which automatically becomes a run with its tests.
4. **Sign:** with the identity of §12.2 loaded, an administrator chooses **Sign** in the report. The platform signs the canonical content of the run and **registers the signature** (document, fingerprint, signer, moment).
5. **Delivery rule:** a report **without a registered and verified signature does not count as delivered**. The run list shows you, for each one, whether it is delivered — and that check verifies the signature against the current content: if the content changed after signing, it no longer counts as delivered.

12.4 Verify a received report

Whoever receives a report can check it without console access:

1. Open the report's **public verification page** (the verification path that accompanies the signature, with rate limiting).
2. The page shows the registered signature data and allows **checking** the document: its fingerprint is recomputed and the signature is verified against the signing installation's public key.

3. Possible outcome: **valid** (the document is exactly the signed one, by that signer, at that moment) or **invalid** (the document, its fingerprint or the signature do not match: do not accept it as delivered and escalate through your usual channel).

---

13. Integrations and notifications

13.1 Channels

Configure in **Integrations** the channels of **your** institution — the names that follow are integrations you connect with your own accounts, not the product's voice:

Channel	What it receives	Configuration per tenant
Email	Alerts and reports through your institution's email (in government, via your Exchange Online)	Sender and recipients of your installation
Microsoft Teams	Alerts to a channel of your Teams (your incoming webhook)	Webhook URL by severity if you wish
Slack	Alerts to your channels, **by severity**	One default webhook URL and, optionally, one per severity
PagerDuty	The alert **opens and closes** the incident with the correct on-call	Default routing key and, optionally, per severity. Acknowledgement (ack) from PagerDuty is reflected back in the console
ServiceNow	Change/incident tickets **linked** to the event and to the recipe revision that caused it	URL of your instance and your credentials. Incident closure links the post-incident report
Generic signed webhooks	Any system of yours subscribed to product events	See §13.2

Each channel is activated separately, with its **minimum severity** (information, success, warning, error, critical): below it, the channel receives nothing. Each channel's secrets (URL with token, keys, passwords) are stored as secrets: the form **only overwrites them when you write a new value** and **never shows them**.

13.2 Signed webhooks and subscriptions

1. In **Integrations → Subscriptions**, create a subscription: name, destination URL and the **event types** you want to receive (deployment, drift, backup, DRP, licence, among those the screen lists) with their severity.
2. Each delivery goes **signed (HMAC)** with the subscription secret: your system must verify the signature over the raw body before accepting the event — it is the same fail-closed standard required of any webhook that authorises actions (§9.2). A delivery without a valid signature must be rejected, also on your side.

3. Failed deliveries are **retried** with the product retry policy and remain recorded: in **Integrations** you see the latest deliveries with their outcome, and the **external links** bidirectional integrations have created (incident/ticket ↔ event ↔ recipe revision).

13.3 What events go out

Successful and failed deployments, drift detected, failed backups and verifications with outcome, DRP and drill events, licences close to expiry or in grace, and observability alerts (§16). The exact catalog of types is listed by your installation's own subscriptions screen: subscribe only to what your system will attend — an alert nobody attends is worse than none, because it trains people to ignore them all.

---

14. Public API, SDKs and Terraform

14.1 Versioned public API

In addition to the internal API with which the console itself operates, the product exposes a **versioned public API** for clients and integrators, in its own prefix (`/api/public/v1`), separate from the internal one:

- **Authentication:** tenant API token (bearer). The tenant is **always** derived from the token: any tenant identifier the client sends is **ignored**.
- **Scope:** agents, recipes, deployments, backups and backup policies, DRP policies, agent enrolments and licence status — with the same guarantees as the console. The normative specification is the product's OpenAPI document.
- **Response contract:** success as `{"data": ...}` ; error as `{"error": {"code": "<stable key>", "message": "..."}}` . Program against `code` , never against the message text (which is support, in your language). Codes: `unauthenticated` (401), `forbidden` (403), `not_found` (404 — **also** for resources of another tenant: their existence is not revealed), `validation_failed` (422, with detail per field), `rate_limited` (429).
- **Guarantees: each call remains in audit**; there is a **rate limit per tenant**; and the surface is **deny-by-default** — only the routes published in the OpenAPI exist.
- **Capabilities per token:** reading is the base; writing (e.g. create/update recipes, which are born with revision 1 and increment it on updating) requires the write capability in the token. Deployments and scheduled backups are exposed read-only through this API.

