ECLIT TECHNOLOGY RADAR

What we run in production.

62 technologies, four rings. What we run in production, what we are trialling, and what we stay away from.

This document is the downloadable form of eclit.com/en/radar.

EDITION	TECHNOLOGIES	UPDATED
2026.1	**62**	**2026-08-25**

What this radar is

A technology radar is a document in which a company states plainly which technologies it uses and at what maturity. For a reader making a purchasing decision it is more useful than marketing copy: unlike “we are cloud experts”, it can be checked.

A ring is not a score. It does not rate the technology; it says what stage we use it at in our own operation. A technology in “Assess” is not a bad technology — we simply have not put it into production.

The four rings

ADOPT	Adopted — in production, our default choice	**34**
TRIAL	In trial — proven on real work, not yet the default	**15**
ASSESS	Under assessment — we are watching it, not running it in production	**12**
HOLD	On hold — we do not choose it for new work	**1**

ADOPT · 34

Adopted — in production, our default choice

Platforms

Hybrid cloud
A hybrid cloud operating model. Adopted because it is the real default in the field: most organizations have a layer that must stay local for regulation, latency or depreciation. The question is not "should we move to cloud" but which workload sits where.

Kubernetes platforms
Production Kubernetes platforms. Adopted: it is the backbone of modern workloads and its self-healing genuinely reduces operational load. The trade-off: the cluster heals itself but nobody heals the cluster — without a platform team, Kubernetes is not a solution but a second system.

Managed DR platform
Managed DR & backup platform. Resilience layer.

Multi-cloud architecture
Architecture that is not tied to a single provider. Adopted, though not in the marketing sense: the aim is not to run every workload on every cloud but to design knowing the exit cost. Portability is decided by the data layer, not the compute layer.

Private cloud
Private cloud on VMware and KVM. Adopted and a core Eclit service: data residency, predictable cost and protecting existing license investment keep this layer in place at most organizations. Not a competitor to cloud but one half of the hybrid model.

ScaleEngine
Our in-house automation platform. Adopted: our own product for moving the repeating part of MSP work into software. The claim here is about the model rather than the technology — writing the work into a platform once instead of doing it by hand at fifty customers.

Tools

Ansible
Configuration control.

ArgoCD
GitOps delivery backbone on Kubernetes. Adopted: desired state lives in a Git repository, drift in the cluster becomes visible against it, and a rollback is one commit. It closes the "who changed what, when" gap left by manual kubectl changes.

Elasticsearch
Search and analytics engine; the core of our log intelligence stack. Adopted: query performance at volume and a mature query language make it the default. The cost to know about is disk and memory — without a retention design from day one, cluster growth gets away from you.

ELK / OpenSearch
The log collection, parsing and visualisation layer. Adopted: keeping collection and parsing separate from the engine means the pipeline survives a change of search engine. It carries our centralized log management service.

Enterprise backup stack

Veeam vb. DR reliability layer.

Foreman / Katello
System lifecycle and patch automation. Adopted: running server inventory, content views and patch distribution from one place makes it auditable which patch reached which server and when. It underpins our patch and vulnerability management service.

GitLab
Code, pipeline and workflow in one place. Adopted: keeping them together rather than in three tools leaves a traceable chain from a change commit to its deployment. In an audit, "where did this change come from" has an answer in one system.

Grafana
Our standard for observability visualisation. Adopted: bringing metric, log and trace sources into one panel removes the need to move between three tools during an incident. It is the first screen the on-call engineer opens.

MongoDB
Our primary document-based NoSQL database for flexible schema designs.

MySQL
Supported and operated as relational database infrastructure. Adopted: mature, well understood and the default for most applications. On the operations side we watch two things — replication lag and restore-tested backups; both degrade silently if not measured regularly.

Palo Alto
Enterprise perimeter security and next-generation firewall. Adopted: application-aware policy and threat prevention are mature. In operation the critical point is policy hygiene — rules accumulated over years that nobody dares delete will slow down even the best appliance.

PostgreSQL
Our default relational database choice for new and modernized workloads.

Prometheus
Our standard for metric collection and alerting. Adopted: the pull model and label-based data model remove the need for manual inventory in dynamic estates. Long-term retention is a separate decision — Prometheus local storage was not designed as an archive.

Prometheus & Grafana
Observability backbone.

Proxmox / KVM stack
Cost-efficient cloud foundation.

Terraform
Infra standardization.

Veeam
Core enterprise backup platform for VM and cloud workloads.

VMware ecosystem
Enterprise private cloud reliability.

