VibePackr

Governed AI Execution for Enterprise Work

AI can propose. You govern execution.

Built on BT-OS | English edition | 22 September 2026

ENTERPRISE OPERATING MODEL

From customer context to governed work, evidence, and accountable decisions.

Environment and declarations / Read-only reconciliation / Connection preparation / Headless and human surfaces / Protected execution / Evidence and acceptance

Technical architecture and enterprise evaluation. Bounded implementation is not commercial availability, deployment approval, contractual commitment, or production acceptance.

Read the operating model, not just the interface

VibePackr addresses the connection between enterprise context, a proposed action, permission to act, controlled execution, and evidence. This whitepaper explains how those concerns fit together and where responsibility remains with the customer. It is written for enterprise architects, platform engineers, security reviewers, operators, and business owners.

What the classifications mean

Current bounded implementation means the described function exists within the reviewed technical scope. It does not mean that every depicted combination is integrated, released, available for purchase, or proven in a customer environment. Supported versions, interfaces and target behavior require evaluation-specific confirmation.

Architectural concept means an explanatory or extension role, not a delivered feature. SDK Engine is specifically an extension concept whose current integration status is not established. This is not a statement that it has been removed, deprecated, cancelled, or restricted to the future.

Future hosted/sharing concept describes the broader knowledge service illustration near the end of this paper. It is distinct from bounded local Packtory functionality. Illustrative workflow and target operating model describe discussion aids, not customer deployment or outcome evidence.

How the figures work

Figure 1 is the primary operating model. It separates context and preparation from actual Work execution. Figures 2-7 answer narrower questions. New technical diagrams use explicit labels and editable vector artwork. Three detailed reference plates retain their complete compositions; view them at PDF zoom or print their landscape A3 pages at actual size.

The reading sequence is not a mandatory software pipeline. In particular, declarations and observations can be reconciled repeatedly; preparation does not grant permission; and every Work does not require a new integration or SDK extension.

Scope of this edition

The public site presents a pre-assembly technical evaluation position. This paper does not announce general availability, Marketplace availability, a completed integration catalog or customer production readiness. It describes supported functions and limits rather than elevating architectural intent into delivery evidence.

Sources: [1], [2], [3], [4].

Start with the enterprise. Govern each Work.

An enterprise already has systems, data, APIs, identities, policies, operators and established responsibilities. Adding AI does not replace them. The practical question is how an AI-assisted request can use an authorized part of that environment without treating model capability as permission.

VibePackr describes a governed execution control plane built on BT-OS. Its operating model starts by making relevant context explicit, distinguishes declared expectations from observations, prepares supported connections, and directs requests through a protected execution boundary. The result is a reviewable Work record, not merely a plausible answer or an activity log.

Headless is the default posture

The core operating posture is headless: machine-driven work and typed operational queries do not require a continuously open graphical client. PackrGUI is an on-demand human interaction and review surface. Administration and status have separate scopes. None of these surfaces becomes an independent permission engine.

This does not imply universal automation coverage. A supported request path, suitable configuration and the relevant authority must exist. A schedule or external event is an initiation pattern, not proof of a shipped scheduler or connector for every enterprise system.

Preparation and execution answer different questions

Technical declarations describe relevant identities, interfaces, policies and constraints. GAPE reconciles bounded source observations and declarations into reviewable projections. Integration Helper prepares candidates from supported metadata. These functions help establish what a connection means; they do not authorize the next action.

The protected core evaluates the Work request within its applicable scope, controls supported execution and records validation and evidence. A technical receipt remains distinct from business acceptance. SDK Engine is not a required component on this current path: its extension role is described separately, with current integration status unconfirmed.

Why this can remain a product model

Structured declarations, bounded preparation and common execution controls allow customer-specific context to be handled without assuming that the supplier owns the customer's implementation or operations. This is the architectural basis for Productized Assistance and a Non-SI model, not a promise of zero integration effort.

Sources: [1], [2], [6].

Make the starting conditions explicit

The starting point is an authorized slice of the customer's environment: enterprise systems, application interfaces, data sources, runtime resources, identity and access arrangements, policy documents, and the people accountable for their use. An architecture drawing does not create access to any of them.

Select the scope before collecting context

For a reporting workflow, identify the intended source, allowed period, permitted output, responsible reviewer and excluded operations. For an operational workflow, identify the target, allowed change, relevant dependencies and recovery expectations. These are illustrative scoping questions, not a claim that every database or operational technology platform is supported.

Customer documents can describe intended rules; configuration can describe a selected state; runtime observations can show a narrower actual state. Keep these sources identifiable. A document that mentions an API is not evidence that the API is reachable, compatible or authorized for this Work.

Existing controls remain relevant

Identity and access management (IAM) establishes actors and access boundaries. Gateways may control endpoints and traffic. Orchestration arranges steps. Work governance connects a particular intent, scope and permitted operation to execution and evidence. Responsibilities can overlap across products; this is not a claim that other products lack governance.

The customer's environment also contains paths that may bypass VibePackr. Only supported requests routed through the governed path are within its stated execution-control coverage. Importing an external log does not retroactively authorize the operation it describes.

Ownership does not move with the data

The customer retains responsibility for source rights, classification, permitted use, policy choices and business objectives. A technical ability to read material is not permission to process or redistribute it. Use only the inputs required for the authorized evaluation and keep confidential material out of public architecture examples.

Sources: [1], [2], [6], [10].

Contract / Document: declare meaning before action

In this paper, Contract / Document means technical declarations and their supporting documentation. It does not mean a service agreement, legal interpretation engine or automatic conversion of contractual prose into enforceable product policy.

From human intent to explicit technical context

Relevant declarations can identify a runtime or interface, the scope in which it is used, required capabilities, data classification, policy references, approval requirements and knowledge-source provenance. Supporting documents explain why these declarations are appropriate. A responsible owner must resolve unclear or conflicting meaning.

The bounded implementation includes structured runtime, model, knowledge and binding definitions. That supports a concrete declaration mechanism. It does not establish a complete product suite that autonomously ingests arbitrary enterprise manuals, extracts every obligation and configures integrations without review.

Four different objects

A declaration states intended or configured context. An observation records what a bounded source reports. A binding associates specific supported identities and revisions under the applicable rules. An authority decision determines whether a particular operation may proceed. A declaration is not any of the other three.