14.2 Official SDKs

There are official SDKs in **Go** and **TypeScript** (Python according to demand), **generated from the same OpenAPI** so they do not diverge from the API. Use them instead of building clients by hand: they inherit the envelope contract, the error codes and tenant derivation by token.

14.3 Terraform provider

For teams that already live in infrastructure as code, the Terraform provider works over public API v1 and shares the same typed client as the Go SDK:

Resource	Operations	Note
`abzu_recipe`	Create / read / update	The public API does not expose recipe deletion: destroying the resource in Terraform only removes it from state
`abzu_agent_enrollment`	Create (issues the token) / destroy (**revokes** it)	The `token` is sensitive and only exists after creation
`abzu_backup_policy`	Full CRUD	—
`abzu_drp_policy`	Full CRUD	Subject to the same dual-signature rules to activate (§9.3)

> **Verified limit — distribution:** the provider **is not published in any provider registry**. It is used **locally** with Terraform's development substitution mechanism (`dev_overrides`): compile the binary, point your Terraform CLI configuration to that folder and configure the provider with environment variables (endpoint and token) — never credentials in code. With `dev_overrides` there is no need to initialise from a registry.

14.4 GitOps: interoperate, do not embed (documented decision)

The evaluation committed in the roadmap was executed and its decision is: **interoperate, do not embed** a third-party GitOps engine inside the product. Reasons, in short: the government form must work without Kubernetes, without a network and on a single server; the agent **already is** the reconciler (versioned desired state + continuous reconciliation + reverse capture is exactly the GitOps pattern); and a second reconciliation loop over the same workloads would create ownership conflicts impossible to audit.

In practice: whoever already operates with a GitOps system consumes compiled recipes **from outside** — through the public API (recipes expose their revision number to detect changes), through the SDKs or through the Terraform provider — and those workloads remain outside the agent's scope **by explicit operator decision**, never in automatic dispute between two loops. A future synchronisation from an external Git repository, if it arrives, will be **unidirectional** (Git → console, with the console read-only for those recipes): the double source of truth writing on both sides is the classic failure of these integrations and will not be built.

---

15. Multi-site console federation

15.1 What it federates and what it does not

A **central** console (e.g. that of a coordinating institution) sees and governs — per permissions — the federated site/institution consoles, **without agents changing their local phone-home**: each agent continues speaking only with its site console.

The federation exchanges **signed metadata**: counted/aggregated inventory, licence status, health and policies. **Never** secrets or workload data (principle of data at the client's home). Synchronisation is **opportunistic, with retries**: if a site does not synchronise, the central shows it with its last synchronisation mark and its health as visibly **stale** — not as if it were up to date.

Each site continues operating **autonomously** even if the central or the federation falls: it loses the aggregated view, not operation.

15.2 Federation operations

1. **Register a site:** in **Federation**, register the site console with its identity. The relationship is established on signed metadata between consoles; agent or workload credentials are not shared.
2. **Synchronise:** synchronisation runs opportunistically; you can also trigger it per site from the same screen. Review the last sync mark and health per site before making decisions on aggregated data.
3. **Backup destination at another site:** define a **federated destination** (the alternate site's object storage, with **referenced** credentials, reusing the existing object destination) and use it as second destination for your policies (§7.5): one datacenter's backup lives in the other.
4. **DR plan between datacenters:** declare the **plan** (alternate site + destination coherent with that site; the model validates it) so a runbook can rebuild at the alternate site from that federated backup.

15.3 High availability of the console itself: decision

V2 decision: active-passive with tested failover. The central console is an aggregated view and administration point: a primary site serves it and an alternate site keeps its replica on standby (database restorable from federated backups) and assumes service with a failover procedure **periodically rehearsed and recorded like any drill** — rebuilding the central at the alternate from the federated backup and verifying the aggregated view.