Techniques

Blameless incident culture

Looking for the system fault after an incident, not the person. Adopted, because it is not a cultural preference but a measurement one: an engineer who expects blame reports late and reports partially. When the record degrades, so does root-cause analysis. Post-mortem discipline is therefore an operational requirement.

Customer health scoring
Tracking service health by reliability measures rather than ticket count. Adopted, because ticket count is a misleading indicator: it falls in a healthy estate and it also falls when a customer has given up. The health score is computed from incident frequency, time to resolution and recurring root causes.

GitOps as Operational
Infrastructure, configuration, policy and change all flowing from one source. Adopted, and for us this is an operating model rather than a tool: the desired state of the estate lives in a repository, drift is measurable, rollback is one commit, and every change has an author and a reason.

IaC-first service provisioning
Manual builds are not accepted; everything comes from code. Adopted: a hand-built server carries a configuration nobody fully remembers a year later and cannot be rebuilt under disaster conditions. A code-built server is reproducible — that is the precondition of a recovery plan.

Reliability-first architecture
Delivering uptime rather than building infrastructure. Adopted, and this is a contractual choice more than an architectural one: when what is delivered is availability rather than servers, design decisions follow — single points of failure, recovery time and drills are on the table from the start.

Service Ownership Model
End-to-end ownership instead of an L1/L2/L3 split. Adopted: handovers between tiers spend most of the incident re-explaining context. Under ownership the same team sees both the call and the root cause, which is why the same fault does not come back a second time.

Languages & Frameworks

Bash
Low-level automation and system scripting. Adopted and staying: it is the one language present on every server and the one that runs in a rescue shell. Python is the backbone; bash is the language of the places where Python cannot be installed, or where you cannot wait for it to be.

Python
Our automation backbone and operations scripting standard. Adopted: the library ecosystem, readability and breadth of team knowledge make it the default. Over an automation lifetime the expensive part is not writing it but someone else reading it two years later.

REST API-first mindset
Designing every capability as an API first. Adopted: it reduces dependence on any one tool and saves rewriting every integration when a tool has to be replaced. The portal and the automation use the same API, so no gap opens between them.

YAML-driven infra definitions
YAML as the configuration standard. Adopted: being a common language across tools and producing readable diffs in version control make it the default. Its known weakness is indentation sensitivity — which is why YAML that is not schema-validated does not reach production.

TRIAL · 15

In trial — proven on real work, not yet the default

Platforms

Internal Developer Platform
An engineering experience platform. In trial: letting a developer provision an environment without knowing the infrastructure speeds delivery, but a platform needs product-style maintenance — an IDP without an owner becomes a layer nobody updates within six months.

Multi-tenant abstraction
A multi-tenant abstraction layer that turns a service into a platform. In trial: a shared platform instead of a per-customer build raises consistency. The hard part is isolation — keeping one tenant load from touching another is the most expensive requirement in the architecture.

Open-Source AI Stack
Experimenting with open-source models and inference stacks for enterprise internal use.

Self-service portals
Infrastructure becoming a product. In trial: letting a customer see their own estate is far stronger transparency than waiting for a monthly report. It backfires when the data is not live — a dashboard showing stale data is worse than no dashboard.

Tools

FinOps dashboards
A transparent cost dashboard for the customer. In trial: showing which workload a spend belongs to is only possible with tagging discipline. In an untagged estate the dashboard does nothing but grow the "unknown" line, so tagging comes before the dashboard.

LLM-based ops assistant
A language-model assistant that goes from ticket to summary, reply and runbook suggestion. In trial: it shortens response time on repetitive L1 requests. The boundary is explicit — it does not go to production until how customer data reaches the model is defined contractually.

n8n
Workflow automation that wires services together with minimal code. In trial: it produces results quickly for internal processes. Our rule: no flow that touches a customer estate stays in a visual tool — that belongs in code and version control.

Voice AI support agents
L1 support automation with voice AI. In trial and cautious: it works for simple, routing calls. In enterprise support the caller usually already has a problem, and the cost of being misunderstood exceeds the time saved. The path to a human stays open at all times.

Zerto
Continuous Data Protection (CDP) for mission-critical disaster recovery.

Techniques

AI-assisted operations
Language-model support for incident summaries, root-cause suggestions and runbook drafts. We keep it in trial because the speed-up in a noisy alert queue is real, but suggestion quality depends on how well the estate is labelled — in a poorly labelled estate it is confidently wrong. It sits in front of the on-call engineer,

not instead of them.

FinOps as discipline
Turning cost control into an operational KPI. In trial: moving the cloud bill from a monthly surprise to a weekly indicator works, but reporting without an owner for the saving ends up as a report nobody reads.