Generated configuration or documentation is also not necessarily applied configuration. Version, provenance, approval and the effective target state matter. Changing a document alone does not change execution policy.

Relationship to GAPE and preparation

GAPE can compare revision-bound facts from contracts, registries, configuration, runtime and other authorized sources. This makes disagreement visible without rewriting the declaration or taking over its owner's authority. Integration preparation can use supported declared metadata to describe a candidate interface. Neither relationship proves a universal document-to-connector-to-execution pipeline.

The useful question is not whether a document was imported. It is whether the required technical meaning is explicit, current, attributable and usable within the selected product scope.

Sources: [2], [6], [9].

GAPE: compare declarations with observations

The Governed Architecture Projection Engine (GAPE) provides bounded, read-only architecture reconciliation and projection. It keeps observed facts, candidate facts, reconciled facts and comparisons with authority references distinct. Its outputs help a reviewer understand the environment; they do not permit an operation.

Supplied sources, explicit provenance

The current bounded model accepts revision-bound observations and references. Sources can represent contracts, registries, configuration, runtime, evidence and user-declared context within supported input forms. Source permission and the authenticity and relevance of those inputs remain prerequisites.

The presence of a source category is not proof that a connector automatically discovers every matching enterprise system. This edition does not claim arbitrary network scanning, complete environment discovery or automatic drift remediation.

Reconciliation does not hide disagreement

A declared owner, observed relationship and effective authority reference may disagree. GAPE can retain the difference, provenance and uncertainty in architecture, dependency, runtime, authority, evidence and drift projections. A stale or insufficient observation must not become current truth because it appears on a polished map.

The same principle applies to evidence. GAPE can consume relevant references; it does not become the owner of the execution record, issue permission or manufacture acceptance. A supplied record and an always-on observation feed are different integration claims.

The relationship with Integration Helper

The bounded Helper implementation can turn prepared-candidate provenance into GAPE observation inputs. This supports a preparation-to-projection relationship. It does not establish an automatic reverse path in which GAPE configures a connection or enables execution. People can use projections to review and improve declarations and preparation choices; that informational loop is not a runtime authority edge.

Figure 3 preserves the detailed public reference composition. Its viewer and broader workflow are conceptual; the current bounded claims are those stated here.

Sources: [6], [9].

Integration Helper: prepare, then request consideration

Integration Helper is a bounded preparation function. It consumes supported declared metadata, normalizes observations, validates technical prerequisites and creates a revision-bound candidate. This is a useful software role even when registration or execution must still be decided elsewhere.

What preparation produces

The candidate identifies the relevant source, schema, capability and adapter revisions, with provenance and a preparation result. Its handoff asks the protected Packr Engine owner to consider that candidate; this identifies responsibility, not a proven live registration interface. A successful preparation result is not a statement that the interface has been registered, bound to a Work, authorized or executed.

Bounded support includes declared metadata associated with HTTP interfaces, command-line tools, local executables, local communication interfaces, structured manifests and existing runtime-provider metadata. These categories are not a compatibility catalog. Support grade and accepted input form must be checked for the actual evaluation.

What it does not do

Reading declared command metadata is not permission to run an arbitrary command. Observing interface metadata is not a network scan or proof of connection readiness. The reviewed preparation boundary does not own registration, eligibility, binding, execution authority, receipts or closure.

Additional protocol support, generated scaffolding or a later implementation cannot be assumed to be part of the selected product configuration. If custom development is required, the customer or appointed implementer must establish it within the supported extension boundary and test the result.