Multi-site active-active remains a **future evaluation, not a v2 function**: it would require multi-master replication and write conflict resolution over operational objects that today have no merge semantics, with risk of double truth between datacenters, in exchange for an RTO ≈ 0 that no current scenario requires (the sites, which are those who operate, are already autonomous). **Do not promise active-active.**

Practical consequences: synchronisation is not real-time (it is opportunistic and is shown as such), federated licensing (§10.6) also holds during a failover because it is verified with public keys, and no procedure of your operation should assume cross-writes between consoles.

---

16. Observability: anomalies, predictions and KPIs

Advanced observability works on the series the system itself already produces (heartbeat every 30 s and status snapshots with the same cadence). Three rules govern it: **no black box** (each alert explains what deviated and by how much), **without data there is no figure** (explicit absence, never a disguised zero) and **nothing is applied on its own** (every recommendation is a draft with approval).

16.1 Anomaly detection on the baseline

- For each series (subject × metric: CPU, memory, disk, network, restarts, health-check latency, queue depth — per workload or per node), the **baseline** is the mean and deviation of the prior historical window (by default: 7 days).
- A sample is anomalous if it deviates at least **3 deviations** from the mean. Each anomaly event carries: metric, observed value, mean, deviation, direction, window and number of samples, plus an **explanation in clear language generated from those same data**. The event feeds the same audit register and alert channels (§13, §17).
- **Declared reliability:** if the baseline is not reliable (few samples — by default fewer than 20 — or little coverage — less than 24 hours —, or a constant series without deviation), **no anomaly is issued** and the result itself explains why. A reliable baseline needs, with default values, **on the order of 7 days of history** per series. And a series with strong seasonality (day/night) mixes regimes in a global baseline: bear that in mind when interpreting.

16.2 Concrete predictions (honest scope)

"Intelligence that predicts everything" is not promised: concrete, measurable predictions with their published basis are promised. Each prediction sealed in audit remains dated with its basis, to be able to measure its accuracy afterwards by comparing what was projected with what occurred.

Prediction	How it is calculated	When it stays silent
Disk filling up	Linear regression of use against time: estimated date when it reaches the total, with its interval (~95%), growth per day and goodness of fit	Without a positive slope → "no foreseeable filling", without an invented date. A discrete jump (importing a database, a dump) invalidates the estimate until the window absorbs it: that is why interval and fit are published, not only the date
Certificate expiring	Arithmetic on the expiry date of observed certificates	Outside the notice threshold (30 days by default), silence
RPO risk	Estimated backup duration = size ÷ **measured** speed; it projects when it will reach the limit (the window or the target RPO)	Without a measured speed it is not projected: explicit error
Restart storm	Count in a sliding window (by default ≥ 5 restarts in 1 hour per workload)	—
		—

Prediction	How it is calculated	When it stays silent
Overlapping backup windows	Intersection of declared daily windows (including crossing midnight)	

16.3 Capacity recommendations

Each recommendation is **actionable but never auto-applied**: it generates a **draft recipe revision** that a human approves (and in workloads with data, it is never applied on its own). The recommendation object itself is born with "requires approval" and "do not auto-apply" by construction; the engine has no method that applies changes.

- **Rightsizing:** compares the real 95th percentile of CPU/memory against what is declared in the recipe. Below 40% of what is declared → it proposes reducing; above 85% → it proposes expanding; the proposed value is the p95 with a 30% margin, rounded up. It requires a minimum of samples (by default 100).
- **Consolidation:** nodes where **all** workloads are underused, with p95 sums against what is declared on the table, so the decision is reviewable.
- **Windows:** converts detected overlaps into a window-shift proposal.

16.4 Security KPIs/KRIs panel

The inputs management review needs, calculated from the system's real sources:

KPI/KRI	Definition	Without data
Age of open vulnerabilities	Mean in days since first seen, with breakdown by severity and the oldest	0 open **with data** = real 0; without a source, "no data"
Patch latency	Mean discovery → resolution of resolved ones, with 90th percentile and maximum	Without resolved ones → "no data"
Recertification coverage	**Expired** recertifications completed on time ÷ expired. Non-expired ones count neither for nor against	Without expired ones → "no data" (not an invented 100%)
Verified restores	Catalog points verified with real restoration ÷ total, per workload	Empty catalog → "no data"

If a source fails or records nothing, **only that** indicator comes out as "no data" with its reason; the others continue to be shown. That a KPI says "no data" is itself the correct signal for management review: it means nobody is feeding that source.

