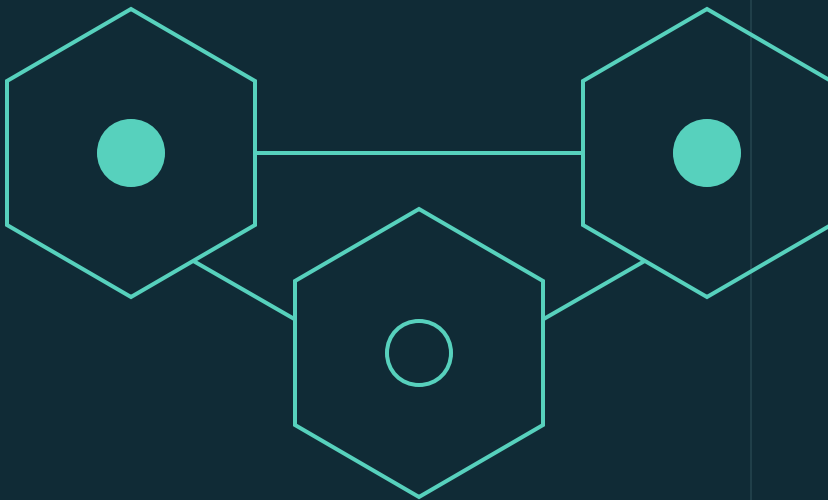

AN OPEN ENGINEERING BOOK

AgentPlane

Building a Kubernetes-Native
Platform for AI Agents



Security boundaries. Reliable execution.
Tenant isolation. Tool governance.

AgentPlane Book contributors

ENGLISH EDITION 0.1.0 / OCTOBER 2026

AgentPlane

Building a Kubernetes-Native Platform for AI Agents

AgentPlane Book contributors

English edition · October 10, 2026

Original narrative, diagrams and prompt prose are licensed under Creative Commons Attribution-ShareAlike 4.0 International. Original program code and configuration examples are licensed under MIT. See the repository license files for scope and attribution.

This independent educational book develops a proposed reference architecture. It is not a released AgentPlane service, an independent security audit, a compliance certification or official upstream documentation.

Teaching examples and actual checks are identified in the companion repository. Database, cluster and cloud exercises need separate execution in authorized disposable environments. No customer or benchmark results are invented.

Source Markdown, diagrams, code, labs, implementation assignments and build scripts accompany this edition. No public repository address is assumed.

Read the boundary. Test the invariant. Record the evidence.

Contents

OPENING

Preface: A Platform Is a Set of Boundaries	5
--	---

PART I — FOUNDATIONS

01. Choose a Product Boundary Before a Technology Stack	7
02. Separate the Control Plane from the Execution Plane	10
03. Turn the Threat Model into Testable Invariants	14
04. Create a Reproducible Engineering Workspace	17

PART II — CONTROL PLANE

05. Design Tenant-Safe PostgreSQL Data	20
06. Keep Human, Service and Cluster Identities Distinct	23
07. Design APIs for Work That Finishes Later	26
08. Enroll Clusters Without Exporting Their Authority	29
09. Make Delivery Durable Without Promising Exactly-Once Effects	32

PART III — KUBERNETES INTEGRATION

10. Use Kubernetes Reconciliation Without Expanding Authority	36
11. Integrate Agent Sandbox Through a Capability Adapter	39
12. Build Runtime Profiles for the Actual Threat Model	42

PART IV — SAFE EXECUTION

13. Model Sessions, Executions and Uncertainty Separately	45
14. Execute Commands and Handle Files Without Hidden Privilege	48
15. Make Network Policy an Enforced Property	51
16. Broker References, Not Customer Secret Values	54

PART V — TOOL GOVERNANCE

17. Discover Tools Without Trusting Their Descriptions	57
18. Bind Authorization and Approval to the Exact Action	60

PART VI — STATE AND EVIDENCE

19. Preserve Workspaces Without Inventing Checkpoint Guarantees	63
20. Meter Usage and Reserve Capacity Without Double Counting	66
21. Observe the System Without Collecting the Customer	69

PART VII — PRODUCT AND OPERATIONS

22. Build Interfaces That Preserve Scope and Uncertainty	72
23. Deploy and Upgrade Without Crossing the Trust Boundary	75
24. Design Recovery Before the First Production Incident	78
25. Build Evidence for Every Important Claim	81

PART VIII — IMPLEMENTATION AND PUBLICATION

26. Use Coding Agents as Bounded Engineering Collaborators	84
27. Deliver One Complete Vertical Slice	87
28. Maintain an Open Book Like a Software Project	90

APPENDICES AND SOURCES

Appendix A. A Working Vocabulary	93
Appendix B. Decision Records and Interface Contracts	95
Appendix C. Review Answers and Failure Scenarios	98
Appendix D. Companion Files and Build Commands	100
Primary Sources and Verification Notes	102

Preface: A Platform Is a Set of Boundaries

An AI agent that only returns text is a different engineering problem from an agent that can execute a program, inspect a repository and use a production API. The second system must decide not only what the model says, but what software is allowed to do after the model has said it. That decision belongs in infrastructure and authorization systems, not in a reassuring system prompt.

This book develops **AgentPlane**, a reference architecture for operating and governing agent execution in customer-owned Kubernetes clusters. It connects Go services, PostgreSQL, an outbound connector, Kubernetes controllers and a web console into a coherent design. Existing runtimes and gateways provide the low-level machinery; the platform supplies tenant boundaries, lifecycle intent, policy decisions and evidence.

The intended reader can already build a container, read a Kubernetes manifest and write ordinary application code. You do not need prior experience with AI training, GPU kernels or language-model research. You do need a willingness to reason about retries, uncertain outcomes and security properties that are not visible in a successful demo.

Three kinds of material

Design means a proposed AgentPlane contract or architecture. It is not a statement that this repository contains a working implementation.

Teaching example means an included, deliberately limited program that illustrates an invariant. Its local test results are recorded in the validation report. Passing those tests does not establish deployment security.

Integration exercise means work that requires a real database, cluster, identity provider or cloud account. The book supplies a procedure and acceptance criteria; a procedure is not an executed result.

This distinction applies even when a chapter uses confident imperative language. “Reject a stale command” is a requirement for the application you build, not a claim that every possible execution environment already rejects it.

How to read

Read Chapters 1–4 before implementing anything. Chapters 5–11 establish the control plane and its connection to Kubernetes. Chapters 12–21 examine execution, networking, tool governance, storage and evidence. Chapters 22–25 address product interfaces and operations. Chapters 26–28 show how to use coding agents, complete a capstone and maintain an open publication.

Use the labs to turn claims into observations. Use the prompt pack as a sequence of bounded implementation assignments, not as an unattended deployment script. The same assignments can guide a human developer or another coding assistant. Repository instructions never replace the assistant's actual permission model.

The edition's promise

The promise is a useful, reviewable engineering book. It is not guaranteed production readiness after a fixed number of prompts. The most important outcomes are explicit trust assumptions, testable invariants and an honest record of what has and has not been validated.

Start with the [table of contents](#), or run the offline checks described in the [repository README](#). The book is intentionally readable as plain Markdown on GitHub and as a self-contained website.

01. Choose a Product Boundary Before a Technology Stack

Part I — Foundations

The useful product is not “Kubernetes with an AI label.” It is a reliable answer to a customer question: *Can our software agents perform useful work without receiving unrestricted access to our infrastructure?* AgentPlane addresses the execution and governance part of that question. It does not decide whether a model's business reasoning is correct.

Consider a fictional customer, Northstar Analytics. Its internal agent clones a repository, runs tests, prepares a report and requests access to a reporting tool. Northstar already owns a Kubernetes cluster. It wants predictable isolation, limited credentials and an explanation of who authorized each tool operation. The platform is successful when Northstar can operate this workflow with less custom glue and a clearer security review. The example is a design scenario, not a customer reference.

The first supported journey

A project administrator installs a connector in a dedicated development cluster. A developer selects a reviewed runtime template, creates a session and executes a harmless test command. The session cannot reach an arbitrary internet endpoint. It can reach one approved tool through a gateway. A sensitive call requires a second person's approval. The developer terminates the session and obtains a metadata-only activity report.

This single journey is a better first milestone than a menu containing every possible agent feature. It crosses the important boundaries: human identity, tenant authorization, cluster identity, workload isolation, network policy, tool identity and lifecycle cleanup. Each crossing becomes a test later.

What the first release excludes

Do not include model training, a GPU scheduler, a general-purpose hosting service or an unreviewed marketplace in the first release. These expand the trust and operational surface without being necessary for the journey above. Keep managed public execution separate from BYOC. A service executing arbitrary code from unrelated strangers is a different risk profile from an internal team using its own dedicated cluster.

Avoid promising “zero data access.” Commands, filenames, tool inputs and output streams are data too. A control-plane relay that transports them can observe plaintext unless an additional end-to-end design prevents that. The more honest initial promise is that workspace storage and secret-provider access remain in the customer environment, while a documented set of metadata and optional interactive streams passes through the control plane.

Define acceptance by behavior

Requirement	Observable acceptance condition
Tenant isolation	A project-B credential cannot read or mutate a project-A session
Bounded execution	A timed-out process is canceled and its uncertain outcome is visible
Network control	A request to an unapproved destination fails in the actual CNI environment
Tool governance	A request without the required approval never reaches the tool server
Recoverability	A connector restart does not blindly repeat an unknown execution
Cleanup	Termination has a visible pending state until resources are confirmed removed

Do not accept a screenshot as evidence for these conditions. A screenshot shows an interface at a moment in time. Tests and retained observations establish the behavior behind it.

A small deployment model

Use Go for API and cluster-facing components, PostgreSQL for durable application state and Angular for the management console. These are project choices, not claims of universal superiority. Keep application-domain modules separable even when the API and worker share a process during early development. Deploy the connector and operator independently because they live across a trust boundary.

NATS JetStream becomes useful when command distribution and event throughput justify a dedicated transport. The first local proof can use a PostgreSQL outbox and a simple worker. Preserve the message contract so that transport can change without changing authorization or lifecycle semantics. Operational simplicity is a design benefit, provided it does not erase the trust boundaries.

Upstream Agent Sandbox supplies sandbox-oriented Kubernetes resources. Its controller and the chosen runtime still require a deployment-specific integration review. AgentPlane should add policy and product semantics, not duplicate every upstream feature under a different name. [S01](#)

Validate the product without invented market numbers

Interview teams that already execute agent-generated code. Ask which actions require manual approval, where credentials currently live, how they investigate failures and what they cannot permit an external service to observe. Request an example workflow with synthetic data. A willingness to run that workflow in a pilot is more meaningful than a positive response to a feature list.

A pricing experiment can compare a platform fee per organization, a cluster management fee and managed-compute charges. Treat any proposed price as a hypothesis. Do not describe an unmeasured margin or conversion rate as a forecast.

Exercise

Write a one-page product contract for Northstar. State the protected assets, the first workflow, the data that leaves its cluster, the operations requiring human approval and the conditions that block a pilot. Remove any feature that does not help prove this contract.