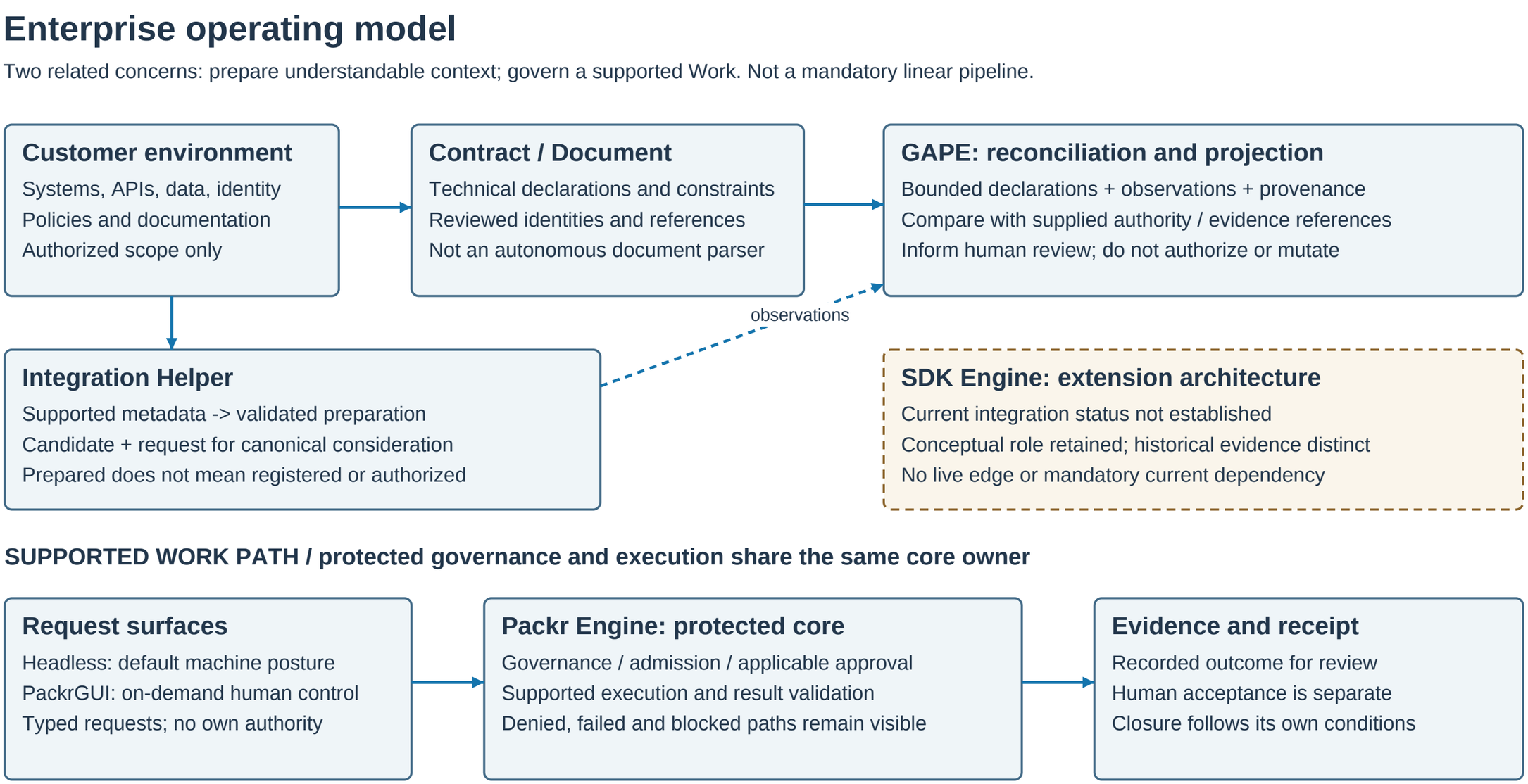
No missing bridge is assumed

The prepared-candidate handoff and a supported Work request are different paths. This whitepaper does not fill the gap between them with an invented live connection. Independent admission, configuration and binding requirements still apply before a capability can be used for Work.

This is why connection preparation is Productized Assistance rather than turnkey systems integration: the product can standardize the preparation boundary without taking ownership of every customer's implementation project.

Sources: [2], [6], [8].

Figure 1. Enterprise operating model



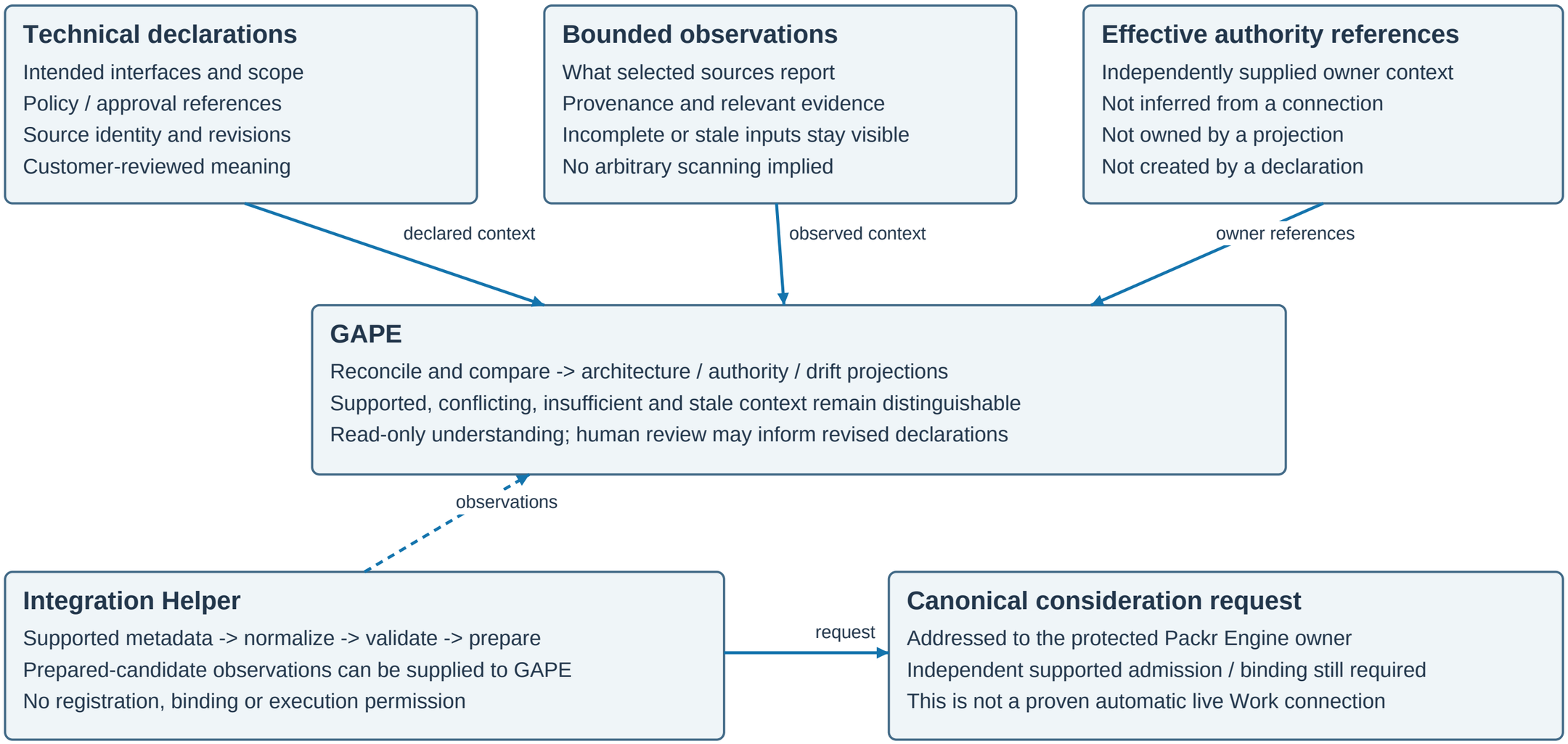
Primary model of current bounded roles, not a deployment or released-package inventory. Context/preparation and Work execution are separate paths. Arrows distinguish bounded inputs, supplied observations and supported Work flow; none transfers authority. Preparation requests consideration rather than enabling Work automatically. The protected core owns governance and execution. SDK Engine is an isolated architectural extension concept with current integration status not established. No live SDK edge is asserted.

Related public explanation: https://www.vibepackr.com/resources/system-overview/

Figure 2. Declaration, observation and preparation are different

Declarations, observations and preparation

Inputs have different meanings. Reconciliation and candidate preparation do not create execution permission.