---

17. Mobile app (PWA)

17.1 What it is and what it is not

The mobile surface is a **hardened PWA** (installable web application), not a native store app: it arrives earlier, updates with the console and covers what on-call needs. Its manifest is dynamic and white-label: the name and brand of the installed app come from **your** installation. Its scope is **deliberately narrow**: it is the on-call remote control, **not** a second console — it does not edit recipes or configure policies, and its route list is fixed and checked.

To install it: open your console's mobile address (`/movil`) from the phone browser and install it from the browser menu ("Add to home screen"). It requires HTTPS.

17.2 Status at a glance (read-only)

The mobile home shows: fleet health, **current RPO** (age of the last correct copy) and **current RTO** (duration of the last finished execution — an honest approximation until the DRP engine records real drill times) and licence status. Designed for the phone from which work is really done.

17.3 Actionable alerts

With push notifications activated (push service keys generated by your installation), the phone receives alerts with the **minimum context to decide**: agent down, drift in a critical workload, failed backup and drill outcome. Each alert links to the mobile home or to the corresponding approval. Subscriptions the push service declares dead clean themselves up.

> **Verified limit — push:** end-to-end push sending requires HTTPS, configured keys and a real browser subscription; it is tested at unit level (message signing and encryption) but **was not tested against an external push service** in this line. If you activate alerts, do a **real reception test** on your phone before deeming them operational.

17.4 DRP approvals from mobile

From **Approvals** you can **approve, reject or veto** (within the deferred approval window of §9.3) a DRP execution with **the same evidentiary force as in the console**:

1. Open the request: you will see the workload, the reason and the windows (until when it is not executed and until when it can be vetoed).
2. Choose the decision. **Each decision requires your password in the same request** (verification per decision, without reusable session marks): it is the step-up of the mobile surface.
3. The decision is written to the audit register with its **channel** (`movil` , just as `consola` or `voz` when those channels decide) and is **hash-chained** to the previous decision of your tenant: the decision chain is verifiable.

PWA security: its offline code only keeps the **public static shell**; the API, approvals and any request that is not a shell read are never cached, and navigation is "network first" with fallback to the offline page.

18. Troubleshooting and error messages

18.1 Access and session

Symptom / message	Meaning	What to do
Error 404 on entering	The tenant identifier or route does not exist — or the resource belongs to another tenant (answered the same on purpose)	Check the identifier with whoever gave you access; do not try random identifiers
"Invalid credentials"	Incorrect email or password in the platform or in the local IdP	Verify and retry. After several consecutive failures the temporary block arrives: insisting only prolongs it
Error 429 (rate limiting)	Too many login, verification or decision attempts in a short time	Wait the indicated time and retry once. It applies to login, public report verification and mobile decisions
The session closes on its own	It expired by maximum duration or by inactivity	Sign in again. If it occurs while working, review your installation times with your administrator
"Authentication service unavailable"	The identity provider does not respond	In commercial, use your foreseen alternative path if your installation has one; in government, local access. Escalate: without an IdP there is no new login, but agents and what runs do not stop
The passkey does not respond	No secure context, outdated browser or revoked/cloned key	Enter over HTTPS with a current browser; or enter with a password and revoke/re-register the key in your security
`step_up_required` (403)	A sensitive action requires step-up re-authentication (§11.2)	Complete step-up (password and, if applicable, code) and retry: the destination is retained. The mark lasts a few minutes
403 without further detail on a route	RBAC: your role does not cover that action, or the route has no assigned action (deny-by-default)	It is not a network failure: request the correct role. The denial is recorded

18.2 Agents and enrolment

Symptom	Meaning	What to do
The token "does not work" on enrolling	It expired (15 minutes), was already used (it is consumed) or belongs to another tenant	Generate a new token in the console and repeat registration within its validity

