

OPEN FOUNDATION GUIDE · 007

Lattice: the complete foundation guide

Mechanism, artefact, interfaces, stack joins, failure, reproducibility, performance method, security, deployment, adoption, and exit for Lattice.

LATTICE

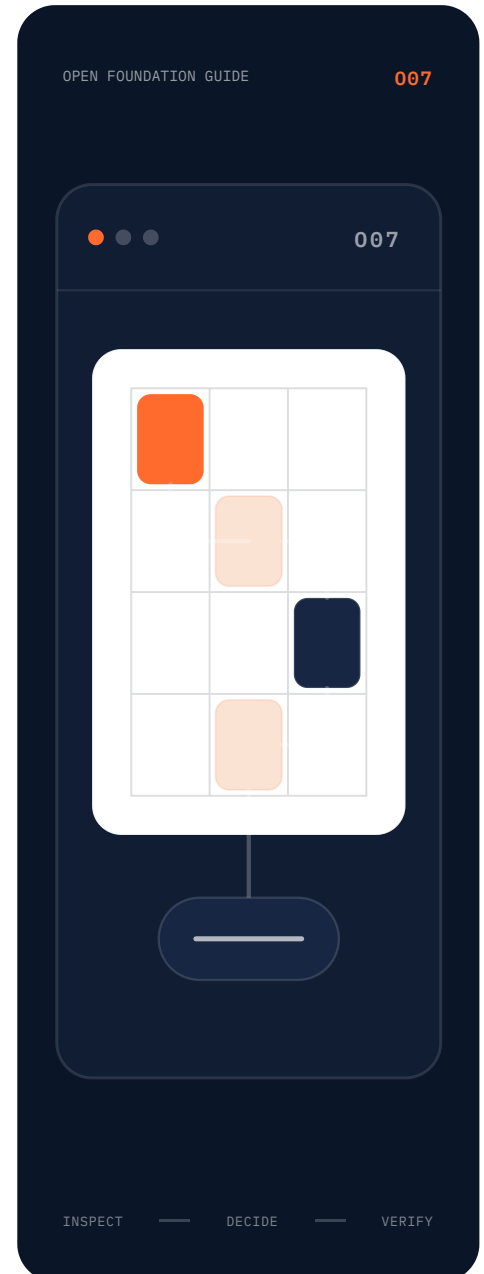
MECHANISM

ARTEFACT

REPRODUCIBILITY

SECURITY

EXIT



Read this when the name Lattice is not enough and the team needs to inspect the component contract and reproduce its claim.

007

CONTENTS

Lattice, from mechanism to independent check

The guide uses one illustrative exercise: an application evaluates eligibility under an approved versioned rulebook; the same facts and artefact return the same answer on supported targets.

PRIMARY READERS

Engineers, maintainers and independent reviewers

READING DEPTH

Deep technical

DECISION THIS SUPPORTS

A component-level guide for teams deciding whether and how to rely on Lattice for compiled deterministic policy and constraint evaluation.

01

Read this first

03–04

Scope, reader decision, vocabulary, source and claim status

02

Mechanism and artefact

05–12

Responsibility, input, pipeline, manifest, interfaces, stack joins, and failure

03

Evaluation and adoption

13–19

Reproducibility, performance, security, deployment, integration, diligence, and exit

04

Sources and next action

22

verification records and component proof package

05

Subject index

20–21

An alphabetical finder for concepts, controls, products, and decisions.

READING RULE

The published Lattice open-foundation page owns factual statements. Illustrative envelopes and evaluation procedures show what a credible integration should specify; they are not invented project APIs.

SOURCE DISCIPLINE

Dweve-specific facts are taken directly from the English website text audited 2026-07-20; web-navigation wording is humanised for print. Evaluation guidance is editorial, and scenarios and derived visuals are illustrative. Unsupported synthesis is replaced by canonical copy or omitted.

READ THIS FIRST • DECISION

Who should read the Lattice guide

Engineers, architects, security and assurance teams, product owners, open-source reviewers, and operators can use this guide to evaluate compiled deterministic policy and constraint evaluation.

Lattice is an inspectable foundation for compiled deterministic policy and constraint evaluation. The decision is whether the mechanism and artefact fit a selected product/workflow boundary, can be reproduced on the organisation's inputs and targets, can be packaged and operated safely, and can be replaced without losing data or historical evidence.