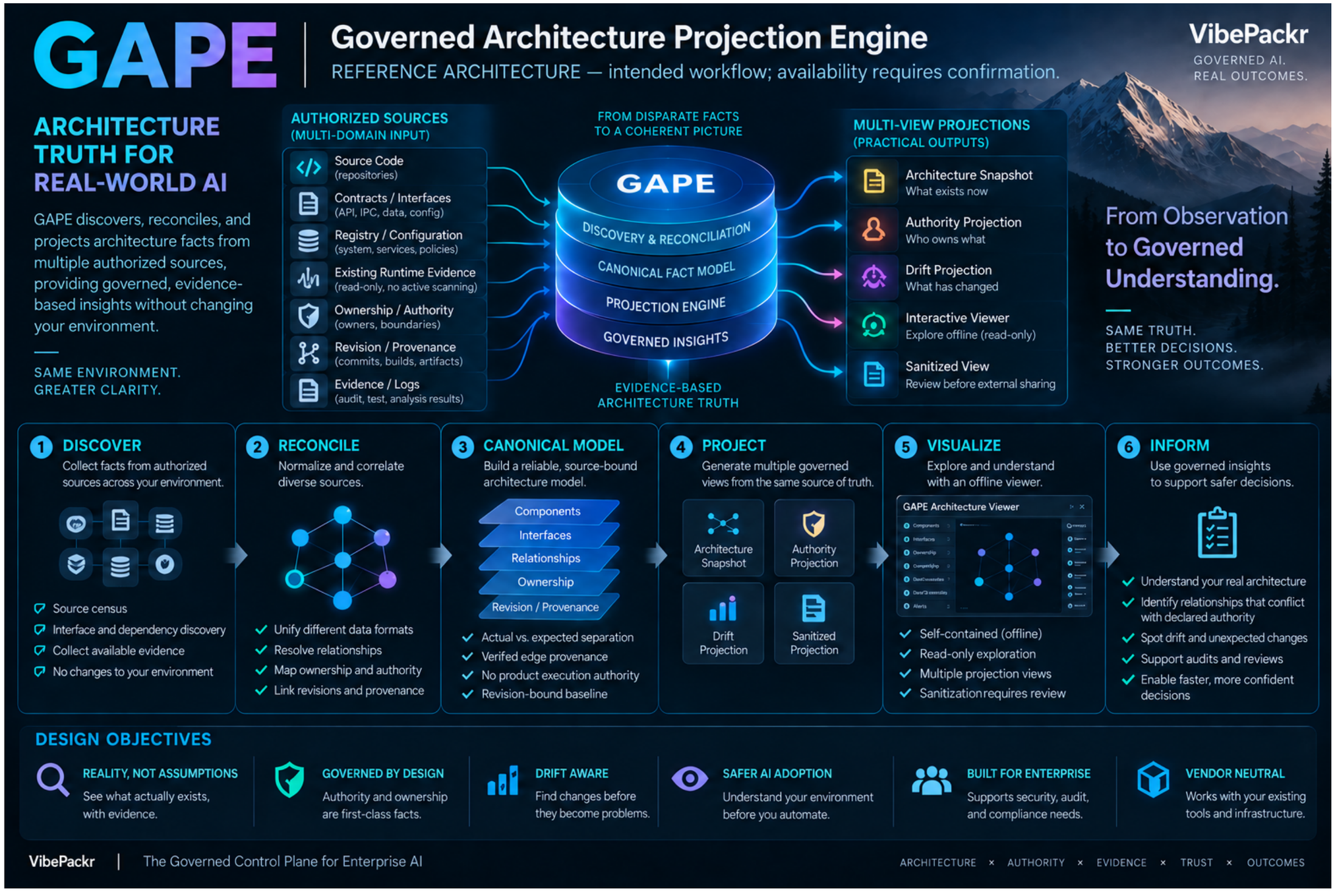


Declaration != discovery != preparation != registration != binding != authority != execution

Drilldown into the preparation boundary. Technical declarations are not legal instruments or automatic discovery. GAPE compares supplied inputs; Integration Helper prepares supported candidates and can supply observations to GAPE. The human review loop is informational, not automatic mutation. A consideration request is not registration, binding or authority.

Related public explanation: https://www.vibepackr.com/architecture/

Figure 3. Detailed GAPE architecture reference



Conceptual reference, retained in full. The current bounded implementation is revision-bound input reconciliation and projection; this plate is not proof of universal source adapters, arbitrary active discovery, an available interactive viewer, automatic remediation or complete enterprise coverage. Projections inform review and never own execution authority. Review any output before sharing it.

Related public explanation: https://www.vibepackr.com/resources/gape-context/

One core. Different ways to interact.

Headless is the default operating posture, not a second execution engine competing with Packr Engine. It describes operation without a continuously open graphical client. Supported machine requests and typed queries reach the core through defined interfaces; the protected decisions remain in the core.

Machine-facing operation

Automation can consume supported typed results without reading a visual dashboard. Bounded operational queries include status, health, Work inspection and local asset/trust status. Read-only queries do not create Work or authorize changes. A machine-readable result is not evidence of a general public HTTP API or compatibility with every scheduler.

Work initiation and operational observation must be distinguished. A system submitting a supported Work request needs the applicable identity, scope, configuration and authority. A status consumer only observes the bounded information returned to it. Neither obtains more authority because no human is watching a GUI.

PackrGUI for human intent and review

PackrGUI provides the on-demand human surface for requesting, inspecting and reviewing Work. It presents product-owned state and sends supported actions; it does not reconstruct its own permission rules or turn a displayed success into acceptance.

Human and machine interactions share the protected authority boundary. This is a governance relationship, not a claim that every operation or control is identically exposed on every client. Confirm supported operations and platform behavior for the evaluation.

Client placement and deployment

The enterprise target model separates the headless appliance from human and administration clients. PackrGUI need not reside inside the appliance or stay open for its core operation. Customer-controlled and single-tenant describe an ownership/isolation objective, not one physical host, one user or a completed customer deployment.

Sources: [2], [5], [6].

Administration is not Work authority

Several surfaces may display similar information while serving different purposes. They must not be collapsed into a generic administration box that appears to own every decision.

Work interaction: PackrGUI

This is the human interaction and review surface for Work. Submitting intent, inspecting progress and considering a result are distinct from changing appliance configuration or governing the reusable-asset lifecycle.

Appliance administration: VMAdmin

VMAdmin names the appliance-operation role. The dedicated Administrative Console is a client surface for bounded, authorized administration such as configuration, service operations, diagnostics and supported recovery. The core owns the applicable decisions. An administrator's ability to operate the appliance is not blanket permission to execute every Work.

Asset administration: PacktoryAdmin

PacktoryAdmin concerns the bounded local asset lifecycle: governed publication, asset management and recovery-related functions within supported scope. It is not a second Work admission engine. Asset availability, eligibility for use and permission to execute remain different questions.

Status and health

Status is a typed observation of core-owned state. Health separates a live process from readiness to perform a supported operation. Historical successful Work does not prove present readiness, and a healthy runtime does not prove that the authority path is ready.