Symptom	Meaning	What to do
The agent starts in "local demo mode"	There is no token in the configuration, the console does not respond or the enrolment endpoint does not exist	Check `console_url` and the token in `/etc/abzu/agent.yaml` (permission 0600) and the server's outbound HTTPS; the agent says so in its log
Certificate error / connection refused after enrolling	The pinned console fingerprint does not match (possible unannounced rotation or you are pointing to another console)	Verify that `console_url` is your console. Do **not** set `insecure: true` in production: resolve the certificate
Agent "offline" in the console	No heartbeats arrive: network, agent stopped or certificate revoked	On the server: service status and its log. Remember: its workloads keep running and the agent retains the last desired state (§2.5)
Duplicate fingerprint marked for review	Two agents declare the same machine (cloning or reinstallation)	Check whether it is a legitimate reinstallation and re-enrol with a new token; the mark exists so a clone does not silently duplicate quota
`status --json` with empty `agent_id`	That agent is in local demo mode (it was never enrolled)	Enrol it (§2.3) if it must count as a managed server
No containers in inventory with the agent active	There is no access to the engine socket on that host	The agent reports it degraded with its reason. Grant socket access or accept that host without container management

18.3 Recipes and catalog

Symptom	Meaning	What to do
The recipe does not save: validation error with the field	The YAML does not meet `abzu.recipe/v1` (e.g. image without an explicit tag, invalid name, a secret in cleartext)	Correct the indicated field. Secrets go only as references `${secrets.<name>}`
Application is blocked "by admission"	An image was quarantined for vulnerabilities under the policy, or the scanner is not available and the policy is fail-closed	Update the image to a version without the finding or knowingly adjust the policy. Do not disable scanning to "get through"
Drift that returns again and again	Something outside the console changes the workload (or the recipe does not reflect a decision already taken)	Use **reverse capture** (§5.5): review the draft and promote it if the change is legitimate; if not, convergence is doing its job
A service with data does not converge on its own	It is deliberate: in `stateful` the `manual` policy is mandatory	Approve convergence in the console after reviewing the diff and having a prior snapshot

Symptom	Meaning	What to do
Catalog entry "not verified" / the agent rejects installation	Missing/invalid signature, publisher not trusted for that agent, or the agent has **no** trusted key configured	Pin the publisher fingerprint through an independent channel and configure its keys in the agents. Without a valid signature there is no installation: there is no exception
Catalog download is blocked with a compatibility reason	The destination, engine or agent version do not meet what the entry declares	Install at a compatible destination or ask the publisher for a version that declares your destination. Compatibility is applied, not skipped
Offline bundle import "imports nothing"	Some verification failed (issuer fingerprint, bundle signature or some entry's signature) and import is atomic	Verify the issuer fingerprint through the separate channel and that the bundle was not altered in transport. Do not retry with a "similar" bundle

18.4 Backups, PITR and tape

Symptom	Meaning	What to do
The job reports "waiting for window"	It is outside the declared backup window	It is correct behaviour: wait for the window or trigger "Back up now" if urgent (it is audited)
The copy to the second destination aborts	The fingerprint verified during the copy does not match: the partial copy is deleted on purpose	Check the destination and network and retry. Never force keeping a copy whose fingerprint does not match
"The object is protected" when trying to delete a point	The point is sealed and/or with current Object Lock (GOVERNANCE/COMPLIANCE)	It is immutability working. In GOVERNANCE only a role with storage bypass permission could — and must not — do it outside the product. Wait for retention expiry
Verification says "scan omitted" (`skipped`)	The antivirus is not on the agent host	The report does not pretend: install the scanner on that host if your policy requires scanning at each verification
Verification fails the count	Rows restored in sandbox do not **exactly** equal the manifest	Do not use that point for production until diagnosed: it may be an inconsistent backup or a sandbox with another engine version. The RTO of that point is not deemed measured
Restoring a point requires "accepting the risk"	The point is marked **infected** and its status propagated to copies	Only with explicit, recorded approval, and knowing you restore marked content. Always prefer the last clean verified point
PITR: "the chain is broken" / cannot restore to that instant	A link is missing (base or some intermediate incremental) or	The system does not promise an impossible PITR: choose an instant covered by a complete chain and review retention — pruning should never break a retained chain;