The worked example begins when an application evaluates eligibility under an approved versioned rulebook. The expected normal behaviour is: the same facts and artefact return the same answer on supported targets. The fault case is: a fact is missing or the artefact checksum/version changes. A sound result is: evaluation refuses/escalates or records the new decision identity rather than guessing a default. These statements are illustrative evaluation framing, not a customer outcome.

Verify current source/repository and release status, licence, documentation, supported interfaces/platforms, product integration, benchmark method, security, deployment artefacts, support, and rights at the decision date. The website is the sole factual source for this edition; the installed component supplies final proof.

FOUNDATION**Lattice**

compiled deterministic
policy and constraint
evaluation

OBJECT**compiled
policy artefact
and query
result**

The bounded technical
artefact.

FAULT**Deliberate
negative case**

a fact is missing or the
artefact checksum/version
changes

DECISION**Fit + reproduce**

Integrate, operate, verify
and exit.

PRIMARY VERIFICATION

Compile the approved rulebook twice, compare artefact identity, run boundary and missing-fact cases on supported targets, switch failover hardware, reproduce the benchmark in its scope, replay a historical query with its old artefact, and alter the checksum.

[READ THIS FIRST](#) • [TERMS](#)

Vocabulary and claim boundary

Foundation, project, surface, artefact, manifest, verifier, benchmark, product integration, and commercial/source rights answer different questions.

Mechanism and artefact

The mechanism is the compiled deterministic policy and constraint evaluation operation Lattice performs. The artefact is compiled policy artefact and query result plus the component-specific material named on the route. A manifest makes inputs, versions, configuration, target and dependencies explicit. A verifier or harness tests the declared proposition.

Project, product, and surface

The open-foundation description describes the project. A product may use it without exposing the same interface or licence. A library, CLI, service, sidecar, FFI, MCP or WASM build is a surface; surface availability and parity require current verification.

Measured and assurance claims

A number remains tied to its corpus/input, hardware, version, configuration, method, baseline and limitation. Integrity, determinism, provenance, containment or offline checking support bounded assurance; they do not prove truth, correctness, legitimacy, legal compliance, fitness or commercial entitlement.

TERM	ANSWERS	DOES NOT ANSWER
Foundation	Which mechanism?	Who owns product work
Artefact	What can be consumed/checked?	Why purpose is legitimate
Surface	How invoked?	Semantic parity automatically
Open/source	What can be inspected?	Current release/right automatically
Benchmark	What was measured?	Production SLA/ROI
Verifier	Which proposition holds?	Every upstream premise is true

Keep these distinctions attached to every adoption claim.

FOUNDATION COUNT WARNING

Lattice keeps the source and outcome of each step. The final answer can point back through the route, so a reviewer sees which rule fired and which checks merely supplied context.

01

MECHANISM AND ARTEFACT

Lattice must earn its place at one technical join

Define the problem, input, mechanism, output artefact, interfaces, dependencies, failure, and evidence before discussing adoption.



The component contract used throughout the chapter.

MECHANISM • RESPONSIBILITY

Lattice solves one inspectable technical problem

Lattice is the foundation for compiled deterministic policy and constraint evaluation. Its value is the contract and artefact it contributes, not “open source” as a decorative label.

Lattice compiles rules and constraints into a memory-mappable artefact evaluated inside the application. A named rulebook checksum and deterministic query path make the same decision reproducible across the route's supported targets without an outside policy service in the critical path.

The worked example begins when an application evaluates eligibility under an approved versioned rulebook. The principal technical object is a compiled policy artefact and query result. Under the normal path, the same facts and artefact return the same answer on supported targets. This gives the integration a bounded proposition it can test and retain.

Bring one recurring knowledge product, such as a compliance dataset, policy answer system, model-training corpus, research benchmark or operational report, and the sources behind it. Spindle shows which preparation becomes reusable, which claims remain unsupported, which rights limit use, how every output is assembled and what happens when a source changes.

FOUNDATION

Lattice

compiled deterministic policy and constraint evaluation

OBJECT

compiled policy artefact and query result

The artefact or state a caller consumes.

PUBLISHED WEBSITE SOURCES

Lattice open-foundation page

Verify current project/release/source status.

DECISION

Fit + reproduce

Inspect mechanism, integrate, fail, verify, and exit.

PRIMARY BOUNDARY

Deterministic rule evaluation does not decide that the rule is fair, lawful, current, or applicable. Those facts remain with the policy owner and source-governance process.