Platform engineering
The MSP model becoming platform engineering. In trial: moving repeating patterns into a platform instead of a bespoke build per customer reduces quality variance. The risk: if the platform does not meet the real need, everyone starts working around it.

Self-service infrastructure
An API for the engineer, a portal for the customer. In trial: serving routine requests without a ticket relieves both the customer and the on-call engineer. Authorisation sets the boundary — opened without a clear definition of who may do what, self-service breaks the audit trail.

Languages & Frameworks

Event-driven automation
Making infrastructure react to events. In trial: replacing scheduled jobs with event-triggered flows shortens response time, but debugging gets harder when the event stream is not observable — a cron job that did not run is obvious; an event trigger that vanished is not.

Go
For platform tooling that needs performance and reliability. In trial: shipping as a single binary is a real convenience for operations tools, but the team knows Python better than Go, and maintaining a tool outlasts writing it. Being trialled in a narrow scope.

ASSESS · 12

Under assessment — we are watching it, not running it in production

Platforms

AI inference platform
On-premise GPU pools and inference infrastructure. In assess because the real question is economic, not technical: owning hardware makes sense when data cannot leave the organization or inference volume is steady; otherwise cloud GPU stays cheaper. Volume and data residency decide it.

Edge workload platforms
Managing distributed infrastructure from the edge. In assess: running workloads locally at a plant floor or a retail branch is a genuine need, but if edge nodes cannot be managed centrally, operational cost overtakes the workload itself as the fleet grows.

OpenStack
For private cloud builds that need deep customisation. In assess: the capability is not in question and neither is the operational cost — OpenStack is not a product but a standing engineering commitment. We recommend it only where scale and customisation justify that commitment.

Sovereign cloud patterns
Regulation-driven cloud architecture patterns. In assess: KVKK, banking regulation and sector rules decide where data may sit, and these patterns turn that constraint into architecture. The field moves — as regulation changes so do the patterns, so we do not fix them early.

Tools

AI log analysis engines
Statistical pattern and anomaly detection across log volume. In assess because these tools are good at finding an unknown problem and expensive at re-finding a known one — a well-written alert rule does the same job for free. We think of it on top of the existing alert set, not instead of it.

Policy as Code frameworks
Compliance automation with OPA and similar frameworks. In assess: moving a compliance rule from a document into code makes "is this rule enforced" measurable at audit time. The difficulty is not writing the rule but deciding what happens when it is violated.

Secrets lifecycle tools
Platform-level automation of secret lifecycles. In assess: storing a secret in a vault is a solved problem; rotating it is not. The real work is knowing which systems break when a secret expires — a vault without that inventory produces outages at regular intervals.

Techniques

AI maturity scoring
AI-assisted scoring of a customer estate technical health. In assess because producing a score is easy and making it DEFENSIBLE is hard: telling a customer "your estate scores 62" only helps if you can show which measurements produced the 62. An opaque score is worse than no score.

Autonomous remediation
Automated flows that go from alert to fix without human approval. In assess, and we are not rushing: for narrow, reversible fixes (service restart, disk cleanup) the value is clear, but as scope widens the cost of a wrong diagnosis rises steeply. Human-approved first, autonomous later.

Predictive reliability

Catching the signal before the incident. In assess: prediction works for slow degradations like disk fill and memory leaks, and does not work for sudden failures. If the false-alarm rate stays high the on-call engineer stops trusting alerts, and that is the real loss.

Languages & Frameworks

AI orchestration frameworks

Management and chaining of agent-based workflows. In assess: the field moves fast and a framework chosen today can be unmaintained a year later. Before it touches production operations we want to see what it does on failure, whether it can be rolled back, and what audit trail it leaves.

Declarative workflow DSLs

Declarative workflow languages that turn a runbook into code. In assess: the promise is clear — an executable runbook is both documentation and automation. But every DSL is a new learning curve and a new dependency; a language the on-call engineer cannot read at 03:00 stops being a runbook.

HOLD · 1

On hold — we do not choose it for new work

Tools

Citrix NetScaler

Proprietary hardware ADC. On hold: the appliance itself is capable, but its licensing model, upgrade windows and hardware-bound scaling limit flexibility next to software-defined load balancing. For new builds we recommend software-defined ADC and WAF; we continue to support existing installations.

The limits of this document

This radar describes **Eclit's own operation**. It is not general technology advice. The right answer in your environment may differ; a technology sitting in our “Hold” ring is not a bad one, it is one that does not fit our workload.

The rings are the engineering team's shared assessment, not the result of a measured benchmark. That is exactly why an edition number and a date sit on the cover: this document is a snapshot of a moment.

The current assessment of every technology, with the related articles: **eclit.com/en/radar**