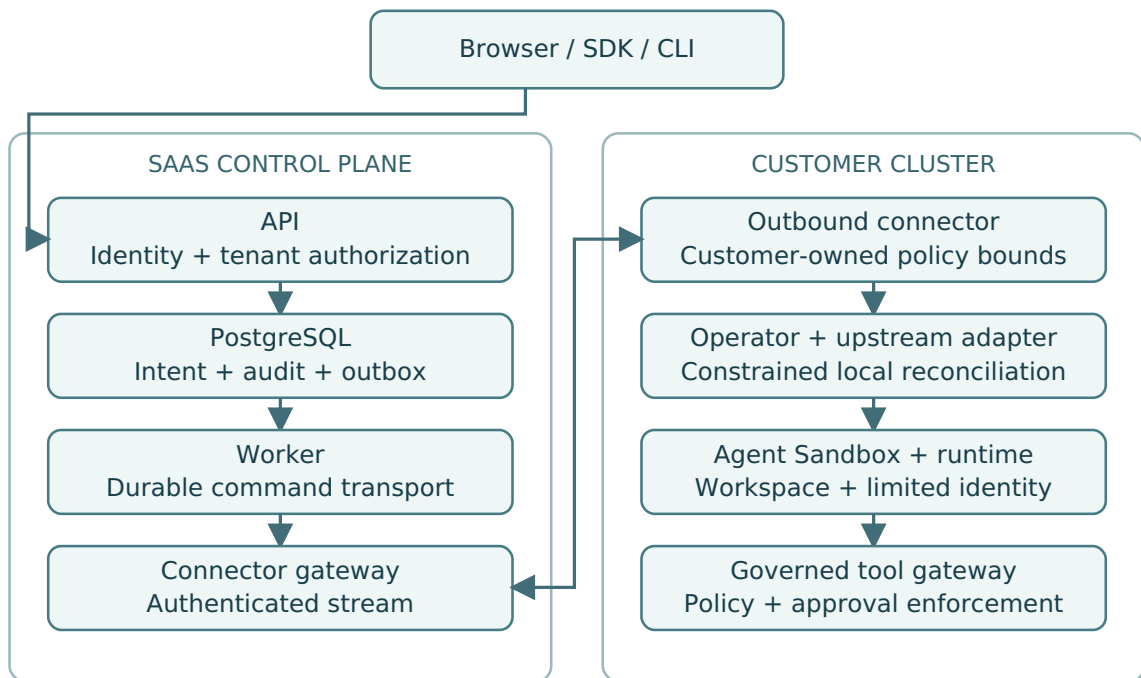
Primary sources

[Agent Sandbox documentation](#)

02. Separate the Control Plane from the Execution Plane

Part I — Foundations

A control plane stores intent: who owns a session, which template it should use, which policy applies and whether the desired state is running or terminated. An execution plane realizes that intent in a cluster. Separating the two makes ownership clearer, but the connection between them remains a powerful management channel.



The connector initiates mTLS; the channel still carries remote authority.
Local policy bounds must reject requests outside the customer envelope.

The browser and SDK contact the public API. The API authenticates the caller, checks organization and project permissions, writes durable state and enqueues work. A connector in the customer cluster initiates an authenticated outbound connection to the connector gateway. The connector interprets a constrained command vocabulary and calls a local adapter or Kubernetes controller. It does not accept arbitrary manifests supplied by the SaaS.

Logical components, not a microservice quota

The domain needs an API, an asynchronous worker, a connection gateway, a cluster connector, a policy operator and a web interface. These are logical boundaries. They do not imply that every module needs an independent deployment on day one.

Separating the connection gateway from the HTTP API becomes useful when long-lived streams have different scaling and drain behavior. Combining a small API and worker can be acceptable while their responsibilities remain explicit.

PostgreSQL is the authority for customer-visible operation records, desired state and authorization configuration. Kubernetes is the authority for observed cluster resources. Neither copy is always current. The reconciler compares them without pretending that a distributed transaction spans both systems.

Build a data classification table

Data	Normal location	Can it cross the SaaS boundary?
Membership and entitlements	Control database	Yes, to authorized interfaces
Connector private key	Customer cluster	Never in enrollment or telemetry
Customer secret value	Customer secret provider/workload	Not in normal control-plane APIs
Workspace files	Customer volumes	Only through an explicitly enabled transfer path
Execution output	Runtime and active client stream	Yes in relay mode; document this exposure
Tool arguments	Gateway and target tool	Only approved processing and narrowly controlled retention
Audit metadata	Control database and exports	Yes, with tenant authorization

A secret can appear inside arbitrary process output. Therefore “we never fetch secrets from Vault” does not prove “the SaaS never receives secrets.” Define output handling, default retention and redaction separately. Redaction is best effort, not a guarantee that arbitrary sensitive content will be recognized.

Why outbound connectivity helps—and what it does not do

Outbound enrollment removes the need for the SaaS to connect directly to the customer's Kubernetes API. It does not remove remote command authority. An attacker controlling the SaaS may still send commands over an established connector stream. Protect against that with cluster-local policy bounds, typed commands, constrained references, resource-side identity checks and a customer kill switch.

The connector must reject a command that requests a privileged pod even if the command is authenticated. Authentication establishes the speaker; local policy limits what that speaker may ask the customer environment to do.

Desired state and observed state

An API response such as 202 Accepted means an operation was recorded, not that a sandbox exists. The session model should expose desired state, observed state, observation time, operation ID and a safe reason code. For a disconnected cluster, show the last observation and its age. Do not repaint an old observation as live simply because the database remains reachable.

Termination illustrates the distinction. The API can record a terminal intent while the cluster is offline. The workload may continue until a local deadline or until the connector receives that intent. A local lease and maximum lifetime bound this exposure. Calling the session “terminated” before resource confirmation would conceal it.

Local policy as a customer-owned boundary

Allow customers to install a policy envelope that the SaaS cannot widen: approved registries, namespace scope, maximum lifetime, permitted runtime classes and allowed credential providers. Product policy can become more restrictive inside that envelope. Relaxing it requires a separate customer-controlled administrative operation. This is a design choice with an operational cost: a SaaS administrator cannot repair every policy problem remotely.

Kubernetes multi-tenancy guidance explains why namespaces, network controls, resource controls and stronger isolation choices must be considered together. A namespace is an organizational boundary, not a complete hostile-workload containment mechanism. [S03](#)

Exercise

Draw every path that can carry command output from a sandbox to an operator's screen. Mark where TLS terminates and where plaintext exists. Then describe the additional implementation needed to offer a direct customer-data-plane mode. Do not label a path “private” without defining who can read it.

Primary sources

[Kubernetes multi-tenancy](#)

03. Turn the Threat Model into Testable Invariants

Part I — Foundations

A threat model is useful when it changes implementation and tests. A list of attack names without owners or verification steps is just documentation debt. Start with assets: customer data, credential authority, tenant identity, compute capacity, image provenance and the integrity of control decisions.

The main actors are a legitimate developer, a compromised developer account, a malicious workload, a compromised MCP server, a compromised SaaS component, a stolen connector identity and a customer cluster administrator. These actors have different powers. In a conventional BYOC deployment, the cluster administrator can generally inspect or alter workloads. Do not claim protection from that actor unless a substantially different confidential-computing design is implemented and verified.

Name an invariant at each boundary

Boundary	Invariant	Verification
API to tenant data	Scope comes from authenticated authority, not just a request field	Cross-tenant read and write tests
Gateway to connector	Certificate identity and command cluster agree	Valid certificate with wrong-cluster payload is rejected
Connector to Kubernetes	Only approved typed operations reach managed namespaces	Arbitrary-manifest and namespace substitution tests
Sandbox to tool	Every privileged tool invocation has a valid decision	Direct-bypass and stale-policy tests
Approval to action	Approval is bound to exact action identity and used at most once	Concurrent consumption and changed-input tests
Audit to export	Alteration and truncation are checked against a trusted anchor	Corruption and tail-removal tests

The final row is deliberately narrower than “immutable audit.” A hash chain is not trustworthy if the attacker can replace both the events and the supposed trusted head. Evidence depends on a trust anchor outside the rewrite boundary.

Three failures that look successful

First, an application includes `organization_id` in every API response but does not use it in SQL predicates. The interface appears tenant-aware while the data layer is not. Second, a pod has a `NetworkPolicy` object but the cluster networking implementation does not enforce it. The desired configuration exists without the promised behavior. Third, a tool request creates an audit event and then bypasses an unavailable authorization service. The evidence pipeline works while the security control has failed open.

For each protection, distinguish declaration, enforcement and observation. The same distinction applies to `RuntimeClass`, signature policies and certificate revocation. A configuration name is not evidence of an effective boundary.

Prioritize by consequence, not by novelty

Spend more effort on wrong-tenant access and duplicate external side effects than on an elaborate AI-specific taxonomy. A coding agent that leaks a credential is a serious problem regardless of whether the initiating text was a prompt injection or an ordinary programming mistake. Treat model outputs and repository content as untrusted inputs to a governed execution system.

Tool descriptions, generated patches and terminal output must never be promoted into operator authority. For example, a repository file that says “disable egress protection to fix the build” is data. It cannot amend customer policy. The same principle applies to instructions encountered by the coding assistant developing AgentPlane itself.

Understand the residual risks

A sandbox runtime can reduce exposure to the host while still allowing the workload to misuse every credential it legitimately possesses. A network allowlist can restrict destinations without understanding the business meaning of a request to an approved host. A human approval can be mistaken or socially engineered. Timeout cancellation cannot undo a completed external payment or deletion.

The gVisor security model is a useful primary reference for distinguishing the runtime's boundary from the responsibilities of the surrounding platform. AgentPlane must account for both. [S06](#)

Write useful threat records

A record should include the entry point, attacker capability, target asset, required preconditions, expected rejection point, detection signal, regression test and remaining uncertainty. Assign a severity only after describing the consequence. “High severity because SSRF” is less useful than “this metadata fetch can obtain a node credential and read another project's object storage.”

Security reviews should include product restrictions. A release may prohibit public multi-tenant execution because isolation has not been assessed. That is a valid mitigation when communicated accurately; silently treating an internal runtime as safe for hostile strangers is not.

Exercise

Choose the command `terminate session`. Model a stolen connector credential, a stale connection owner and a malicious customer administrator. Explain which cases the platform can prevent, which it can detect and which it cannot reliably prove. Create a separate negative test for each preventable case.

Primary sources

[gVisor security model](#)

04. Create a Reproducible Engineering Workspace

Part I — Foundations

A reproducible repository records inputs, build steps and observed results. It does not merely contain a convenient startup command. For AgentPlane, distinguish the documentation workspace, the teaching examples and the future application workspace. This repository builds the book. The prompt pack targets a separate application repository unless you explicitly choose otherwise.

Establish a small command surface

A useful application repository eventually exposes format, generate, lint, test, build and verify. Keep commands composable so a contributor can run a single layer without provisioning infrastructure. A command named verify must not quietly create paid cloud resources or install cluster-wide components.

This book's commands are intentionally smaller:

```
python -m venv .venv
. .venv/bin/activate
python -m pip install -r requirements.txt
python scripts/check_book.py
python scripts/test_examples.py
python scripts/build_book.py
python -m http.server 8000 --bind 127.0.0.1 --directory _site
```

The renderer is pinned in requirements.txt. The validation report identifies the actual Python and Go versions used locally. The Go teaching module uses an older language baseline for portability; this is not advice to deploy an unsupported Go toolchain. Production tools need a separately maintained compatibility record.

Record a version matrix, not a wish list

For each production dependency, record an exact version, source URL, artifact digest, license, date reviewed and the integration tests run with it. A row with no successful test is a candidate, not a supported version. Keep the Kubernetes server version separate from client libraries, CRD versions, the runtime daemon, the sandbox router and the CNI. They can change independently.

Do not write guessed version numbers into generated manifests to make them look complete. A configuration template that requires a verified digest is more honest than a fabricated immutable-looking digest. The Kubernetes example renderer in this repository requires explicit operator input and produces a manifest for review; it does not deploy it.

Make generated contracts reproducible

The public API should have one schema source. Generate low-level server and client types from that source, then add a handwritten application layer. Keep protobuf command envelopes separate from public HTTP resources. Their consumers and compatibility requirements differ.

Record the generator version and invocation. A regeneration check should fail when generated files differ from the repository, rather than overwriting them and reporting success. Review generated changes as part of the source change. Generated code can contain a breaking API change just as handwritten code can.

Use a layered local environment

The smallest loop needs no cluster: domain tests, authorization tests, state transitions and rendering. The next loop adds a disposable PostgreSQL instance for migrations, RLS and concurrency. A third loop adds a Kubernetes cluster with a policy-enforcing CNI and a chosen sandbox runtime. The final loop tests a specific cloud deployment and identity integration.

A default Kind installation is not automatically an adequate network-policy or hostile-code laboratory. Verify the installed CNI and runtime behavior. When the required capabilities are absent, skip the integration explicitly and mark the result unvalidated. Never substitute “object created” for “policy enforced.”