MECHANISM • INPUT CONTRACT

Define the component input before invoking it

A file, query, module, model, event or value becomes a reproducible input only when its identity, schema, configuration, rights, and comparison scope are declared.

For the example, an application evaluates eligibility under an approved versioned rulebook. The component call should bind work/purpose, input identity and origin, schema/format, Lattice version, configuration/profile, target/runtime where relevant, resource and security limits, expected output or verdict, and evidence requirements.

Use content hashes for immutable bytes and signed/versioned identities plus effective metadata for state that cannot be reduced to a file. Preserve collection, parsing, normalisation or translation steps that materially affect meaning. If input is a reference, the manifest must make it resolvable under the selected retention and access policy.

Reject malformed, missing, ambiguous, inaccessible, unauthorised, incompatible, over-limit, or unsupported input before producing a normal-looking output. A valid negative result is different from invalid input; a partial parse or collection is different from empty content; an unsupported target is different from a failed supported execution.

ILLUSTRATIVE LATTICE REQUEST

work_id	component-4f9e
purpose	bounded-evaluation
input	content-addressed identity
schema	declared format + version
component	Lattice@pinned
config	profile + target + limits
result	typed artefact or verdict
evidence	manifest + raw check

Illustrative envelope, not a claimed current API.

INPUT MUTATION TEST

Change one identity, schema, configuration, target or authority field and verify the result either changes under a new manifest or is rejected explicitly.

MECHANISM • EXECUTION

The Lattice mechanism as a four-stage pipeline

The exact implementation is component-specific; the evaluation pattern remains validate, execute, emit a typed artefact, and bind it to evidence.

Lattice compiles rules and constraints into a memory-mappable artefact evaluated inside the application. A named rulebook checksum and deterministic query path make the same decision reproducible across the route's supported targets without an outside policy service in the critical path.

At receive, validate identity, schema, configuration, capability and limits. At execute, run the declared compiled deterministic policy and constraint evaluation mechanism without silently substituting an implementation or profile. At emit, return the artefact plus explicit success, negative, partial, refusal, invalid, failure or unverifiable state. At attach, commit versions, inputs, configuration, raw result and verifier dependencies.

The component's described surfaces are embedded application evaluation, customer-controlled hardware, beside existing systems, isolated execution, benchmark harness, and pairings with Spindle/Numerus/AION. Surface adapters can change transport and ergonomics; they should not change the component proposition. Equivalent supported calls through a library, CLI, service, sidecar, FFI, MCP or WASM surface should reconcile to the same input and artefact identities under the declared mode.



A reusable component-evaluation flow for Lattice.

SURFACE PARITY

Run one supported case through two selected interfaces and compare semantic result, artefact identity, failure class, and evidence, not only transport status.

MECHANISM • RESULT

The artefact manifest is part of the output

The caller needs the Lattice artefact and enough identity to interpret, verify, reproduce, migrate, or reject it later.

The material interface includes source rulebook/version, compiler identity, compiled artefact/checksum, schema, query facts, target/runtime identity, evaluated clauses, result/refusal, explanation or replay path, and optional AION certificate. The exact object names and schemas come from the current project. The architectural requirement is that the result does not lose the input, component, configuration, target, comparison, and verifier information that gives it meaning.

Return work/result and artefact identity; semantic schema and component versions; input references; configuration/profile; runtime/target; state/error; raw and derived outputs; warnings/limits; integrity or signature; evidence references; and compatibility/retention requirements. Keep human-friendly summary separate from machine-verifiable material.

A downstream product may store only a stable reference when the artefact is sensitive or large. The packet manifest must still resolve the authorised object and make missing, withdrawn, expired or incompatible dependencies explicit. Export should preserve the artefact and manifest, not only a rendered report.

COMPONENT RECORD		lattice-artefact
COMPONENT	Lattice + version	PINNED
INPUT	identity + schema	PINNED
CONFIG	profile + target + limits	PINNED
RESULT	compiled policy artefact and query result	TYPED
INTEGRITY	hash/signature where used	CHECKED
VERIFIER	format + runtime + trust	RETAINED
Illustrative manifest; use the implemented component format.		

EXPORT TEST

Export a Lattice result and reproduce the declared check using only the documented artefact, manifest, and verifier dependencies.

MECHANISM • INTERFACES

Integration surfaces must preserve one semantic contract

Lattice can appear as a library, executable, service, sidecar, FFI, MCP or WASM component; each surface is still the same bounded mechanism.