Symptom	Meaning	What to do
	the chain does not validate in order	if a manual action outside the product did, own it in diagnosis
CDC: RPO (lag) grows without stopping	The shipping service is down or cannot upload segments: position only advances after verified upload, so the gap is real	Revive the shipping service and verify it uploads and confirms segments. Meanwhile, your real RPO is the lag you see: not the policy's
Writing to tape fails "cartridge outside the pool"	The loaded cartridge does not belong to the policy pool	Mount a cartridge from the pool. Reading, by contrast, follows the catalog: to restore, mount the barcode the catalog indicates
Recycling a cartridge is rejected	It has points with current retention — or the pool is WORM, where not even forcing can do it before expiry	Wait for expiry. In WORM there is no operator path: that is the sense of the pool
The console warns "a driver is required for your unit"	The detected tape unit requires its own driver per the unit table	Generate the agent requirement (§8.3) and send it through the secure channel. Do not install driver packages from another source: without a valid vendor signature, installation is rejected whole
Driver installation is rejected	No configured public key, invalid signature, fingerprints that do not match or incompatible unit — in that check order	The attempt is recorded with its reason. Correct the cause (key, intact package from the secure channel, correct unit); do not retry the same altered package
Tape status "no hardware on this host"	No drive/changer detected on that host (detection also says so when tools are not present)	That host cannot be a tape server. The virtual library serves for tests and demos, not as production tape

18.5 DRP, licences and signatures

Symptom	Meaning	What to do
The autonomous policy "never acts"	Some guardrail fails at each evaluation: most commonly it remains in draft without the **second signature**, the level exceeds the installation ceiling or the kill switch is active	Open the persisted evaluation: it brings the result **per guardrail**. Activate the policy with dual signature, adjust the ceiling or deactivate the switch, as appropriate
Evaluation remains at "rollback required"	Post-verification (health or consistency) failed **or is not recorded**	Treat it as an untrustworthy restoration: review the restored workload and the associated drill

Symptom	Meaning	What to do
		before any new autonomous attempt on it
Nobody approved and "nothing with data was executed"	The approval chain (console → voice) did not obtain a valid "I authorise" in time	It is fail-closed working: review the on-call chain (registered numbers, PIN) and trigger again through the channel that responds
Voice authorisation is not accepted	The call webhook did not verify its signature, or the PIN does not match	Do not authorise DRP by voice until verifying the webhook signature configuration: it is a registered prerequisite, not a formality
`entitlement_required` (403) on VM actions	The paid VM function is in "no function" state for your assignment	Inventory continues to be seen: it is a soft block. Manage the function in the vendor assignment (§10.4); in trial, remember everything works and the modal does not block
"Cannot add servers" with everything else working	Quota exhausted or trial/ subscription expired: **soft block**	What runs is not at risk. Expand quota or renew (§10.3); rejected new servers enter when quota is released
A disappeared server "keeps counting" for some days	Its quota is released after **7 days** without reporting	Wait for grace or verify that the server really should no longer count and that it is not a down agent that will return
The offline licence "does not validate"	Invalid signature, different installation identifier, unsupported version/algorithm or expired subscription	Request the file issued **for your installation identifier** and unaltered. Any edit of the file invalidates the signature
"Cannot sign: no identity"	Your installation has no active signing identity loaded	An administrator must load it in **Configurations → Report signing** (§12.2) before the first delivery
A signed report "no longer counts as delivered"	Content changed after signing: the check verifies the signature against **current** content	Re-issue and sign the correct report. A report without a registered and verified signature does not count as delivered (§12.3)
Exported evidence brings a control with an empty list	There are no recorded facts for that control in your installation	It is the exporter's honest answer: generate the facts (operate, recertify, verify) or own the control as your institution's responsibility (§11.7)

18.6 How to ask for help with evidence

When you escalate a problem, attach: (1) the agent or workload sheet with its status snapshot (`status --json` in the agent; it includes the tape and Fail2Ban sections with their reasons), (2) the job or evaluation identifier in the console, (3) the audit entries for the interval (exportable) and (4) in backups, the **manifest** of the affected point. With that, any diagnosis starts from facts and not from assumptions.