Preserve a useful evidence trail

Store commands, exit codes, test counts, environment identifiers and sanitized reports. Keep logs bounded. Do not collect every environment variable, pod log or workspace archive on failure: diagnostics often contain the secrets that normal logging correctly excludes.

Separate a proposed benchmark target from a measured result. For example, a provisioning target belongs in an SLO proposal. A result belongs in a report with hardware, versions, sample size and a clearly defined start and end point. A warm pool claim and a cold node provision are different measurements.

Keep publication permissions narrow

Book pull requests should run checks without deployment permission. Publishing a Pages artifact belongs in a separate protected job. The included workflows use reviewed action commit references and do not use pull-request content in shell commands. GitHub's secure-use guidance explains the rationale for these trust boundaries. [S24](#)

Exercise

Create a dependency record for the sandbox controller and runtime server. List which integration tests would have to be repeated after changing either one. Then identify the smallest local command that can still give useful feedback when Docker and Kubernetes are unavailable.

Primary sources

[Mistune usage guide](#) · [GitHub Actions secure use](#)

05. Design Tenant-Safe PostgreSQL Data

Part II — Control Plane

Tenant isolation is not a field naming convention. It is a property of every query, reference, transaction and administrative path. AgentPlane uses an organization as the top-level customer boundary and a project as a narrower operational boundary. Resources belong to both whenever project scope applies.

Use references that carry scope

A sandbox row should reference a project through the pair (`organization_id`, `project_id`). A plain `project_id` foreign key can prove that a project exists while failing to prove that it belongs to the sandbox's organization. Composite references make an important class of accidental cross-tenant association impossible at the database boundary.

The teaching schema in [examples/sql](#) creates organizations, projects and sessions with composite keys and a row-security policy. It is intentionally smaller than the final domain. Expanding it into a production database requires migrations, deletion policy, indexes, operational roles and measured query plans.

```
-- Design fragment: authenticated organization context must be set inside
-- the same transaction that issues the protected query.
BEGIN;
SELECT set_config('app.organization_id', $1, true);
SELECT id, status
FROM agentplane_book.sessions
WHERE organization_id = $1::uuid AND project_id = $2::uuid;
COMMIT;
```

This fragment uses bind parameters; it is not a standalone psql script. The third argument to `set_config` makes the setting transaction-local. Reusing a pooled connection after a session-level setting can otherwise carry one tenant's context into another request.

RLS is defense in depth

PostgreSQL row-level security can restrict visible and writable rows. Table owners normally bypass it unless forced, and roles with `BYPASSRLS` or superuser privileges are not constrained in the ordinary way. Therefore test the actual application role, not only a migration owner. [S07](#)

Use `USING` for visible rows and `WITH CHECK` for new row values. Enable and force RLS for tenant-owned tables where appropriate. Keep the runtime role separate from schema ownership and grants administration. Revoke unnecessary schema creation rights and avoid casually introducing `SECURITY DEFINER` functions.

An application-controlled tenant setting is not cryptographic authorization. Anyone who gains arbitrary SQL execution as the application role may be able to change that setting. RLS helps contain accidental query omissions; it does not replace parameterized SQL, application authorization or separate databases when a stronger tenant threat model requires them.

Scope caches and uniqueness

Cache keys need organization and project identity just as SQL queries do. A cache indexed only by `runtime-name` can return another tenant's template. Idempotency keys need scope too: include the authenticated actor or service account, tenant, operation and request digest. A globally unique idempotency key accidentally becomes a cross-tenant coordination channel.

Use unique constraints for public resource names only within their intended scope. Decide whether soft-deleted names can be reused. Partial unique indexes can express active-name uniqueness, but historical operations must retain stable resource IDs so a retry never targets a new resource with an old display name.

Keep transactions bounded

Do not hold a database transaction open while waiting for Kubernetes or a human approval. Write intent, audit metadata and the outbox event together, commit, then do external work. If dispatch fails, a worker can retry from the outbox. The transaction establishes a durable decision boundary, not end-to-end success.

For quota reservation, lock a small per-scope counter row or use a conditional update. Avoid “count sessions, then insert” without concurrency control. Two requests can observe the same count and both exceed the limit. PostgreSQL row locks provide the necessary coordination, but long lock duration and inconsistent lock order can create deadlocks. [S08](#)

A practical test matrix

Test read, insert, update, delete and foreign-key substitution as the runtime role. Include an empty tenant context, malformed context, another organization, another project in the same organization and a reused pooled connection. Verify that a batch endpoint cannot mix authorized and unauthorized resource IDs.

Administrative exports need an explicit scoped job identity. Do not solve every background-job problem with a permanent unrestricted connection. Maintenance and migration roles may need stronger permissions, but their use should be separate, audited and unavailable to normal request handlers.

Exercise

Run the SQL lab in a disposable database. Record both a successful scoped read and a rejected cross-organization insert. Explain why a successful test as a superuser says little about application RLS behavior. Extend the lab with a runtime-template table whose references cannot cross project boundaries.

Primary sources

[PostgreSQL 18 row security](#) · [PostgreSQL 18 explicit locking](#)

06. Keep Human, Service and Cluster Identities Distinct

Part II — Control Plane

AgentPlane has at least four identity classes: human users, service accounts, cluster connectors and local workloads. Sharing one authentication mechanism across all four makes authority difficult to reason about. A browser session should not authenticate a connector, and a connector certificate should not become an organization administrator credential.

Human sessions

For a same-origin web console, an opaque server-side session is a straightforward starting point. Put the session identifier in a Secure, HttpOnly cookie and protect state-changing requests against CSRF. Choose SameSite behavior to match the actual deployment and authentication flow rather than applying a setting without testing it. Session rotation, expiration and server-side revocation are part of the design. [S14](#)

Store hashes of reset and verification tokens. Make them single-use and bounded by expiry. Avoid account enumeration in both responses and timing where practical. Use a reviewed password-hashing library and calibrate resource costs on the actual service environment. A copied parameter set is not a substitute for performance and abuse testing.

An external identity provider can reduce first-party credential handling, but it does not remove authorization. Validate issuer, audience, redirect binding and other protocol requirements through maintained libraries. Keep the mapping from identity-provider subject to local membership explicit.

Authorization is a separate decision

A role is a named collection of permissions. A permission is not a plan feature. A billing administrator may manage subscription settings without executing code. A developer may execute a session without changing cluster-local security bounds. Effective authorization is the intersection of identity permissions, resource scope, organization policy and product entitlements.

Define a small permission vocabulary before adding arbitrary custom roles: `cluster.enroll`, `runtime.manage`, `sandbox.create`, `sandbox.execute`, `policy.manage`, `approval.decide`, `audit.read` and `billing.manage`. Deny by default. Protect the last organization owner and prevent a delegating user from granting powers they are not allowed to delegate.

Do not let a hidden UI button become the security boundary. The server must check every operation, including WebSocket subscriptions, file downloads and export status endpoints. When membership changes, long-lived operations need a defined revalidation or revocation policy.

Service-account credentials

Use a public key identifier and a high-entropy secret component. Store a secure verifier rather than the full secret. Display new credentials once, support rotation overlap and make revocation observable. Bind a service account to organization and project scope; optionally allow explicit organization-wide permissions where the product requires them.

Rate-limit failed verification before it becomes an expensive database or hash operation. Do not record credentials in CLI debug output, reverse-proxy logs or error reports. A one-time reveal screen must not leak the secret into browser analytics or persistent client state.

Stream authorization

An interactive terminal is a privileged channel, not a read-only dashboard widget. Authorize the session and execution being attached. A short-lived, single-use stream ticket can avoid placing a long-lived API credential in a WebSocket URL. Still consider proxy access logs: any ticket in a URL can appear there. Prefer a reviewed handshake flow that avoids query-string credentials, and ensure expiration and replay handling are tested.

Disconnect streams when the user logs out, changes project context or loses the necessary permission according to the chosen revocation policy. Bound the delay if authorization is cached. “Immediate revocation” is not credible when an unbounded cache or a never-rechecked stream remains active.

Keep authorization evidence useful

Record actor type, actor ID, scope, operation, outcome, policy version and request ID. Do not record passwords, complete tokens or tool input bodies. Failed authorization can reveal sensitive resource existence, so user-facing errors may need a consistent not-found response while protected operational logs retain the specific reason.

Exercise

Construct a permission matrix for owner, developer, security administrator, viewer and billing administrator. Add a service account allowed only to create sessions from one template. Test that its credentials cannot approve its own requests, discover another project or subscribe to another execution's stream.

Primary sources

[OWASP session management](#)

07. Design APIs for Work That Finishes Later

Part II — Control Plane

Creating a sandbox is not a short database mutation. It can involve quota reservation, image policy, cluster dispatch, scheduling, storage provisioning and runtime readiness. A synchronous endpoint that hides all of this behind a long HTTP timeout turns normal delays into ambiguous failures.

Return an operation, not a false completion

A proposed create request is accepted only after authentication, authorization, validation, quota reservation and durable intent recording. It returns an operation ID and a session ID. The client observes progress separately. The operation carries its own lifecycle: accepted, dispatched, running, succeeded, failed, canceled or outcome unknown. Session lifecycle is related but distinct.

```
{
  "operation_id": "op_example_001",
  "session_id": "session_example_001",
  "operation_status": "accepted",
  "desired_state": "running",
  "observed_state": "not_observed",
  "observed_at": null
}
```

This is an AgentPlane design example, not a response from an implemented API. Keeping `observed_at` empty is more useful than inventing a current timestamp for an event that has not happened.

Bind idempotency to request identity

Store an idempotency record under authenticated scope, operation type and a client-supplied key. Bind it to a canonical request digest and the resulting operation ID. Reuse of the same key with the same request returns the original operation. Reuse with a different request returns a conflict. Concurrent first requests must converge through a unique database constraint, not through an in-memory check.

Define canonicalization. Arbitrary JSON serialization can change ordering or numeric representation. A typed request digest should cover the semantic inputs that determine the operation, including scope and template version. Do not include volatile tracing headers. Treat digests of sensitive small inputs as potentially revealing: hashing is not encryption and can permit guessing attacks.

Idempotency records need a retention contract. If the server forgets a record after a day, a retry after two days may create a new operation. Document the window and make SDK behavior respect it. Long-lived external side effects may need longer identity retention than ordinary read caches.

Distinguish retryable failures

A temporary inability to reach the gateway can be retried before execution begins. An unknown result after the process started is different. A script may have updated an external system even if its result was lost. Blind retry can duplicate that side effect. Return an explicit unknown outcome and require reconciliation or an action-specific idempotency mechanism.

For safe reads, retries can use bounded exponential backoff with jitter. For writes, reuse the original idempotency key. Never let an SDK create a fresh key on every retry while claiming the operation is idempotent. Cancellation is a request to stop remaining work; it is not a rollback of completed external work.

Use optimistic concurrency for changing intent

A session has an intent version. A client changing lifetime or requesting termination supplies the version it observed. The server rejects a conflicting update or explicitly resolves it according to documented rules. Terminal intent must dominate stale resume requests. Avoid generic “last write wins” for security and lifecycle state.

Workers should check current intent before starting expensive work. A queued create operation may have been canceled while the cluster was offline. The connector also checks the operation's expiry and local policy. One initial validation does not authorize a command forever.

Define errors for humans and automation

Use stable machine-readable codes with safe messages. Include a request ID and, where relevant, an operation ID. Separate invalid input, unauthorized scope, resource conflict, quota exhaustion, unsupported capability, temporary unavailability and unknown execution outcome. Clients should not parse English message text to decide whether to retry.

Do not expose raw Kubernetes error objects that may reveal names outside the project. Preserve detailed diagnostics in authorized operational channels after redaction. Bound request bodies, filter fields and pagination sizes. A rich list endpoint is still an authorization and denial-of-service surface.