The published description describes embedded application evaluation, customer-controlled hardware, beside existing systems, isolated execution, benchmark harness, and pairings with Spindle/Numerus/AION. For every selected surface, confirm authentication/authorisation where applicable, schema and version, capability discovery, payload/reference limits, memory/time/fuel or resource limits, streaming/order, concurrency, cancellation, errors, retries, idempotency, logs/evidence, packaging, and supported platform/target.

The application should validate the component result before promoting it into product state. Preserve a stable adapter boundary so an implementation can be upgraded or replaced while the caller's semantic contract remains. Provider-specific data belongs in declared extension fields and the manifest, not in hidden behaviour.

Contract tests drive the same corpus through selected surfaces and compare accepted/rejected input, output state, artefact identity, integrity/verdict, resource behaviour, and failure semantics. Transport codes and process exits should map to documented component result classes.

SURFACE QUESTION	REQUIRED ANSWER
Capability	Which operation/version/profile is supported?
Identity	How are input and artefact named?
Limits	Size, time, memory, concurrency, fuel, target?
Errors	Invalid, negative, partial, failed, unverifiable?
Evidence	Which manifest/raw result/verdict returns?
Packaging	Which runtime/dependencies/rights are required?

The same checklist applies across local and service interfaces.

ADAPTER RULE

An adapter may translate representation; it must not silently weaken the component's correctness, determinism, security, or evidence contract.

MECHANISM • COMPOSITION

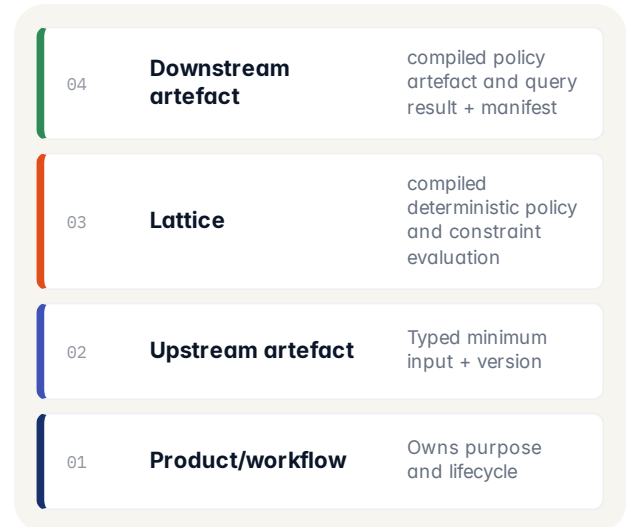
Place the foundation at the correct stack join

A foundation contributes a narrow artefact to products and other foundations; it does not become a product owner merely because it is reusable.

A normal SaaS continuity plan asks whether the vendor will stay available, affordable, compatible, and willing to support the deployment. A Charter licence changes that question by moving direct operation to customer infrastructure. Source escrow, source access, modification, and independent-operation rights remain explicit contract choices.

The upstream caller passes only the minimum input with source, purpose, rights and configuration. The component returns a versioned artefact or reference and state. The downstream caller validates compatibility and may derive new state while preserving lineage. A feedback or update path uses an explicit event/control contract rather than mutating hidden component state.

Mark dependencies required/optional/selected, supported versions, trust/data boundary, resource and determinism effect, packaging and licence, failure mapping, update coupling, and evidence. Then remove or substitute the foundation in a test design to see exactly which promise and object disappear.



A conceptual stack join; exact component dependencies follow the current project manifest.

NO FOUNDATION SOUP

If several foundations appear in a diagram but no one can name the artefact each contributes, the design is branding rather than architecture.

MECHANISM • FAILURE

Failure is part of the component API

The worked fault is: a fact is missing or the artefact checksum/version changes.
 The safe result is: evaluation refuses/escalates or records the new decision identity rather than guessing a default.

The component-specific failure model is: Invalid rule, ambiguous precedence, incompatible schema, missing fact, stale artefact, checksum mismatch, unsupported target, numeric divergence, unavailable owner for a policy conflict, or rulebook change after review must stop or create a new decision state.

Expose invalid input, unsupported capability/configuration/target, resource or dependency failure, integrity mismatch, verification failure, valid negative/refusal, and partial result as distinct states. Include stable work/input identity, failing stage, version/configuration, retry safety, partial artefact policy, diagnostic detail, and evidence. Never encode all conditions as empty output.