---

19. Operating best practices

1. **Single-use token by default;** fleet key only for bulk registration windows, revoked on closing them.
2. `insecure: true` **never in production.** If an enrolment "only works that way", the problem is the certificate: fix it.
3. **Everything with data, in** `driftPolicy: manual` **and with a prior snapshot.** Automatic convergence is for what has no state.
4. **Promote reverse capture the same day** as the emergency fix: the unreviewed draft is tomorrow's drift.
5. **Scan on admission and do not disable it to "get through"** a deployment: quarantine is the function, not the obstacle.
6. **3-2-1 from day one:** second destination with its own credentials, dedicated bucket with Object Lock and critical points in COMPLIANCE. The hardened repository is not optional (§7.5).
7. **Verify by restoring, not by looking:** verification policy per workload + scheduled integrity check + monthly drill. The RTO that counts is the **measured** one.
8. **Monitor CDC lag** as you monitor disk space: a silently down CDC is a false RPO.
9. **The tape catalog is critical data:** without it there is no possible restoration from tape. Back it up off tape.
10. **Autonomous policies: start in** `manual`**, rise to** `deferred approval` **and only then to** `bounded autonomous`, with drills under the policy at each level. The kill switch is tested as a fire extinguisher is tested: before needing it.
11. **Real dual signature:** the second signer reviews scope, window and maximum RPO; signing without reading turns SoD into a formality and the evaluator cannot save you from that.
12. **Licence assignment is touched only in the vendor's panel.** If Abzu "does not reflect" a function, diagnose validation and its grace before looking for a parallel register: it does not exist, on purpose.
13. **Configure the signing identity before the first delivery** and treat its private key as the key it is: encrypted at rest, never exported on screen.
14. **Alerts someone attends:** subscribe by severity and by channel with judgement. Review quarterly which alerts arrived and who acted.
15. **Recertify on time:** recertification coverage is a KPI of management review (§16.4); letting it expire is seen, and must be seen.

16. **In government, zero telemetry by default:** no egress from the installation without the institution's explicit, recorded authorisation.

---

20. Glossary

- **Agent:** lightweight program installed on each managed server or cluster; it always initiates the connection toward the console (phone-home) and authenticates with its own certificate (mTLS).
- **Recipe:** YAML document `abzu.recipe/v1` that declares the desired state of an application. It is the source of truth; everything else is generated from it.
- **Revision:** immutable version of a recipe; each change creates a new one (a commit) and never rewrites previous ones.
- **Drift:** difference between the real state and the desired state. Its treatment depends on policy: `auto`, `manual` or `report`.
- **Reverse capture:** generating a draft recipe revision from the real state, for human approval.
- **Manifest (backup):** JSON document accompanying each restore point with its fingerprint, counts and measured times.
- **PITR:** restoration to a concrete instant within the retention window, replaying the base + incrementals chain up to that moment.
- **CDC:** continuous backup: sending of log segments (WAL/binlog) as they close, with confirmation only after verified upload. Its metric is the lag.
- **Sealed point:** GFS point (weekly/monthly/annual) that cannot be pruned or, with Object Lock, deleted until its period expires.
- **Sandbox (verification):** isolated, disposable environment, with no network toward production, where restoration is performed to verify.
- **Runbook:** special recipe that orders reconstruction in a disaster (network → volumes → data → applications).
- **Guardrail:** verifiable limit of an autonomous policy (scope, window, RPO, prior snapshot, ceiling, kill switch). If one fails, there is no autonomous action.
- **Kill switch:** global switch per installation that cancels all DRP autonomy instantly.
- **Seat:** licence unit: an enrolled agent. Containers per agent are unlimited.
- **Soft block:** on a licence or paid-function problem, what is new stops and what exists continues intact.
- **Step-up:** step-up re-authentication on anomalous context, required before sensitive actions.
- **Bundle:** signed package transportable offline (government installation, offline catalog, offline licence). It is verified by signature before use; if it does not verify, it is not used.
- **Fail-closed:** on doubt — signature, licence, policy, channel — the system does not act and escalates. It is the default behaviour of the whole product.

End of manual. SEGRD — Abzu v2 User Manual, 2026-10-18. This document describes the product according to its code and design in force in the integrated *`main`* branch; verified limits are declared in the corresponding section of each function.