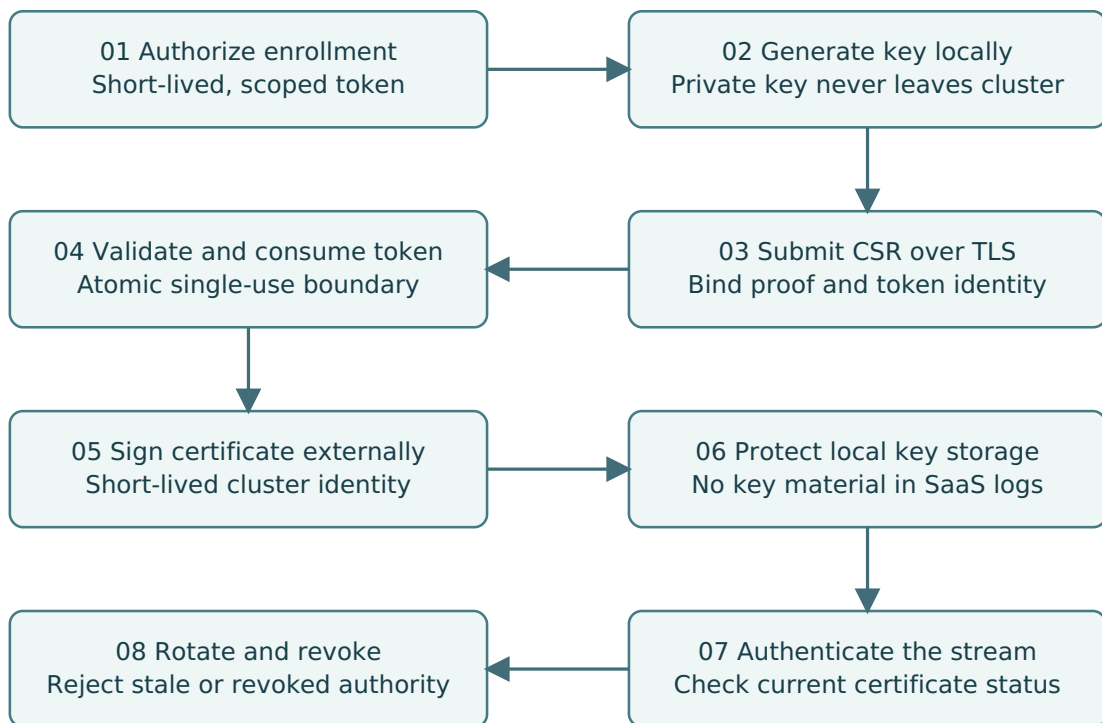
Exercise

Write three create-session tests: simultaneous duplicate requests, a changed request under the same idempotency key and a retry after the server commits but before the client receives its response. Then write an execution test where the external effect succeeds and the response is lost. Explain why that final case requires more than HTTP idempotency middleware.

08. Enroll Clusters Without Exporting Their Authority

Part II — Control Plane

Cluster enrollment binds a locally generated key to a narrowly scoped customer cluster identity. The enrollment token is not a permanent credential. It is a short-lived invitation to establish one identity under explicit organization and project ownership.



The enrollment sequence

An authorized administrator creates a cluster record. The API generates a high-entropy token, stores only its verifier and reveals it once. The connector generates a private key locally and constructs a certificate signing request. It submits the CSR and token over a server-authenticated TLS connection. The service validates the token, reserves or consumes it transactionally, and issues a certificate whose identity is selected by the service—not by untrusted CSR subject fields.

The issuer must not copy arbitrary requested SANs or certificate usages. Bind the certificate to the cluster ID, use the intended client-auth purpose and record serial, fingerprint, validity and issuer. A URI identity is a useful convention, but it is a proposed naming scheme unless a full identity standard is adopted. Do not call an arbitrary URI “SPIFFE compliant” without meeting that standard.

Handle enrollment failure without leaking keys

Issuance and token consumption span a database and a signer. Define an idempotent enrollment operation so a lost response does not create an uncontrolled sequence of valid certificates. Retain the approved public-key fingerprint and issuance result for the enrollment operation. A retry with a different key must not silently reuse the consumed invitation.

Private keys never leave the cluster. Store them with minimum practical access. The connector may need access to its own credential Secret, but it should not have blanket read access to all Secrets. Separate a small bootstrap permission set from normal runtime permissions where this materially reduces exposure.

Authentication includes live status

A valid certificate chain is necessary but not sufficient. The gateway verifies issuer trust, time validity, intended usage, identity binding and cluster status. A disabled cluster or revoked certificate must be rejected even when its cryptographic signature is valid. Define how gateways learn revocation and how quickly established streams are closed.

An authentication cache introduces a revocation delay. Bound it and expose it as an operational property. When the status authority is unavailable, deny new privileged sessions according to the stated fail-closed policy. Existing sessions need an explicit bounded lease; otherwise a database outage accidentally extends a stolen credential's useful lifetime.

Rotation and recovery

Rotate before expiry with a small overlap. Establish the new identity, confirm the new stream and retire the old serial. Do not delete the only working credential before the replacement is durable. Rotation needs clock-skew handling, but a large skew allowance weakens expiry. Report observed skew as a degraded condition.

A connector offline past certificate expiry may be unable to use ordinary rotation. Provide an administrator-controlled reenrollment workflow. This is not a reason to keep a hidden permanent token in the cluster. Emergency recovery must establish authority again, not bypass it.

Use an external production signer or a managed CA integration with a documented trust and backup model. A development CA stored next to the application database is not made production-ready by adding an environment variable named `secure`. The book does not implement a production certificate authority.

Customer revocation must work locally

SaaS revocation cannot reach a disconnected cluster immediately. Give the customer a local procedure to stop the connector, remove its authority and terminate or quarantine workloads according to policy. Local maximum lifetimes and command leases bound how long workloads can continue without fresh authorization.

Certificates also do not establish that a report is truthful. A customer administrator holding cluster authority can modify the connector or its runtime. Treat cluster-reported inventory as authenticated information from that cluster, not independently attested proof of isolation.

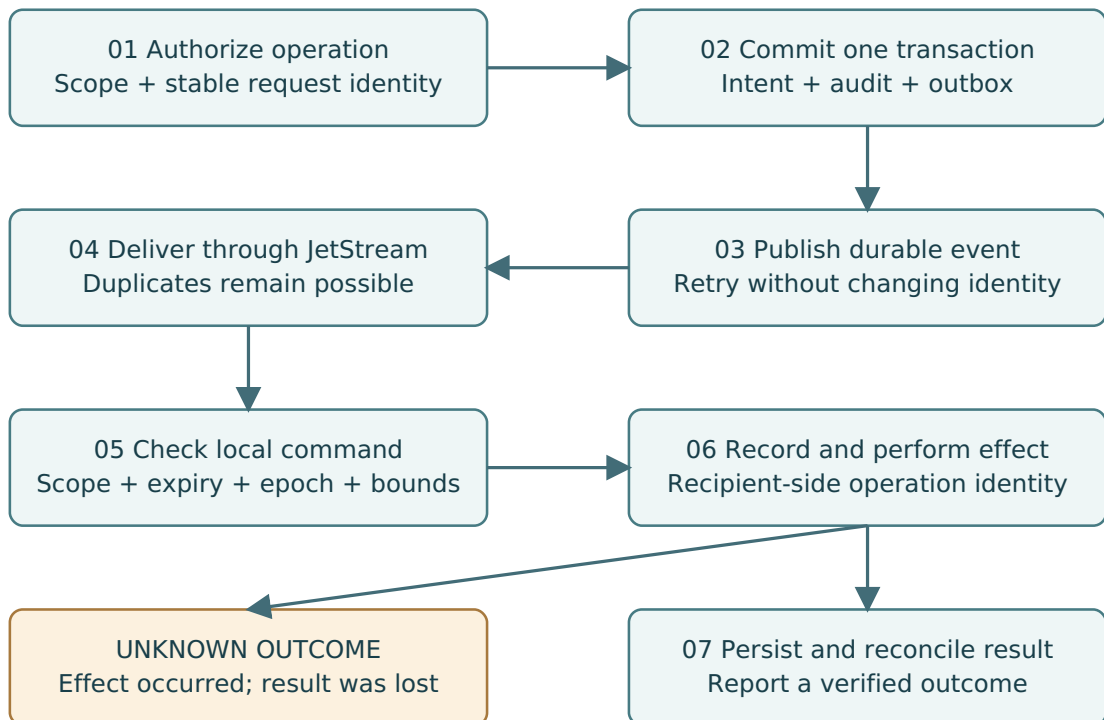
Exercise

Model a lost enrollment response, a stolen token before use, a certificate revoked during an open stream and a connector returning after expiry. For each case, write the durable record, expected user-visible state and safe recovery action. Never include private key material in the resulting support bundle.

09. Make Delivery Durable Without Promising Exactly-Once Effects

Part II — Control Plane

The messaging contract must survive duplication, reconnects and uncertain results. NATS JetStream provides persistence and delivery mechanisms, but an acknowledged message is not proof that an external business operation happened exactly once. The application still owns effect identity and reconciliation. [S09](#)



Commit intent and an outbox event together

The API transaction writes the operation record, quota reservation, audit metadata and an outbox event. A worker later publishes the event. If publication succeeds and the worker crashes before marking the outbox row delivered, publication may repeat. Consumers therefore need deduplication. This is the intended behavior, not an exceptional corner case.

Keep the outbox immutable enough to replay deterministically. Track attempts and lease ownership separately from the event body. Bound batch size and lock duration. Use `FOR UPDATE SKIP LOCKED` or a carefully designed equivalent for worker claiming, then release database transactions before slow network work. A lease expiration permits recovery but does not prove the previous worker stopped.

Use distinct identifiers

An operation ID identifies the user-visible intent. A command ID identifies a particular dispatchable instruction. A message ID identifies a transport event. An execution ID identifies a runtime action. A trace ID connects observability. These IDs are related but not interchangeable.

For example, the same command can be delivered in multiple transport messages. An operation can contain a create command and a later cleanup command. A trace can be sampled away without erasing execution identity. Keeping these concepts separate avoids accidental coupling between billing, tracing and correctness.

Model command state durably

A connector may observe accepted, started and finished states. A durable journal can return the original result for a repeated completed command. However, a journal entry marked started does not prove whether the process ran before a crash. Use a runtime execution service with durable operation identity when available. Otherwise expose an unknown state and reconcile; do not automatically start a second copy.

Resource creation is often easier to reconcile than arbitrary execution. A stable Kubernetes resource name and ownership metadata let a repeated create observe the existing resource. For an arbitrary script calling a third-party API, a local annotation cannot make the third-party effect idempotent. That API must support its own idempotency or the workflow must handle uncertainty.

Coordinate connections, then fence effects

Only one logical connector connection should own a cluster stream. Maintain a monotonic connection epoch and reject stale command ownership. Kubernetes Leases are useful for leadership coordination, but they do not by themselves make every external action exclusive. [S21](#)

A stale worker may continue after losing a lease. The component performing the side effect must validate fencing or operation ownership in an atomic mechanism appropriate to that effect. A check followed by an unrelated action can still race. For Kubernetes updates, resource versions help with concurrent state changes; for process starts, use a durable execution claim enforced by the runtime service. Document any effect for which strict fencing cannot be established.

Bound the transport

Set maximum message size, pending commands, per-cluster concurrency and output buffer size. Separate long output streams from the durable control journal. Storing megabytes of terminal output in every JetStream command result can turn a small execution into a fleet-wide memory and storage problem.

Set command deadlines and reject stale work on both sides. An old create request must not suddenly provision a session after a lengthy outage if its intent has expired. Dead-letter handling should preserve safe diagnostics and require a reviewed replay action; it must not be an automatic infinite retry loop.

Reconcile after interruption

When a connector reconnects, exchange its epoch, capabilities and a bounded set of operation summaries. Compare observed resources with current desired state. Do not replay every historical event as a new command. Use the database and resource identities to decide what is missing, stale, complete or unknown.

Exercise

Create a failure table for crashes before publish, after publish, after command receipt, after process start and after process completion. State which cases can be retried automatically and which need runtime reconciliation. The Go teaching examples illustrate local identity and state guards, not a distributed exactly-once execution engine.