Recovery can correct input, select an explicitly approved substitute, retry a safe pure operation, resume from checkpoint, rebuild a derived artefact, escalate to an owner, quarantine the result, or stop. If a component can cause external effect, reconcile it before retry. Preserve failed attempts and manifest changes.

STATE	MEANING	CALLER ACTION
Invalid	Input/contract cannot be accepted	Correct; do not retry unchanged
Unsupported	Declared capability absent	Select approved path or stop
Negative/refusal	Valid mechanism result	Respect outcome/escalate by policy
Partial	Some material unavailable	Use only if contract permits
Failed	Execution/dependency/resource error	Retry/recover by class
Unverifiable	Integrity/check cannot be established	Quarantine/investigate

Exact state names may differ; the semantic distinctions must remain.

FAULT INJECTION

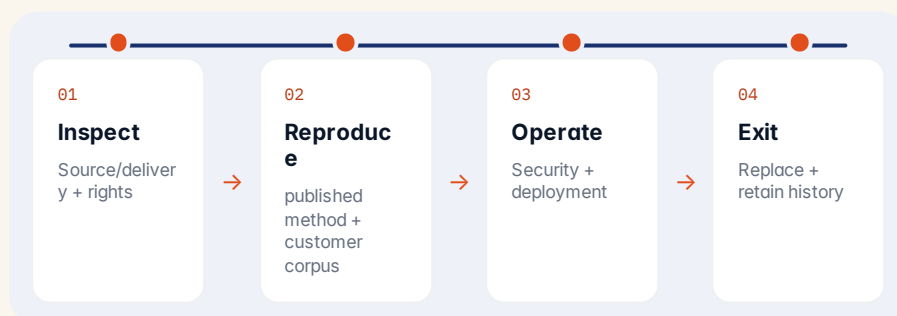
A fact is missing or the artefact checksum/version changes. Verify that evaluation refuses/escalates or records the new decision identity rather than guessing a default.

02

EVALUATION AND ADOPTION

Reproduce, secure, package, integrate, and exit

A foundation becomes production infrastructure only when its narrow claim survives customer inputs, failure, deployment, operations, change, and removal.



The component decision is evidence-backed and reversible.

EVALUATION • REPRODUCIBILITY

Reproduce the Lattice claim, not a summary of it

A credible reproduction pins every identity that can affect the declared proposition and retains raw results, limitations, and negative tests.

The canonical exercise is: Compile the approved rulebook twice, compare artefact identity, run boundary and missing-fact cases on supported targets, switch failover hardware, reproduce the benchmark in its scope, replay a historical query with its old artefact, and alter the checksum.

Capture component source or delivered binary identity, build/provenance and runtime environment, dependencies, input/corpus and rights, schemas, configuration/profile, target/hardware/ISA/backend, warm/cold and concurrency conditions where measured, harness/method, repetitions, comparison rule, raw outputs, verdict, reviewer and date.

First reproduce the narrow published claims under its published scope. Then run the customer's representative corpus and target environment. Include malformed, boundary, unsupported, malicious, resource, dependency, integrity and historical cases. A different workload may produce a different performance or quality result without contradicting the original measurement.

COMPONENT RECORD		lattice-reproduction
COMPONENT	source/binary + version	PIN
INPUTS	corpus + identities + rights	PIN
ENVIRONMENT	target + hardware + runtime	PIN
METHOD	harness + comparison	PIN
RESULT	raw output + verdict	RETAIN
LIMITS	scope + exceptions	RETAIN
A screenshot or supplier summary is not a reproducibility record.		

NEGATIVE CONTROL

A fact is missing or the artefact checksum/version changes. The harness should expose the changed condition rather than still returning a green headline.

EVALUATION • PERFORMANCE

Treat every performance number as a measured boundary

Throughput, latency, memory, recall, token reduction, operation coverage, cold start, or energy figures belong to a named workload and method, not the entire platform.

If the Lattice website description publishes a number, preserve its exact component path, corpus or input distribution, size, hardware and instruction set, software version and configuration, warm/cold state, batch/concurrency, repetitions and duration, metric/percentile, accuracy/recall/correctness condition, resource/power method, baseline, exclusions, raw harness and non-SLA qualification.

A fast component can be a small part of end-to-end time; a compact representation can shift work into conversion; an exact result can cost more than an approximate candidate stage; a cold-start advantage can coexist with a slower hot path. Report the full selected path and counter-metrics rather than promoting one route result into universal savings.