Some components or operations can be unavailable, unknown or deferred. A route label in a console is not proof that its full capability is implemented. The software should expose these limits rather than infer success from a menu, a configuration file or an earlier receipt.

These distinctions give operators a clearer responsibility model: observe the right scope, use the appropriate authorized operation, and reconcile the returned state. Do not treat a display or administrative convenience as a bypass around Work governance.

Sources: [2], [5], [10].

Permission belongs to the governed Work

A model can propose a plan. A prepared integration can describe a possible capability. An authenticated actor can reach an interface. None of these facts alone authorizes the proposed operation.

The protected Packr Engine owns the relevant governance and execution decisions. Policy, approval, runtime binding, supported capabilities and the request's scope constrain whether Work proceeds. Governance and execution are shown as different functions for clarity, not as independent authorities that can contradict or override each other.

The request is specific

Intent must identify enough purpose and scope to be considered. Admission can permit, deny or require approval. Approval does not override applicable policy. The runtime requested by a caller is not necessarily the runtime actually bound under the supported rules. Availability or selection does not, by itself, prove execution permission.

Supported execution, not universal interception

For authorized Work, the product controls the supported execution path and records its state. Requests outside that path are not brought under governance merely because they use the same model, endpoint or enterprise data source. Existing IAM, gateway and application controls remain part of the surrounding system.

The lifecycle includes incomplete, denied, interrupted, failed and blocked outcomes. They are meaningful results, not exceptional cases that can be hidden behind a success label. A missing prerequisite or missing validation evidence limits the conclusion even if an output file exists.

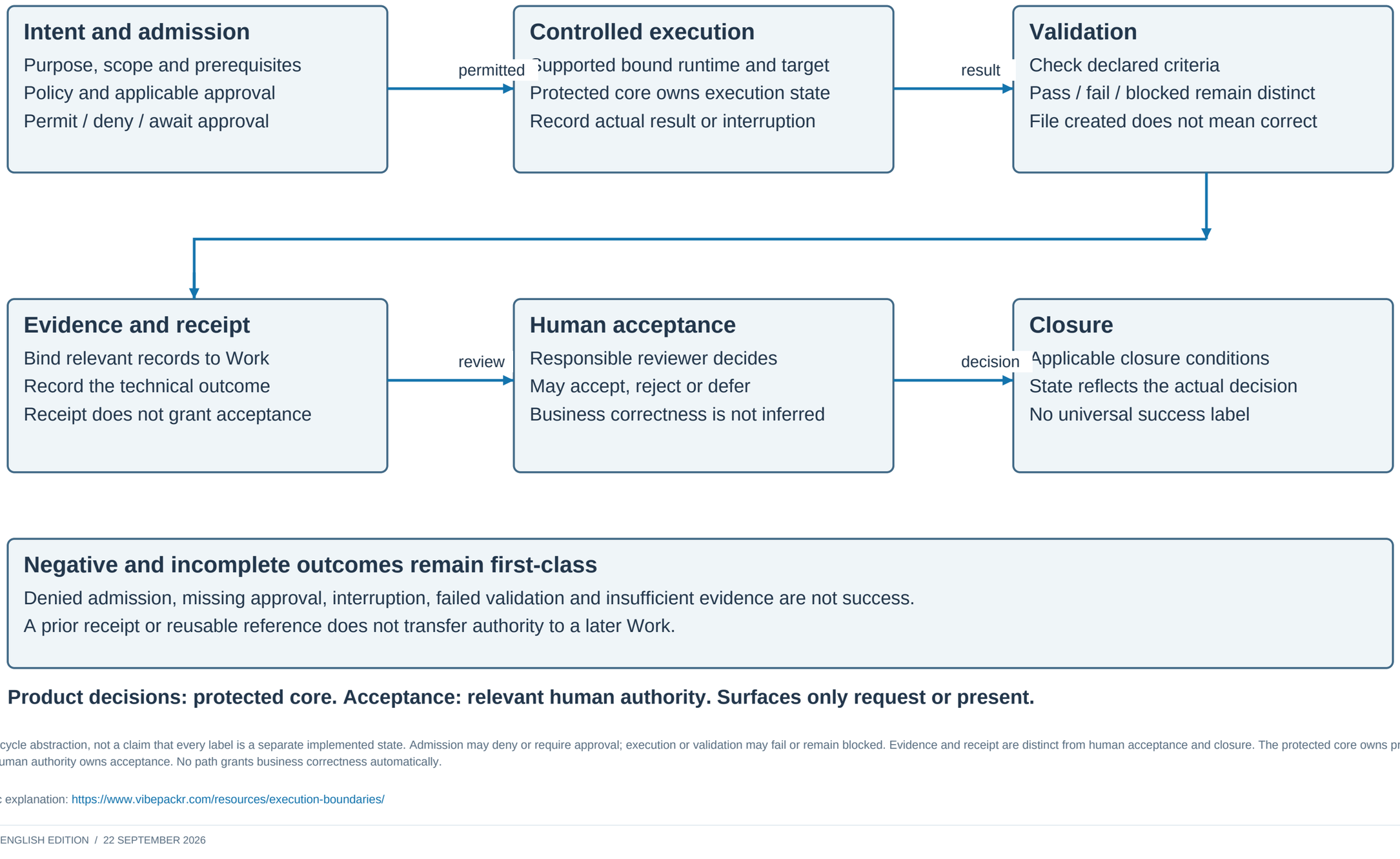
An illustrative reporting request

For "Generate the September financial report," define an authorized source, period, allowed transformations and permitted output. A valid reporting request does not grant permission to alter the source system or export unrelated records. Execution evidence supports review of the operation; the finance owner still decides whether the report is correct and acceptable.

This example is not an enterprise resource planning (ERP) connector claim, customer case study, finance certification or production record. Figure 4 abstracts the bounded state progression rather than listing every implementation state.

Sources: [1], [2], [3], [7].

Figure 4. Work progression and separate acceptance



Functional lifecycle abstraction, not a claim that every label is a separate implemented state. Admission may deny or require approval; execution or validation may fail or remain blocked. Evidence and receipt are distinct from human acceptance and closure. The protected core owns product state; the relevant human authority owns acceptance. No path grants business correctness automatically.

Related public explanation: https://www.vibepackr.com/resources/execution-boundaries/

Evidence supports a decision. It does not make it.