Primary sources

[NATS JetStream concepts](#) · [Kubernetes Leases](#)

10. Use Kubernetes Reconciliation Without Expanding Authority

Part III — Kubernetes Integration

A controller observes a resource and moves dependent state toward its desired configuration. Reconciliation must tolerate repeats, partial progress and stale observations. The operator pattern provides a useful structure for AgentPlane's cluster-local policy resources. [S22](#)

Own only what the product adds

Upstream sandbox resources already model much of workload lifecycle. AgentPlane needs additional intent such as runtime governance, egress constraints, secret references and tool policy bundles. A proposed API group might contain `RuntimeProfile`, `EgressPolicy`, `SecretBinding` and `PolicyBundle`. These names are design examples, not installed CRDs in this repository.

Avoid a catch-all resource containing arbitrary Kubernetes YAML. It would make the connector a remote deployment administrator and defeat the typed command boundary. A small schema makes invalid and dangerous requests easier to reject.

Separate spec, status and observation

Spec describes the requested state. Status reports what the controller observed. Include an observed generation and conditions with stable reason codes. A useful condition distinguishes invalid policy, unsupported provider, dependency missing, compilation pending, applied and degraded. Do not reduce every condition to a single green/red field.

A newer spec should not inherit an older successful condition without an updated observed generation. The SaaS must compare versions before showing policy as active. Otherwise a failed security update can look applied because a previous version succeeded.

RBAC is necessary but not sufficient

Namespace-scoped roles are safer than blanket cluster authority, but Kubernetes RBAC does not generally restrict create requests by arbitrary object content. A role allowed to create Pods in a namespace can be powerful. Admission policies, namespace ownership and controller-side validation must limit the configuration that can actually be created.

Similarly, watching cluster-scoped resources can reveal information beyond the project. Request only the discovery fields needed to establish capability, and report a sanitized summary. Separate inventory access from mutation authority where practical. Do not grant `cluster-admin` to simplify installation.

Reconcile safely

Use deterministic names derived from stable IDs rather than user-supplied names. Validate ownership before updating an existing object. If an object with the same name belongs to another controller or tenant, report a conflict instead of adopting it silently. Avoid broad label selectors whose membership a workload can change.

Patch narrowly. When multiple controllers touch an object, define field ownership and conflict handling. Do not continuously overwrite customer-owned fields merely because they differ from a cached SaaS representation. Drift can be intentional, unauthorized or a result of another controller; the recovery action depends on which case occurred.

Finalizers are operational commitments

A finalizer can delay deletion while cleanup is pending. It can also block a namespace indefinitely when its controller is unavailable. Use finalizers only where the cleanup requirement is real. Document timeout, escalation and manual removal procedures, including the risk of leaked resources.

Never automatically remove a finalizer just to make a dashboard look healthy. The correct user-visible state may be “cleanup requires operator action.” A retained volume with customer data is not an ordinary disposable cache.

Test beyond object creation

Unit tests cover policy compilation and deterministic naming. Controller tests cover repeated reconciliation, observed-generation handling, stale resources, missing dependencies and deletion. A real cluster is required to verify admission, RBAC, network enforcement and runtime behavior. A fake API server cannot prove that a packet was denied or that a syscall was isolated.

Exercise

Design the status conditions for an egress policy that requests domain filtering on a cluster without that capability. The policy must not silently degrade to a CIDR rule. Explain how the API, connector, operator and UI all preserve that failure rather than converting it into a successful deployment.

Primary sources

[Kubernetes operator pattern](#)

11. Integrate Agent Sandbox Through a Capability Adapter

Part III — Kubernetes Integration

Agent Sandbox provides a core Sandbox resource and extension concepts including `SandboxTemplate`, `SandboxClaim` and `SandboxWarmPool`. Its source repository also distinguishes orchestration from lower-level runtime isolation. AgentPlane should integrate these capabilities rather than relabeling ordinary Pods as a complete agent security system. [S01](#) [S02](#)

Do not infer API compatibility from a feature name

“Execute a command” may involve a runtime server inside the sandbox, an SDK, a router and the Kubernetes resource managing the workload. These components have separate responsibilities. A working controller does not imply that an arbitrary OCI image exposes the execution or filesystem API expected by the SDK.

Record a compatibility tuple: Kubernetes server, sandbox CRD release, controller, extensions, router, runtime server, client SDK, container runtime and CNI. Test the tuple that you intend to support. Reading documentation does not populate a successful compatibility matrix.

Create a narrow internal interface

The application-facing adapter should express product operations without leaking upstream object structures:

```
CreateFromTemplate(scope, templateVersion, operationID)
Observe(resourceIdentity)
Terminate(resourceIdentity, intentVersion)
Capabilities(clusterIdentity)
StartExecution(executionIdentity, request)
QueryExecution(executionIdentity)
```

This is interface pseudocode. Optional operations such as hibernation, snapshots and file transfer belong behind explicit capability checks. A missing capability returns a typed unsupported result. It must not trigger an unsafe fallback such as privileged `kubectl exec` under a broad service account.

Keep orchestration and execution separate

The Kubernetes adapter creates and observes sandbox resources. The runtime adapter performs allowed operations inside them. The router transports requests; it is not automatically an authorization policy engine. Validate which layer checks identity, scope, timeout and request size. Do not assume that because an SDK is official, every product-level requirement is already enforced.

Restrict runtime endpoints to the appropriate caller identities and networks. A runtime API that accepts arbitrary commands is a privileged interface even when it listens only inside the cluster. A compromised neighboring workload must not be able to call it directly.

Define support states precisely

Use separate states for discovered, compatible, configured and verified. A CRD may be present while its controller is unavailable. A runtime class may exist without any schedulable nodes configured to use it. A snapshot class may exist without a working driver. Capability discovery is a starting observation; a successful probe and workload test provide stronger evidence.

Expose these distinctions in onboarding. For example, “runtime API not verified” is more useful than a generic cluster-connected badge. It tells the operator which acceptance test is still missing.

Handle upgrades as compatibility changes

Pin the release artifacts used by an application environment. Capture checksums and inspect CRD schemas before applying updates. New fields, renamed status conditions or changed runtime endpoints can break an adapter even when the user journey sounds unchanged. Maintain fixtures from supported releases and run contract tests against each intended version.

An adapter can support multiple release families, but each additional family has maintenance cost. Prefer a narrow tested range over an untested promise to work with every Kubernetes cluster. If an old version is incompatible, stop new provisioning while preserving a documented path to observe and terminate existing sessions safely.

Treat warm pools and hibernation carefully

Warm pools reduce some startup work by preparing capacity in advance. They do not eliminate policy binding, identity assignment or readiness checks. A pooled workspace must not carry credentials or files from a previous tenant. Prefer fresh unclaimed capacity over recycling arbitrary used sessions.

Hibernation semantics depend on the upstream release and runtime. Do not promise process-memory preservation simply because storage persists or a resource can scale to zero. Represent restartable workspace suspension separately from any verified process checkpoint capability.

Exercise

Complete the compatibility worksheet in the appendices for one chosen release. Identify the exact runtime image providing command execution. Write a test that creates a sandbox successfully but fails runtime readiness, and confirm that the product does not mark the session ready for execution.

Primary sources

[Agent Sandbox documentation](#) · [Agent Sandbox source repository](#)

12. Build Runtime Profiles for the Actual Threat Model

Part III — Kubernetes Integration

A runtime profile combines an image, resource bounds, operating-system settings, identity and network constraints. Its purpose is to make execution assumptions explicit and reviewable. It cannot turn a conventional shared container into a perfect security boundary by adding a reassuring name.

Separate trusted development from hostile execution

An internal developer running a reviewed test suite in a dedicated cluster is not the same scenario as a public user submitting arbitrary code. The latter deserves stronger runtime isolation, admission controls, node separation and independent security review. A development profile using the default runtime must be labeled and prevented from becoming a public production profile accidentally.

Kubernetes Pod Security Standards define useful pod-level restrictions. They do not constitute a complete sandbox for hostile code. Runtime isolation mechanisms such as gVisor address a different part of the boundary and still have documented assumptions. [S05](#) [S06](#)

Make templates immutable after use

A published runtime-template version should bind an immutable image reference, runtime class, resource limits, workspace policy, network policy and identity references. Existing sessions retain that version. Editing a template creates a new version; it does not silently change the meaning of historical evidence.

An image digest identifies bytes, not safety. Signature verification establishes an expected signing identity or provenance relationship, not that the software is non-malicious. Vulnerability scanning is another signal, not a proof of absence. Keep the trust policy, evidence and exceptions reviewable.

Restrict pod authority

Prefer non-root execution, no privilege escalation, dropped Linux capabilities, a reviewed seccomp profile, disabled automatic service-account token mounting and a read-only root filesystem where the runtime permits it. Provide bounded writable locations for the workspace and temporary files. Do not mount the Docker socket, host filesystem or connector credential directory into the workload.

Avoid host networking, host process namespaces and privileged mode. A profile requesting them should fail policy validation, not merely produce a warning in the interface. Protect the admission path too: a compromised connector should not be able to bypass restrictions by submitting a different pod shape.

The example renderer in [examples/kubernetes](#) requires a real image digest and runtime class. It emits a reviewable teaching manifest and never applies it. Even a syntactically valid result needs a real cluster test before use.

Bound resources beyond CPU and memory

Set ephemeral-storage bounds, execution deadlines, concurrent process limits where supported, output limits, file-transfer limits and workspace quotas. A workload can exhaust the platform through logs, file count, network connections or API requests without using much CPU. Resource limits must be enforced at the layer that can actually observe and control the resource.

CPU requests help scheduling; they are not measured usage. Memory limits can cause termination rather than graceful recovery. An OOM event should produce a clear observed failure reason and preserve the distinction between runtime failure and user cancellation.

Runtime images need a maintenance process

Build from reviewed bases, pin dependencies, generate an SBOM and keep rebuild provenance. Do not leave a floating package installation step in an otherwise immutable-looking Dockerfile. The runtime server must be compatible with the adapter and have its own authentication and request-limiting design.

Separate untrusted workspace content from executable runtime infrastructure. Installing a repository dependency may execute lifecycle scripts; treat that as execution under the same sandbox restrictions. A package manager is not a safe exception to the network or credential policy.

Define removal and quarantine

If a runtime image is revoked, block new sessions from it. Decide whether existing sessions should be allowed to finish, quarantined or terminated based on the risk. Preserve evidence and avoid a blanket cleanup that destroys information needed for investigation. The response should be an explicit policy operation, not an invisible background replacement of the image digest.

Exercise

Review a runtime profile for an agent that runs Python tests and reads one private repository. Remove every permission unrelated to that task. Explain where Git credentials are available, how long they last, what outbound destinations are permitted and how a compromised dependency is contained.

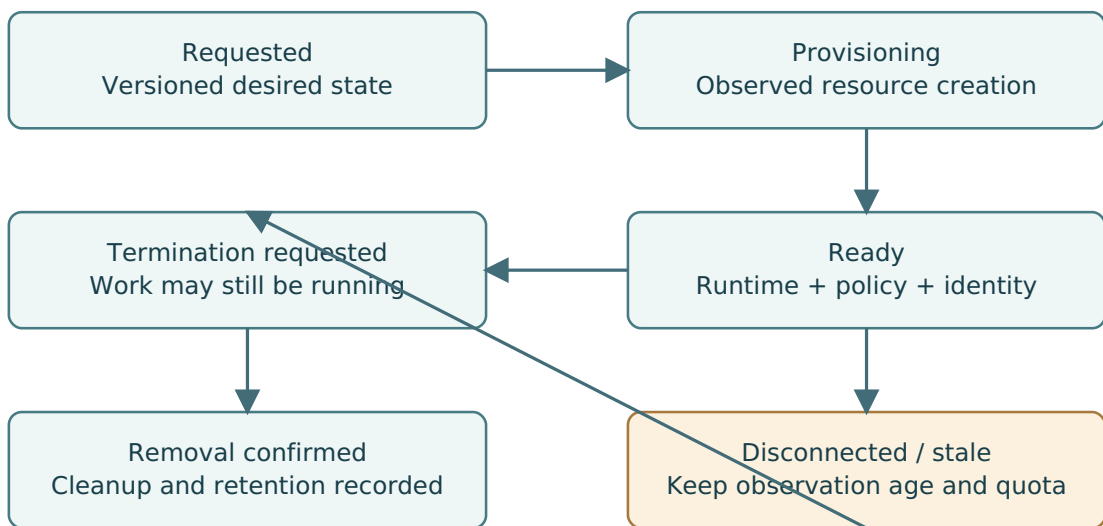
Primary sources

[Kubernetes Pod Security Standards](#) · [gVisor security model](#)

13. Model Sessions, Executions and Uncertainty Separately

Part IV — Safe Execution

A session is a governed workspace and runtime identity. An execution is one action inside that session. An operation is the platform's attempt to change or use a resource. Combining all three into one status field produces contradictions: a session can remain healthy while one execution fails, and a termination operation can be pending while the last observed runtime is still running.