Reproduce relied-on numbers as written, then profile the integrated customer workload under steady, burst and degraded conditions. Include preprocessing, transfer, validation, evidence, stores, retries and caller backpressure. Size from service objectives and measured bottlenecks with recovery and growth headroom.

MEASUREMENT FIELD	WHY IT MATTERS
Workload/corpus	Defines what was exercised
Hardware/ISA/backend	Defines execution path
Config/warm/batch	Changes throughput/latency
Correctness/recall	Prevents speed-only comparison
Baseline boundary	Prevents unlike systems comparison
Raw output/repeats	Shows distribution and exceptions

Use this record for every quantitative claim relied on.

NO BENCHMARK ALCHEMY

A component benchmark can inform architecture; it cannot become a production SLA, total-cost saving, or whole-workflow outcome without additional evidence.

EVALUATION • SECURITY

Security includes the component supply chain and parser/runtime boundary

Open or inspectable code helps review; it does not remove build, dependency, input, resource, capability, key, update, and disclosure risks.

Threat-model the Lattice inputs, parsers/decoders or runtime, dependencies, extension/plugin surfaces, FFI or service boundary, resources, files/network where applicable, artefact stores, signatures/keys, verifier, logs, packages, update channel and support/export path. Treat every external input and produced artefact as untrusted until validated.

Request source/repository or delivered binary identity as applicable, licence/rights, build provenance, dependency manifest/SBOM, compiler/toolchain, signatures, vulnerability and disclosure process, unsafe-code posture where stated, sandbox/capability/resource controls where relevant, hardening/fuzz/property tests, and supported platforms/versions.

Test malformed and adversarial inputs, decompression/size/resource exhaustion, path/URL/redirect escape where relevant, parser ambiguity, FFI misuse, service authentication/authorisation, capability denial, modified package, dependency removal, key rotation, verifier abuse, sensitive error/log output, and air-gap closure.

ORIGIN

Provenance

Source/binary, build, dependencies, signature, licence and release status.

INPUT

Robustness

Malformed, malicious, huge, ambiguous, private, unauthorised and incompatible cases.

RUNTIME

Containment

Capabilities, memory/time/fuel, files/network, concurrency and failure isolation.

OUTPUT

Disclosure/integrity

Schema validation, sensitive logs, artefact access, signing, export and verifier limits.

SECURITY BOUNDARY

Deterministic rule evaluation does not decide that the rule is fair, lawful, current, or applicable. Those facts remain with the policy owner and source-governance process.

EVALUATION • DEPLOYMENT

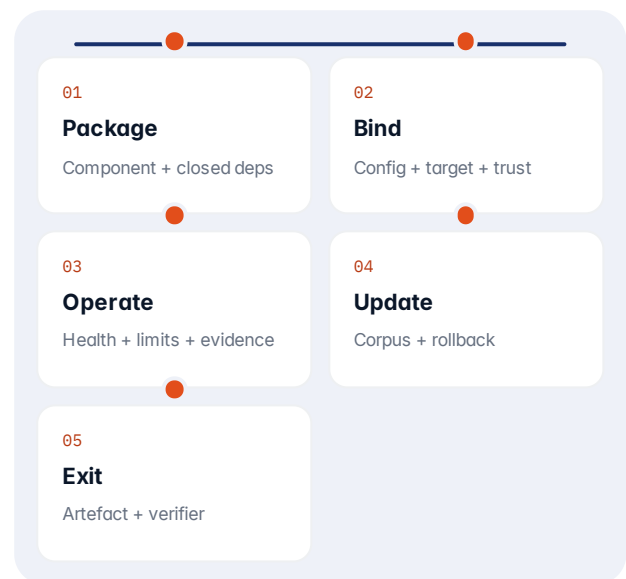
Package and operate Lattice in the selected posture

A foundation may run embedded, as a local executable, sidecar, service, browser/WASM component or another published form; deployment must preserve its contract.

The published surfaces are embedded application evaluation, customer-controlled hardware, beside existing systems, isolated execution, benchmark harness, and pairings with Spindle/Numerus/AION. Build an installed manifest for component, dependencies, runtime/toolchain where required, configuration/profile, platform/CPU features, stores, models/grammars/rules/keys or other data packages, interfaces/ports, limits, health/telemetry, evidence, licence and documentation.