A useful Work record connects intent, admitted scope, the selected execution context, what occurred and the checks that were applied. These relationships help engineering, security, operations and business reviewers ask different questions of the same bounded activity.

Keep the objects distinct

An observation reports a signal. Evidence binds relevant records to a claim or Work. Validation compares results with declared criteria. A receipt records a technical outcome. Acceptance is the responsible authority's decision. Closure records the resulting state under the applicable process.

Producing evidence does not prove that it is sufficient. Issuing a technical receipt does not automatically close the Work as accepted. The lifecycle explicitly allows a result to await human acceptance; incomplete or rejected outcomes remain visible.

Integrity is not universal truth

Digests and provenance can help bind identity and detect changes. They do not independently prove that a source was truthful, that every action was observed, that a business result is correct or that a regulatory obligation was met. The strength of the conclusion depends on the evidence, criteria and scope actually reviewed.

Views do not duplicate authority

An engineer may inspect execution and validation references, an operator may inspect current readiness and a business owner may review the result. A projection for one audience is not a new canonical record or an independent approval engine. The same applies to GAPE's evidence-related views.

Appropriate sharing also requires review. Raw logs, credentials, private topology and customer content are not made publishable simply by appearing in an evidence package. The record should be fit for its intended recipient and bounded purpose.

Sources: [1], [2], [3], [9].

SDK Engine: an extension concept, not a required step

SDK Engine describes an architectural role for development and extension: declared interfaces, integration patterns, guidance, testing, version compatibility and reusable components. That role is useful to explain, but its current integration into the Enterprise Product is not established for this edition.

Preserve the distinction

Historical implementation and execution artifacts exist. They are not proof that the historical path is fully integrated into the present enterprise operating model or shipping as a complete capability. Nor does the existence of a public programming interface establish that the component named SDK Engine has that current role.

This paper therefore does not place SDK Engine on the mandatory current path. It does not assert a live integration with Integration Helper, Headless, Packr Engine or authority surfaces. Figure 1 places it in a separate concept box with no live connection arrow.

What extension would still require

For an actual extension, developers must establish the supported interface, implemented behavior, version compatibility, testing and governed handoff. Generated scaffolding is not a finished integration; a successful build is not permission to execute; a test artifact is not proof of a supported customer deployment.

Any future classification change requires current, approved integration evidence. This is not a declaration of removal, deprecation, cancellation or future-only status. It is a precise limit on what can presently be claimed in this whitepaper.

Responsibilities remain with named owners

Customer-specific adapter implementation and workflow integration require assigned owners. Productized tools can reduce repeated preparation work without promising that no engineering is needed or transferring the customer's implementation and operational responsibilities to the supplier.

Source: [8] describes the broader architectural concept, not evidence of current integration.

Standardize the boundary, not every customer's project

Productized Assistance means standardized software functions and guidance that help customers understand context, prepare supported connections, evaluate behavior and operate the agreed product. It does not mean that the vendor undertakes every systems integration (SI) project.

The architecture explains the responsibility split

Technical declarations make customer-specific scope explicit. GAPE makes bounded observations and disagreements inspectable. Integration Helper produces candidates without granting execution permission. Headless and human surfaces use the same protected decision owner. Evidence supports review without taking over business acceptance.

This separation makes a common product plus governed configuration and supported adapters a coherent design objective. It does not prove that every customer requirement is configurable, that custom development is unnecessary, or that a customer-specific fork can never be needed.

Customer and implementer responsibilities

The customer assigns business objectives, policy, data rights, source accuracy, access, credentials, infrastructure operation and final business decisions. Custom adapters, migrations, workflow design, production cutover and training require their own implementation owners and scope.

Product Support concerns included components and agreed supported behavior under applicable terms. A problem discovered during integration can still be a product defect; the Non-SI boundary is not a reason to classify every integration-related issue as outside product responsibility. Included Official Packs and customer-authored Packs should not be assumed to have identical support scope.

No contract is created by this explanation

This paper does not establish support response times, remedies, warranties, liability allocation, managed operations or legal approval. The current public SLA is a draft requiring the applicable review and execution process. Its publication does not activate support commitments. Applicable law and executed agreements govern obligations.

Third-party models, infrastructure and applications retain their own dependencies and terms. Examples and logos in reference material are not endorsements, partnerships or compatibility certifications.

Sources: [10], [13], [14].

Figure 5. Customer, product and third-party responsibilities

Reference responsibility model, not a contract or deployment inventory. Customer implementation and operation remain assigned responsibilities; product controls apply within supported scope. The administration label identifies client access, not a GUI that must be installed in the appliance. Product defects remain subject to applicable support scope. No outcome, compliance or managed-service guarantee is implied.

Related public explanation: https://www.vibepackr.com/resources/shared-responsibility/

Recover state and authority, not just compute

The target enterprise model separates replaceable execution compute, persistent system and evidence data, access dependencies and tested backups. Recreating a virtual machine (VM) is not the same as recovering the correct state or establishing permission to resume Work.

Installation is not readiness

The customer or appointed operator must establish the supported environment, identity and access arrangements, selected runtime, intended policies and scope. A process that starts may still lack a usable runtime or current authority. An old successful receipt does not prove present readiness.

Control-layer sizing must be distinguished from model inference requirements. The existence of a lightweight control function does not mean the selected models have no separate compute, memory or accelerator needs.

Restart and replacement

A restart requires health and readiness checks, configuration and identity verification, supported reconnection and validation before resumption. Replacement additionally depends on compatible compute and the correct persistent data, credentials, keys and recovery procedure. Persistent storage alone is not a backup.