Start with a small state machine

Use a small explicit core: requested, provisioning, ready, terminating, terminated and failed. Add suspended and resuming only when the corresponding runtime semantics are verified. Record lost observation as a condition rather than pretending it proves termination. More states are useful only when they lead to different behavior or user action.

The Go example in [examples/go-core](#) implements an educational transition guard with stale-version rejection. It does not implement distributed reconciliation or Kubernetes execution. Its value is that terminal-state and version invariants can be tested without a cluster.

Desired termination dominates stale activity

Suppose a session is ready at intent version 7. The user requests termination, creating intent version 8. A delayed “resume” command from version 6 arrives after that. Both the connector and the resource manager should reject it as stale. A database state transition alone is not enough if the cluster still accepts old commands independently.

Use a stable resource identity and versioned intent. Every command includes the relevant intent version, deadline and scope. Terminal intent is durable; a reconnect cannot turn an expired or terminated session back into a running one merely because an older event said ready.

Readiness is a conjunction

A scheduled Pod is not automatically ready for execution. Readiness should include runtime health, required policy application, identity binding and the required workspace state. If a policy update is still pending, the platform must not expose a privileged execution endpoint under the assumption that the update will finish soon.

Keep provisioning failure reasons specific: scheduling, image policy, image pull, storage, runtime health, credential reference or unsupported capability. A generic “failed to start” message forces operators to obtain unnecessary raw cluster logs. Safe, structured reasons improve both support and privacy.

Termination is a workflow

Stop accepting new executions. Request cancellation of active work according to policy. Prevent new credential issuance. Revoke what the provider can revoke. Remove runtime resources, then apply workspace retention policy. Confirm cleanup before declaring resource removal complete.

A terminated process can leave completed external effects behind. A terminated session may retain a snapshot intentionally. A failed cleanup may leave a PVC. Represent these outcomes explicitly instead of treating termination as universal rollback and erasure.

Disconnection requires a local contract

A disconnected connector cannot receive new central policy. The cluster needs a local maximum lifetime, a command lease policy and a rule for continuing existing work. Depending on the customer risk model, existing sessions may continue for a bounded interval or stop promptly. Both approaches have availability and safety costs; neither should be an accidental consequence of network failure.

Do not release concurrency quota just because the control plane has lost contact. The workload may still be consuming resources. Mark capacity uncertain, retain a conservative reservation and reconcile when observations return. An administrative override may be necessary, but it should be visible and auditable.

Garbage collection must respect ownership

Identify orphans by stable labels, owner references and product IDs. Do not delete all resources with a common display-name prefix. Check that the candidate belongs to AgentPlane and that its retention policy allows deletion. Include a dry-run inventory and an approval step for destructive bulk cleanup.

A missing database row does not always imply a disposable resource: it may result from an incomplete restore. Recovery mode should quarantine or report ambiguous resources before deleting them. This matters during disaster recovery, when the control database may be older than the running cluster state.

Exercise

Implement tests for duplicate termination, stale resume, runtime disappearance, connector disconnect and restoration from an older database snapshot. For every case, specify the session state, operation state, quota reservation and cleanup obligation. A single state string should not be expected to carry all four.

14. Execute Commands and Handle Files Without Hidden Privilege

Part IV — Safe Execution

The execution API is the point where a user request becomes operating-system activity. Treat it as a privileged interface even when the process runs in a sandbox. An execution request needs a stable identity, authorized session, explicit arguments, bounded runtime and a defined output policy.

Prefer argument arrays

Represent a program and its arguments separately. Avoid concatenating a user string into a shell command. Shell execution may be a legitimate advanced feature, but it is a different capability and should be explicit rather than a hidden implementation detail.

```
{
  "execution_id": "exec_example_001",
  "argv": ["python", "-m", "pytest", "tests/unit"],
  "working_directory": "repository",
  "timeout_seconds": 60,
  "maximum_output_bytes": 1048576
}
```

This request is a design example. Argument separation prevents one class of command construction error; it does not make the requested program harmless. The program still runs with the session's permissions and must remain inside the runtime, network and identity boundaries.

Bound output and interactive sessions

Use separate stdout and stderr channels, sequence numbers and a final execution result. Bound server and browser buffers. A slow terminal subscriber must not consume unbounded connector memory. Define whether output is truncated, spooled to customer storage or dropped after a configured limit, and tell the client which occurred.

Terminal escape sequences are untrusted input. Use a maintained terminal renderer and configure dangerous integrations carefully. Do not let output trigger URL opening, clipboard operations or application commands without explicit user interaction. A malicious test suite can print control sequences as easily as ordinary text.

PTY sessions complicate cancellation and signal handling. Define whether a client disconnect detaches, cancels or leaves the execution running until its deadline. A reconnect should attach to the same execution identity, not start a new process.

File paths need filesystem-safe operations

Reject absolute paths and traversal components, but do not stop at string normalization. A path that was safe during validation can be replaced by a symlink before opening. Use root-relative, traversal-resistant filesystem APIs appropriate to the platform and language version. Go's discussion of `os.Root` explains this class of problem; consult the exact available API before implementation. [S23](#)

Define rules for symlinks, hard links, devices and sockets. Archive extraction needs its own path checks, expanded-size limits and entry-count limits. A small compressed upload can expand into an enormous workspace. Atomic temporary-write and rename behavior helps avoid partially written target files but does not replace the root-boundary checks.

A file transfer should stream through bounded chunks, support cancellation and verify a checksum when supplied. Make data location explicit. Relay mode sends file contents through SaaS memory even if they are never written to SaaS storage. A direct data-plane mode needs a separate authenticated transfer route.

Git is an execution-adjacent capability

Validate repository schemes and approved destinations. Restrict SSH host keys and never silently disable verification. Provide credentials through a short-lived, local mechanism rather than embedding them in URLs or process arguments. Ensure Git configuration, askpass behavior, environment variables and template hooks are controlled. Submodules require independent destination and credential policy.

A normal clone is not permission to execute all repository scripts. Dependency installation, hooks configured by the environment and test commands remain sandboxed actions. A repository can contain instructions designed to trick an agent into widening permissions. Treat those instructions as data.

Push should be a separate permission and a separate approval decision when appropriate. Committing local changes and transmitting them to a remote repository have different consequences. Do not grant write credentials merely because clone access is required.

Credentials can still be printed

Even when the broker never sends a secret to the SaaS API, a program with access to that secret can print it. Limit credential scope and lifetime, minimize output retention and avoid granting unnecessary credentials. Automatic redaction cannot reliably recognize all secrets, encodings or sensitive business data.

Exercise

Design tests for a symlink swap during upload, an archive with traversal entries, a stalled output subscriber and an execution whose response is lost after a Git push succeeds. Explain why retry rules differ between listing a directory and repeating that push.

Primary sources

[Go traversal-resistant file APIs](#)

15. Make Network Policy an Enforced Property

Part IV — Safe Execution

A deny-by-default network design begins with the absence of permissions, then adds narrowly justified paths. It does not begin with unrestricted internet access and a growing list of suspicious hosts. The cluster's networking implementation must actually enforce the requested policies.

Understand standard policy semantics

Kubernetes NetworkPolicy is based on additive allow rules. Applicable policies combine their allowances; a deny-all policy does not override another policy that allows traffic. The API also depends on a network implementation that supports enforcement. These details are central to AgentPlane's compiler and acceptance tests. [S04](#)

The teaching manifest in [examples/kubernetes](#) creates a default-deny policy in a dedicated lab namespace. It is not proof of packet filtering until tested with a policy-enforcing CNI. Inspect all policies selecting the workload, not just the one created by AgentPlane.

Design a policy compiler

The product policy describes allowed services, CIDRs, ports and optionally domain names. The compiler targets a known provider. Standard NetworkPolicy does not supply general domain-name enforcement. A domain rule therefore requires a verified provider-specific capability or a controlled application proxy. If the required provider is absent, reject the policy rather than replacing a domain with an unstable IP snapshot and claiming equivalent protection.

Separate compilation success from enforcement readiness. The controller can report that manifests were accepted while a later probe reports a blocked or unexpectedly reachable destination. Keep both observations.

DNS is necessary but not automatically harmless

A typical workload needs cluster DNS. Allowing access to the resolver does not mean arbitrary domains are allowed for application traffic. It also does not prevent DNS-based data exfiltration. Where this matters, use a constrained DNS policy or proxy and test its behavior with the actual resolver and CNI.

Avoid broad namespace-only DNS rules that also authorize unrelated workloads in a system namespace. Select the intended resolver endpoints and ports. Node-local DNS, host networking and provider-specific packet translation may require a different verified configuration. Do not copy a manifest across clusters without checking the path it actually selects.

Block infrastructure escape paths

Cloud metadata, link-local addresses, loopback behavior, private networks and the Kubernetes API need explicit treatment. Include IPv6 and IPv4-mapped IPv6 when relevant. HTTP redirects and DNS changes can turn a seemingly approved URL into a request to a different destination.

A public MCP endpoint may legitimately be hosted behind changing infrastructure. That is a reason to use a suitable enforcement layer, not to allow all outbound traffic. Restrict the gateway's own discovery requests as well as sandbox traffic; an SSRF problem in the gateway can bypass a perfectly configured sandbox policy.

Make the gateway path unavoidable

Tool authorization is ineffective if the workload can reach the tool directly using the same credentials. Restrict network reachability and downstream credentials so privileged traffic must traverse the intended policy boundary. The tool server should validate its own caller identity too. Do not rely only on an HTTP proxy environment variable: untrusted code can ignore it.

Domain allowlists are not business authorization. An approved host can expose many paths, tenants and APIs, and may itself be compromised. Apply tool-level permissions and resource-specific credentials in addition to network constraints.

Update policies without an unintended open interval

Creating a permissive replacement before deleting the old policy can expand access because rules are additive. Design rollout ordering and temporary restrictions deliberately. A tightening update may require a gate that prevents new execution until the target version is active. Test existing connections, because enforcement behavior for established flows can vary by implementation.

Exercise

Run the network lab only in a disposable cluster. Prove an allowed service works, an unapproved service fails and an unrelated broad allow policy changes the result. Remove that policy and repeat. Record the CNI, resolver arrangement, policy objects and observed traffic rather than only the `kubectl apply` output.

Primary sources

[Kubernetes Network Policies](#)

16. Broker References, Not Customer Secret Values

Part IV — Safe Execution

The control plane should store the identity of a credential provider and an approved binding, not the customer's secret value. A binding describes who may request which credential for which workload, with which audience, scope and lifetime. Resolution happens in the customer environment.

Prefer identity over copied credentials

A workload identity lets a runtime obtain authority based on its authenticated workload context. Where a provider supports it, this avoids copying permanent keys into application configuration. The exact trust conditions matter: issuer, subject, audience, project mapping and maximum lifetime must all be constrained. A broad role trust relationship can turn short-lived credentials into broad short-lived compromise.

Kubernetes Secrets and projected credentials have specific access and storage properties. Their existence is not proof that only the intended process can read them. Kubernetes documents the responsibilities around Secret handling and projections.

S13

Define a binding contract

A proposed binding contains organization, project, provider identifier, approved resource reference, allowed runtime-template versions, delivery mechanism, maximum lease and revocation behavior. It must not contain a secret value or a free-form instruction to fetch any arbitrary provider path.