In managed use, define the service and data boundary. In licensed operation, assign customer owners for package, dependency, access, capacity, observability, updates, backup/restore and support. In air gap, close every runtime, licence, trust, verification, update, and diagnostic dependency and validate controlled-media procedures.

Exercise install/cold start, representative load, dependency/resource failure, update and rollback, backup/restore of retained component state, historical artefact verification, no-network operation where required, supplier disconnect, export and decommission. Preserve exact package and environment versions.



Operational lifecycle for Lattice.

AIR-GAP CHECK

A local executable is not proof of air-gap support if it still requires remote packages, identity, time, keys, licence, telemetry, model, or verification services.

EVALUATION • INTEGRATION

Adopt one reversible component path

Start where the foundation replaces a specific opaque, inconsistent, slow, expensive, unsafe, or unverifiable step, and retain the old path until evidence supports the switch.

This page is a product-level security overview. Availability and exact data flows depend on deployment posture, provider, contract and customer configuration. The public Trust Centre holds the current framework and evidence boundary; use the security-report route for responsible disclosure. Each route below answers one question and names its own owner, so a reviewer can go straight to the record rather than back to us for a summary of it.

Integrate behind a versioned adapter; pin component and dependencies; run side-by-side or shadow where possible; compare semantic results and counter-metrics; inject invalid/failure cases; validate artefact/evidence; measure customer hardware and operations; obtain domain/security/technical approval; and cut over segmentally with rollback.

A trace cannot make an external source, model endpoint, human decision or physical device deterministic. It can identify the dependency, capture the response where policy permits, record the version or attestation available and state how replay treats the boundary. Changed or unavailable dependencies are results, not noise to hide.

01

Read

inspect published page, source/delivery, licence, interfaces and tests.

02

Baseline

Measure current component boundary and failures.

03

Integrate

Versioned adapter and minimum data path.

04

Compare

Customer corpus, security, performance and evidence.

05

Cut over

Segmented, reversible, monitored and owned.

REVERSIBLE PATH

At each stage the organisation can remove the component and return to the former contract without losing its data, evidence, or ability to explain work.

EVALUATION • DECISION

Due diligence, portability, and the exit test

The foundation is production-ready only when mechanism, availability, rights, reproducibility, security, operation, historical verification, and removal are evidenced for the selected use.

Request the current Lattice source or delivered artefact status, repository/release and documentation, licence/rights, supported surfaces/platforms, semantic schemas/formats, dependency and build manifest, benchmark/verification harnesses, security evidence, known limitations, roadmap status, support/update policy, deployment package and compatibility.

Run the component-specific exercise: Compile the approved rulebook twice, compare artefact identity, run boundary and missing-fact cases on supported targets, switch failover hardware, reproduce the benchmark in its scope, replay a historical query with its old artefact, and alter the checksum. Record gaps as verified, not applicable with reason, accepted risk and owner/expiry, contractual condition, remediation and date, or a stop decision. Do not infer “shipping”, “source available”, “MIT”, or product inclusion across components from the open-source overview count.

Binaries, model weights, charts, and every dependency bundle into a signed archive, delivered on encrypted physical media through a documented chain of custody. The import script validates checksums before unpacking. Licence validation is offline-signed. No network egress of any kind after install.

COMPONENT RECORD		lattice-decision
MECHANISM	contract + artefact	INSPECT
AVAILABILITY	repo/release/delivery	VERIFY
RIGHTS	licence/use/modify/support	CONFIRM
PROOF	harness + customer corpus	RUN
OPERATION	package/update/recover	PROVE
EXIT	replace + history verifies	PROVE
A technical selection should retain this decision and its evidence.		

EXIT EXERCISE

Archive the Lattice artefact, manifest, and verifier dependencies; reproduce the declared check without a Dweve-operated service and record what remains dependent.

FIND A SUBJECT

Subject index

Page references point to the substantive explanation, worked example, control, boundary, or decision record, not merely to a passing label.