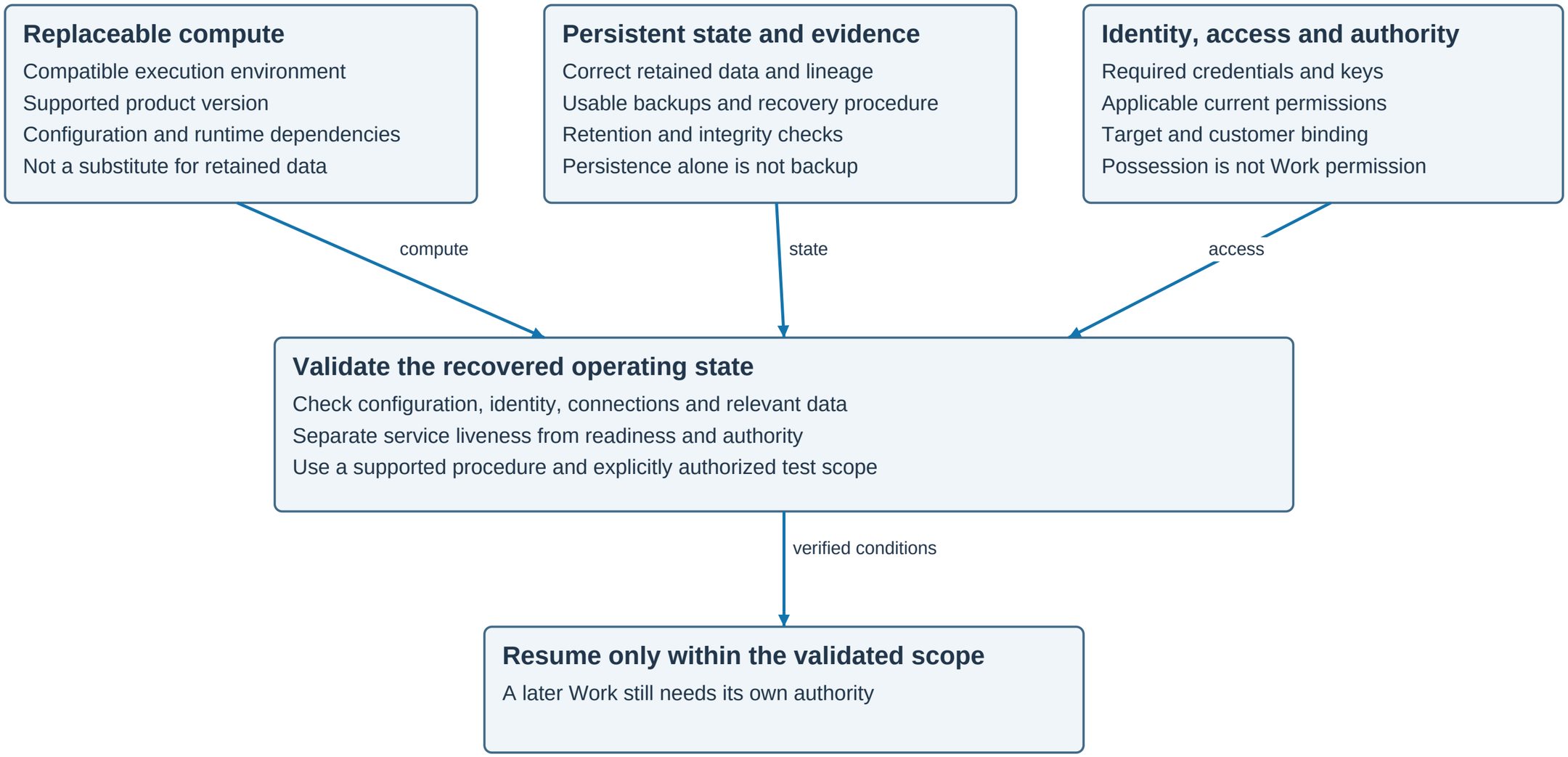
Bounded recovery mechanisms are not evidence that a particular customer restore has succeeded. Define the selected interruption scenario, required records and acceptance criteria before testing. Destructive or production-impacting exercises require their own authorization.

Figure 6 is a dependency model

It shows prerequisites and verification, not a complete installation recipe or guaranteed recovery sequence. No recovery-time, recovery-point, uptime or zero-loss commitment is made. SDK Engine is not a mandatory installation or recovery dependency in this edition.

Sources: [2], [11], [13].

Figure 6. Dependencies for recovery and resumption



Target operating model. Replaceable compute, persistent state, backups, access and compatible versions are separate dependencies. Validation precedes resumption; an installation or restored file alone is insufficient. No customer restore proof, deployment authorization, automatic recovery, uptime or zero-loss promise is represented.

Related public explanation: https://www.vibepackr.com/resources/recovery-boundaries/

Reuse useful work without inheriting permission

A result is more useful when its purpose, provenance, conditions, evidence and limitations remain available. But recorded execution, distilled knowledge, a reusable Pack and a later authorized Work are different objects.

Bounded local functionality

Packtory includes bounded local asset discovery, consumption and governed administration. A Pack represents governed execution knowledge within its applicable scope. These functions are distinct from the wider hosted and sharing architecture in Figure 7 and from commercial service availability.

A potentially relevant reference is not automatically eligible. Rights, sensitivity, versions, target conditions and applicable policy must be considered. Selection, recommendation, binding and execution authority remain separate decisions.

Feedback is not self-authorization

Evidence from earlier work can inform a later candidate or review. It cannot transfer the original permission to another customer, target or operation. A successful historical execution does not remove the need to evaluate present requirements and authority.

Likewise, semantic relevance is not permission to retain, process or redistribute customer material. A knowledge mechanism should preserve the distinctions among customer data, operational records, reusable knowledge and shareable assets.

Future sharing remains separate

The preserved appendix plate describes broader hosted storage, search, distribution and sharing concepts. Its categories and arrows are not proof of an available remote service or a general right to ingest customer work. Remote Rescue transport remains disabled. There is no delivery date, measured learning gain or automatic knowledge-compounding claim in this paper.

Source: [12] for the future sharing concept, not an exhaustive inventory of implemented local functions.

Evaluate dependencies, then expand deliberately

A first evaluation should establish one useful governed workflow and its limits. The sequence depends on what the customer already has; it is not a fixed six-stage delivery schedule with an obligatory SDK extension.

Establish the record

• Environment: authorized sources, target resources, owners, data rights and excluded actions.

• Declaration: relevant technical context, identities, policy and approval references, versions and unresolved meaning.

• Reconciliation: bounded observations, provenance and disagreements; no assumption of complete discovery.

• Preparation: supported interface metadata, candidate status, integration owner and independent binding requirements.

• Surface and execution: supported machine or human entry, applicable authority and denied or incomplete cases.

• Evidence and operations: result checks, receipt versus acceptance, current readiness, recovery responsibilities and review access.

Include negative and incomplete cases

A useful evaluation checks more than a successful demonstration. Consider missing approval, out-of-scope requests, unavailable prerequisites, denied execution, failed validation and incomplete evidence. The exact tests must fit the supported workflow; this list is an evaluation aid, not a statement that tests have already run.