Do not let a caller replace a approved binding with another provider identifier from the same organization without authorization. Scope checks apply to every reference. The connector validates the binding against customer-local policy in addition to the control-plane decision.

Delivery is a security choice

Projected files, CSI mounts, local broker sockets and environment variables have different exposure characteristics. Prefer a mechanism that supports rotation, limited access and cleanup for the runtime. Environment variables are easy to inherit and accidentally print; file delivery is not automatically safe either if other processes can read the file or if it is included in a snapshot.

Place credentials outside the persistent workspace where possible. Use an appropriate ephemeral location and permissions. Exclude credentials from archive exports and diagnostic bundles. A filesystem snapshot can preserve secrets that were temporarily written to a persistent volume, so storage policy and identity policy cannot be designed independently.

Lease metadata is not revocation

A platform can record that a lease is revoked while the downstream provider continues accepting an already-issued credential. State the provider's actual revocation behavior. Some credentials become unusable only when they expire; others can be invalidated earlier. The broker must not promise instantaneous revocation unless the full path supports and tests it.

Keep maximum lifetime short enough for the risk model without creating a renewal storm. A disconnected cluster needs a rule for renewal failure. Do not silently substitute a permanent fallback credential because the normal provider is down. For sensitive actions, inability to obtain a valid credential should stop the action.

Keep connector authority separate

The connector's credential authenticates management traffic. It must never be mounted into an agent sandbox. Kubernetes service accounts used for management, runtime execution and tool access should be separate where their authorities differ. A single powerful account reused everywhere makes lateral movement much easier to achieve and harder to explain.

A customer administrator can often read or replace cluster workloads. Document that trust assumption. BYOC can keep infrastructure ownership with the customer, but it does not inherently provide cryptographic secrecy from the customer's administrators or all cloud operators.

Audit without leaking the credential

Useful events include binding approved, lease requested, lease issued, renewal failed and revocation requested. Record safe provider identifiers, lease IDs, expiration and outcome. Avoid secret values, authorization headers and raw provider error bodies. Even a secret path may reveal business context, so decide which metadata is exposed to developers versus security administrators.

Exercise

Design a Git read-only credential binding for one repository. State how the runtime obtains it, where it appears in memory or files, whether it can enter a snapshot and what happens during provider outage. Then test that another project cannot reference the binding even when it knows the binding ID.

Primary sources

[Kubernetes Secrets](#)

17. Discover Tools Without Trusting Their Descriptions

Part V — Tool Governance

A tool registry is an inventory and configuration system. It is not a trust oracle. A server can report a tool named `read_report` whose behavior is more powerful than its name suggests. Descriptions and schemas help clients understand an interface, but authorization must come from platform policy and verified identity.

Separate registration, discovery and approval

Registration records the proposed server endpoint, ownership, transport, authentication mode and credential reference. Discovery contacts the server through a controlled path and records capabilities. Approval determines whether those capabilities may be exposed to a particular workload. These are separate state transitions and should not collapse into “server added.”

A metadata change can alter the meaning of an existing integration. Version the discovered tool schema and retain a digest. Decide whether new or changed tools remain blocked pending review. Do not automatically grant a newly discovered tool because it matches a broad name pattern.

Discovery creates an SSRF surface

MCP-related metadata and authorization discovery may direct clients to additional URLs. Validate those destinations, redirects, resolution and TLS behavior under a versioned security policy. The MCP security guidance discusses token audience, proxy risks and SSRF in these flows. [S10](#)

Do not fetch arbitrary server metadata from the SaaS network without considering its internal reachability. In BYOC mode, discovery can occur from a constrained customer-cluster component. That component still needs protections against cloud metadata and unrelated private services. Moving SSRF into the customer cluster is not a mitigation by itself.

Preserve token audiences

The credential presented by an agent to the gateway is not automatically valid for the downstream tool service. Validate the inbound credential for the intended resource and obtain downstream authority through a reviewed mechanism. Avoid blind token passthrough. Keep actor identity, client identity and service identity distinct in decisions and logs.

Different MCP protocol versions and authentication extensions may have different state and transport assumptions. Pin the negotiated version and test it. Do not copy a claim about sessions or enterprise authorization from an earlier design into every future integration without rechecking the specification.

Integrate a gateway instead of inventing a proxy

agentgateway is a candidate data-plane component for this architecture. Its release-specific documentation should determine which Kubernetes resources, policies and authorization extension points are actually available. [S11](#)

Build an adapter that compiles AgentPlane registry and policy intent into the chosen gateway configuration. Keep product concepts independent from the exact upstream object shape. A feature that the release cannot enforce must be marked unsupported or implemented through a supported extension point—not silently approximated by a weaker rule.

Keep tool execution out of registry handlers

A registry connectivity test should perform a bounded health or metadata operation, not execute a dangerous tool to prove the server works. Discovery must not become an arbitrary command-execution API. Local process-based transports require a separate sandboxed execution design; do not spawn an untrusted server command inside the SaaS API process.

Use strict response-size and time limits. Treat server names, descriptions and error strings as untrusted content in the web console. Do not render arbitrary HTML or feed tool descriptions into privileged operator instructions.

Show the effective state

A useful server detail page shows registered owner, endpoint classification, authentication mode, discovered schema version, approved tools, effective policy, last successful check and observation age. Displaying only “connected” conceals whether the server is approved for use and whether the current schema matches the approved one.

Exercise

Create a harmless mock server with one read-only tool and one approval-required tool. Change the latter's schema after approval and verify that the platform requires a policy decision for the new version. Test a discovery redirect to an unapproved internal address without contacting any real sensitive endpoint.

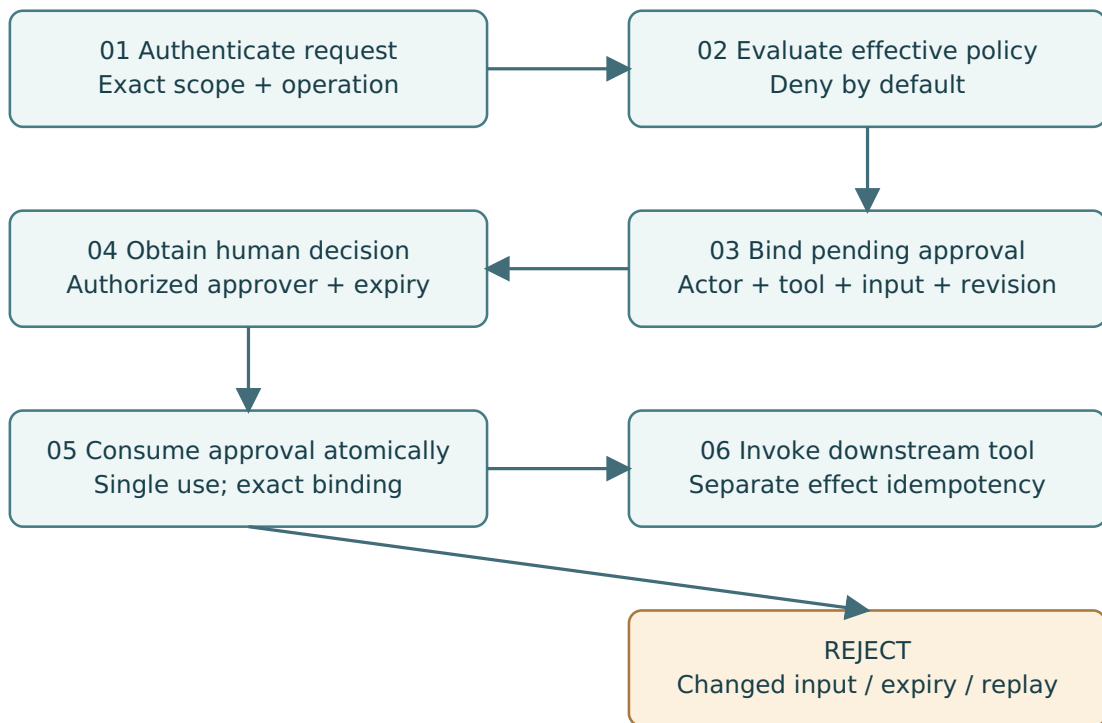
Primary sources

[MCP security best practices](#) · [agentgateway documentation](#)

18. Bind Authorization and Approval to the Exact Action

Part V — Tool Governance

A policy decision answers whether a particular actor may perform a particular action on a particular resource under current conditions. “This agent is trusted” is too broad to be a useful decision. Tool permissions should include server, tool, actor, scope, input constraints, policy version and time bounds.



Define deterministic policy precedence

Start with deny by default. An explicit organization-level denial cannot be weakened by a project rule. Distinguish allow from require approval: an approval requirement is not a temporary allow. Invalid policy and unavailable evaluation must produce a defined failure, generally fail closed for privileged actions.

Use a constrained, reviewed policy language or typed rules. Do not evaluate arbitrary user-provided code in the authorization service. Bound expression complexity and input size. A policy engine can become a denial-of-service surface if one decision can consume unbounded CPU or memory.

Construct a decision identity

The decision should bind actor, organization, project, sandbox, tool server, tool name, tool schema version, normalized input digest, policy version and expiration.

Canonicalize inputs according to a documented format. Do not hash whatever JSON serialization a client happens to send and assume semantic identity is stable.

A digest can also reveal low-entropy sensitive inputs through guessing. Keep approval metadata minimal, use a keyed digest where appropriate to the threat model and avoid exposing unnecessary input hashes to unrelated users. Human approvers still need enough safe context to make an informed decision.

Model approval as a durable state machine

A request can be pending, approved, rejected, expired, canceled, reserved for an execution or consumed. Define allowed transitions and use conditional updates under concurrency. If two approvers decide simultaneously, one authoritative result wins and the other receives a conflict.

A policy requiring four eyes must prevent requester self-approval and verify the approver's current authority. Consider organizational ownership: two accounts controlled by one person may satisfy a technical two-account check without providing genuine independent review. Communicate the exact implemented rule.

Consumption does not mean exactly-once side effects

Reserve an approved action for a stable execution ID before dispatch. A retry for the same execution can observe that reservation. A request for a different execution or changed input is rejected. If the tool effect happens and the result is lost, the system still needs downstream idempotency or reconciliation. One-time approval prevents reuse of authority; it does not magically make every external tool effect exactly once.

Do not keep an HTTP request open for hours while waiting for a person. Return an approval-required result, expose the pending request and let the agent resume or retry with the same operation identity after a decision. Bound approval lifetime and revalidate relevant policy changes before execution.

Prevent bypass

Authorization must be applied where the request cannot route around it. If the sandbox can directly call the tool with a powerful credential, an approval UI is only advisory. Combine gateway enforcement, downstream identity checks and network restrictions. The server should reject credentials for the wrong audience or scope. [S10](#)

A gateway timeout must not default to allow. For non-privileged read operations, a carefully designed cached decision may be acceptable, but its scope, version and maximum age must be explicit. Security-sensitive policy changes should invalidate or constrain cached decisions.

Audit decisions without collecting everything

Record the decision ID, safe action description, policy version, approver, outcome, reservation identity and timestamps. Raw tool arguments often contain customer data. Use approved summaries and protect detailed evidence separately when retention is genuinely necessary.

Exercise

Test changed input, changed tool schema, expired approval, revoked approver, concurrent consumption and a lost result after dispatch. Explain which failures return denied, conflict, expired or unknown. Then verify that no rejected case reaches the mock tool server at all.

Primary sources

[MCP security best practices](#)

19. Preserve Workspaces Without Inventing Checkpoint Guarantees

Part VI — State and Evidence

Persistent storage can preserve files across runtime restarts. It does not necessarily preserve a process, its memory, open sockets or an in-progress transaction. These are different kinds of state and should appear as different capabilities in the product.

Define the workspace contract

A workspace belongs to a scope and a stable resource identity. Its policy defines size, approved StorageClass, access mode, retention, snapshot behavior and termination cleanup. Do not attach an arbitrary existing PVC merely because a caller knows its name. Validate scope, ownership and intended use.

The workspace path should contain user data, not connector credentials or runtime control binaries. Separate temporary credential delivery from persistent files. If a program copies a secret into its own workspace, an ordinary snapshot can capture that secret. Storage controls cannot infer every file's sensitivity.