A Adoption 4-5, 13	A AION 8-10, 17	A Air-gapped deployment 17
A Architecture 11, 15	A Artefact identity 3, 8-10, 14, 19	A Assurance 3-4
A Authority 7, 10, 16	A Availability 4, 18-19	B Benchmarking 3-4, 6, 8, 10, 14-15, 17, 19
B Boundaries 3-4, 6, 10-11, 14-19	C Capabilities 8, 10, 12, 16	C Charter 11
C Compliance 4, 6	C Configuration 4, 7-9, 11-12, 14-15, 17-18	C Continuity 11
C Contracts 5-7, 10-12, 17-19	C Cost and budgeting 15	D Dependencies 4-5, 8-12, 14, 16-19
D Deployment 3, 11, 13, 17-19	D Determinism 3-6, 8, 10-11, 16, 18	D Due diligence 19
E Energy 15	E Evidence 3, 5, 7-13, 15, 17-19	E Exit 3, 6, 13, 17, 19
F Failure 5, 8, 10-13, 16-18	G Governance 6, 16	H Hardware 3-4, 8, 10, 14-15, 17-19
I Identity 3, 5, 7-10, 12, 14, 16-17, 19	I Inputs 4-5, 7-12, 14-16	I Integration 3-4, 6, 10, 18
K Keys and signing 9, 16	L Lattice 3-11, 14-17, 19	L Ledger 9, 14, 19
L Licensing 3-4, 11, 16-19	L Limits and exclusions 4, 7-10, 14, 16-17	L Lineage 11

SUBJECT INDEX CONTINUED

Subject index · M–W

Terms are grouped alphabetically. Consecutive page references are collapsed into ranges.

M Manifests 4–5, 7–12, 16–17, 19	M Models and solvers 6–7, 12, 16–19	N Numerus 8, 10, 17
O Operations 4, 10–12, 15, 17, 19	O Outputs 5–7, 9–10, 12, 14–16	O Ownership 6, 11–12, 16, 18–19
P Performance 14–15, 18	P Policies 3–9, 11–12, 16, 18–19	P Portability 19
P Proof 3, 17, 19	P Provenance 4, 14, 16	R Recovery 12, 15, 17
R Replay 3, 9, 14, 18–19	R Reproducibility 3, 6, 9, 13–15, 19	R Responsibility 6
R Retention 7, 9	R Rights 3–4, 6–7, 10–11, 13–14, 16, 19	R Risk 19
R Rules 4, 6, 10, 12, 14, 16	S Security 3, 7, 10, 13, 16, 18–19	S Service levels 4, 15
S Sources 3–4, 6, 9, 11, 13–14, 16, 18–19	S Sovereignty 3, 6, 8, 12, 16, 18	S Spindle 6, 8, 10, 17
S State 6–12, 15, 17–18	S Supply chain 16	T Testing 6–7, 9, 11, 16, 19
T Trace 18	V Validation 15–16, 19	V Verification 3–4, 6–7, 9, 12, 17, 19
V Versioning 3–4, 7, 9–12, 14–15, 18	W Workflows 3, 11, 15	

CLOSING COMPONENT RECORD

The next Lattice action is an independent component proof

Use the real input, configuration, target and failure that matter to the selected workflow. Preserve the artefact and enough dependencies that another authorised team can repeat the check.

Begin with the integration statement: the upstream product/workflow passes a typed input to Lattice for compiled deterministic policy and constraint evaluation; the component returns a compiled policy artefact and query result and explicit state; the downstream caller validates it and preserves the manifest/evidence. Name every owner and dependency.

Run the primary exercise: Compile the approved rulebook twice, compare artefact identity, run boundary and missing-fact cases on supported targets, switch failover hardware, reproduce the benchmark in its scope, replay a historical query with its old artefact, and alter the checksum. Include the negative case, a fact is missing or the artefact checksum/version changes, and confirm that evaluation refuses/escalates or records the new decision identity rather than guessing a default. Capture source/binary, dependencies, inputs/corpus, configuration, target/hardware, method, raw output, verdict, security/failure results and limitations.

Then package and operate it in the intended posture, update and roll back, restore any retained state, verify a historical artefact, export and replace the component, and record what still depends on Dweve. Decide proceed, narrow, remediate, accept with owner/expiry, contract a condition, or stop.

COMPONENT RECORD		lattice-proof
CONTRACT	input + output + failure	INSPECT
ARTEFACT	identity + manifest	RETAIN
REPRODUCE	published method + customer corpus	RUN
SECURITY	abuse + supply chain	RUN
OPERATION	package/update/recover	PROVE
EXIT	replace + history	PROVE
A component proof ends in an auditable selection decision.		

BOUNDARY

Deterministic rule evaluation does not decide that the rule is fair, lawful, current, or applicable. Those facts remain with the policy owner and source-governance process.