Extension is conditional

An existing supported path may need no new adapter. A different use case may require engineering or an unavailable capability. SDK Engine's current integration status remains unconfirmed; do not make it a required adoption stage. Additional systems, protocols or wider data access are separate scope decisions.

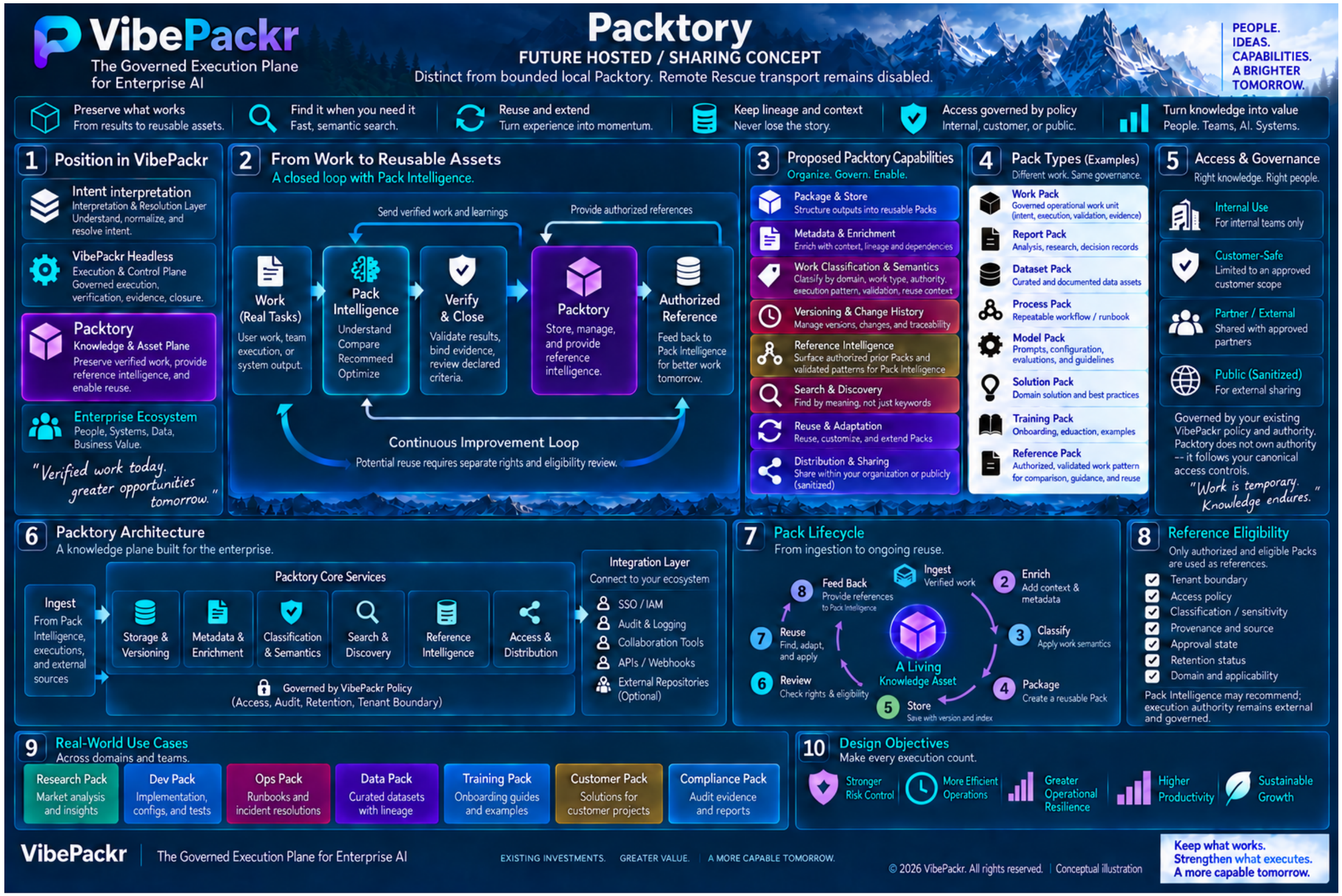
End with an explicit decision

Record what was demonstrated, which versions and conditions were used, which limitations remain and who may accept or expand the scope. A technical demonstration is not automatically production readiness, commercial support or customer acceptance.

Contact support@vibepackr.com to discuss a bounded evaluation. Describe the tools, actions requiring approval and execution environment; do not send credentials, confidential logs or customer data in the initial inquiry.

Sources: [1], [3], [15].

Figure 7. Optional future hosted and sharing architecture



Future hosted/sharing concept appendix, retained in full and distinct from bounded local Packtory implementation. The plate's Headless label denotes the operating surface, not a second authority or execution engine. Proposed loops concern candidate/reference flow, not inherited permission. Rights, scope and eligibility require independent review. This is not the primary operating model, a service offer or a release commitment; remote Rescue transport remains disabled.

Related public explanation: https://www.vibepackr.com/resources/knowledge-concept/

Sources and edition notes

Public references explain the product narrative and conceptual material; they are not independent implementation or production certifications. Related web figures may use broader architectural labels. This edition's classifications and captions delimit its claims. Exact availability and supported evaluation scope must be confirmed separately.

[1] Overview - https://www.vibepackr.com/

[2] Architecture - https://www.vibepackr.com/architecture/

[3] How It Works - https://www.vibepackr.com/howto/

[4] Reference Library - https://www.vibepackr.com/resources/

[5] Layered architecture reference - https://www.vibepackr.com/resources/iceberg-architecture/

[6] Enterprise system reference - https://www.vibepackr.com/resources/system-overview/

[7] Execution reference - https://www.vibepackr.com/resources/execution-boundaries/

[8] Extension concept - https://www.vibepackr.com/resources/integration-extension/

[9] GAPE reference - https://www.vibepackr.com/resources/gape-context/

[10] Responsibilities - https://www.vibepackr.com/resources/shared-responsibility/

[11] Recovery reference - https://www.vibepackr.com/resources/recovery-boundaries/

[12] Future sharing concept - https://www.vibepackr.com/resources/knowledge-concept/

[13] Public SLA draft - https://www.vibepackr.com/legal/sla/

[14] Legal index - https://www.vibepackr.com/legal/

[15] Evaluation reference - https://www.vibepackr.com/resources/evaluation-path/