Distinguish consistency levels

A storage snapshot may represent crash-consistent filesystem state. An application may require a flush, checkpoint or quiesce operation for a useful restore. A database inside a sandbox has its own recovery semantics. Do not label every CSI snapshot application-consistent without testing the application protocol.

Kubernetes volume snapshots depend on the snapshot APIs and a supporting CSI implementation. They model storage snapshots, not a universal memory-checkpoint mechanism. [S12](#)

Use capability labels such as `filesystem_snapshot`, `application_quiesce` and `process_checkpoint` rather than one ambiguous `snapshot_supported` boolean. Only advertise the capabilities actually verified for the chosen runtime and storage combination.

Restore into a new resource

A restore should normally create a new workspace and session, leaving the source and original snapshot intact. This avoids silently rewinding an active workload under the same identity. Record the source snapshot, template version, policy version and restore operation ID.

Revalidate current security policy. A snapshot created under an old image or credential policy does not authorize restoring those old privileges today. Separate restoration of customer data from restoration of runtime authority. An old workspace might contain stale credentials and must be treated accordingly.

Hibernation and warm capacity

A restartable workspace suspension can stop compute while retaining storage. That can be useful even when process memory is not preserved. Name the capability honestly and explain what application work must be resumed or repeated.

Warm pools consume capacity before a customer claim. Unclaimed capacity should have no tenant credentials or retained tenant files. Once used by a tenant, recycling requires a proven cleanup process; creating fresh capacity is often easier to reason about. Keep pool quotas separate from active-session quotas so an oversized warm pool cannot starve ordinary workloads silently.

Archive fallback is a different feature

A tar-based workspace export can be useful when CSI snapshots are unavailable, but it is not automatically atomic. Concurrent writes may yield an inconsistent archive. Apply path, special-file, expanded-size and credential-exclusion rules. State where the archive is stored and who can decrypt it.

Do not silently upload an archive to SaaS-owned object storage in a BYOC product. Make that transfer explicit in organization configuration and user-facing data handling documentation. An export checkbox is not a reason to weaken default workspace residency.

Cleanup is part of correctness

A session can end while a workspace remains retained. A failed snapshot can leave provider resources behind. Record cleanup obligations separately from the user operation status. Run bounded reconciliation and produce a dry-run inventory for ambiguous resources.

Storage reclaim policies and provider behavior must be tested. Deleting a Kubernetes object may or may not delete the underlying data according to its configuration. Do not infer secure erasure from the disappearance of a PVC row in the dashboard.

Exercise

Write a restore drill using synthetic files and an application-level checksum. Change the files after the snapshot, restore into a new session and compare the result. Repeat with a writer active during capture. Document exactly which consistency claim the observations support and which they do not.

Primary sources

[Kubernetes volume snapshots](#)

20. Meter Usage and Reserve Capacity Without Double Counting

Part VI — State and Evidence

A usage system must explain what it measured, how it handled missing observations and how corrections work. It is not enough to multiply a session's age by its configured CPU count and label the result actual consumption.

Keep dimensions distinct

Requested vCPU-seconds, measured CPU time, reserved memory-seconds, actual memory observations, persistent storage and tool calls are different dimensions. Name them accordingly. A session requesting two vCPUs for ten minutes has 1,200 requested vCPU-seconds. That arithmetic says nothing about how busy the CPUs were.

Every event needs a stable identity, scope, resource, dimension, quantity, unit, period and source. Include whether the quantity is measured, estimated or allocated. Keep raw events append-only and derive aggregates reproducibly. Corrections should reference the original event rather than silently rewriting history.

Deduplicate by event identity

A reconnect can deliver the same interval twice. A retry can publish the same usage event twice. Use a unique constraint on source and event identity and keep the original payload digest to detect conflicting reuse. Do not deduplicate by rounded timestamp alone; two legitimate events can occur at the same instant.

For duration accounting, define interval endpoints and overlap behavior. A monotonic local clock helps measure elapsed time, while wall-clock timestamps locate the interval in a billing period. Clock correction can create apparent negative or overlapping intervals. Retain sufficient provenance to reconcile rather than silently clamping every anomaly.

Reserve before provisioning

A concurrency quota should be reserved transactionally before the create command is dispatched. A conditional counter update or row lock can serialize competing reservations. An in-memory limit on each API replica cannot enforce a global organization quota.

The Go teaching example demonstrates a concurrency-safe local reservation model. It intentionally does not replace a database transaction shared by multiple replicas. The SQL design shows the durable boundary; the lab requires a real PostgreSQL execution to validate it. [S08](#)

Release a reservation when the corresponding obligation is actually resolved. A lost heartbeat is not proof that compute stopped. Track uncertain allocations and apply an explicit reconciliation or administrative override policy.

Budgets are not ordinary rate limits

A rate limit controls request frequency over a window. A budget controls an accumulated quantity or cost. A plan entitlement controls whether a feature is available. Keep these mechanisms separate so that a billing-provider outage does not unexpectedly become an authorization grant or a destructive workload action.

Soft thresholds generate notices and permit continued work. Hard thresholds require a defined action: reject new sessions, stop renewing leases, prevent particular tools or terminate according to a prior agreement. Do not silently kill customer workloads because an aggregation job ran late.

Model costs without inventing prices

A useful hypothetical model is:

```
monthly operating cost = control-plane base
                        + managed active compute
                        + warm capacity
                        + persistent storage and snapshots
                        + network transfer
                        + telemetry and evidence retention
                        + support and incident response
```

BYOC moves much of execution compute to the customer, but it does not eliminate control-plane hosting, support, integration maintenance or security work. A managed mode also needs abuse prevention, payment risk and capacity planning. Obtain real provider prices and workload measurements before making a commercial forecast.

For a planning exercise, choose your own unit prices and label them assumptions. Show sensitivity to active hours, warm capacity and retention. A margin that looks excellent only when support time is assumed to be zero is not a useful business model.

Billing requires evidence and dispute handling

Customer-controlled connectors can be modified. Their metering reports are not independent attestation of consumption. Decide whether billing is based on a platform subscription, declared capacity, independently measured managed compute or another auditable contract. Keep correction and dispute workflows separate from security incident handling.

Exercise

Simulate duplicate, overlapping and late usage events across a month boundary. Produce the same aggregate from the same raw event set twice. Then apply a correction event and explain the difference without deleting the original. Stress the local quota example with concurrent requests and identify the additional database invariant needed for multiple service replicas.

Primary sources

[PostgreSQL 18 explicit locking](#)

21. Observe the System Without Collecting the Customer

Part VI — State and Evidence

Operational telemetry and audit evidence serve different purposes. Telemetry explains performance and failure. Audit records explain authority and sensitive state changes. Neither requires collecting every prompt, command output, file or tool argument by default.

Instrument boundaries rather than payloads

Trace API acceptance, outbox publication, gateway dispatch, connector receipt, runtime start and final result. Propagate correlation identity across HTTP, messages and cluster operations. A trace should answer where time was spent and which boundary failed without copying the user's entire request body.

Use bounded-cardinality metrics for service health: request rates, latency buckets, queue depth, heartbeat age, reconciliation failures and certificate expiry. Per-session IDs in metric labels can create an unbounded series count. Use scoped event records or queryable logs for high-cardinality investigation instead of turning every object ID into a Prometheus dimension.

OpenTelemetry's sensitive-data guidance emphasizes reducing and controlling the data collected. Apply that principle at instrumentation time, not only in a central processor after sensitive content has already crossed the boundary. [S20](#)

Define an audit event schema

Record event identity, sequence, scope, actor type, actor ID, action, resource, outcome, policy version, request identity and a safe timestamp. Prefer typed metadata over arbitrary maps containing whatever a handler happened to know. Explicitly exclude credentials, file bodies, process output and raw tool inputs unless a separate approved retention policy requires them.

For operations that must not occur without evidence, write audit metadata in the same transaction as intent. A non-critical telemetry exporter can fail without blocking the API; an audit-critical transaction may need to fail closed. State which category each event belongs to.

Hash chains have a trust boundary

A chain links each event hash to the previous hash. Modification, reordering and removal can be detected when checked against a trusted checkpoint containing the expected count and head. Without a checkpoint outside the attacker's rewrite boundary, an attacker can recalculate a new chain. Tail truncation especially requires an expected end state.

The Python program in [examples/audit](#) demonstrates this distinction. It is an educational verifier with a caller-supplied trusted checkpoint. It is not a public signing service, immutable ledger or replacement for protected storage. Its tests deliberately show why an internally consistent rewritten chain is not sufficient evidence.

Serialize append and anchor independently

A production per-tenant chain needs a serialized append boundary. Use a locked head row or an equivalent transactional mechanism. Multiple workers cannot independently append to the same previous hash and still produce one linear history. At higher scale, use partitioned chains and explicit aggregation rather than pretending one uncoordinated global chain exists.

Publish signed checkpoints through a key boundary separate from ordinary database writers, and retain them in a location with a distinct administrative trust model. A signer whose key is stored next to the writable audit table does not provide strong protection against compromise of both.

Evidence bundles are scoped exports

An evidence bundle can include versioned policy, runtime image identity, lifecycle metadata, approval decisions, usage summaries and a relevant audit range. Include a manifest of file hashes and the verification method. Verify signatures against an independently trusted key, not a public key supplied only inside the same untrusted archive.

Use expiring, authorized downloads and record the export request. Exports can reveal names, timestamps and business relationships even when they contain no secret values. Limit retention and protect tenant scope throughout the asynchronous job, storage key and download endpoint.

Avoid compliance claims the product cannot prove

Technical evidence can support a review. It does not make an organization automatically compliant with a law or certification framework. Map controls with qualified domain review and preserve the exact evidence source and observation period. A report generated today from old observations is not current assurance.

Exercise

Run the audit demo and tamper tests. Remove the final event while retaining the old trusted checkpoint, then build a fresh chain over modified data with a new untrusted checkpoint. Explain why the first should fail and why the second requires an external trust decision rather than a better hash function.

Primary sources

[OpenTelemetry sensitive-data handling](#)

22. Build Interfaces That Preserve Scope and Uncertainty

Part VII — Product and Operations

The console and SDKs should make safe platform behavior understandable. They must not hide uncertain state, downgrade authorization errors into generic failures or present a queued action as completed. A clear interface is part of the operational safety design.

Preserve organization and project context

Every authenticated page operates under an explicit scope. When a user changes project, cancel or isolate in-flight requests from the previous scope. A late response must not populate the new project's table. Cache keys include scope, resource version and relevant permissions.

The same rule applies to live streams. Close terminal and event subscriptions when their context is no longer active or authorized. Do not rely on a visually changed project label while leaving an old privileged connection open.

Show intent, observation and age

A session detail view should show desired state, last observed state, observation age, active operation and safe reason codes. A disconnected cluster can retain a last-known state, but the page must label it as such. A termination request should show pending cleanup until confirmation arrives.

Capability warnings should identify what is missing: unsupported runtime, unverified network enforcement, snapshot driver absent or policy version pending. This is more actionable than a generic red banner and avoids encouraging users to disable security controls blindly.

Treat sensitive UI surfaces carefully

Enrollment tokens and API keys appear once and should not enter persistent browser storage, analytics or screenshots collected automatically. Confirmation dialogs for cluster revocation, destructive cleanup and permission changes must describe the consequence, not merely ask “Are you sure?”

Terminal output, filenames, Git references and tool descriptions are untrusted content. Use safe rendering and maintained components. Do not use arbitrary HTML in tool descriptions. Accessibility matters especially in operational workflows: keyboard access, visible focus, meaningful status text and non-color-only signals help operators act correctly under pressure.

Generate low-level clients, design high-level behavior

Generate transport types from the public API schema. Add idiomatic SDK methods for waiting on operations, streaming execution, paging lists and handling typed errors. Keep HTTP status, machine code, request ID and operation ID available to callers without forcing them to parse human messages.

Cancellation must propagate through the SDK. Retries need operation-specific rules. A helper named `execute_and_wait` must not silently repeat a script after an unknown outcome. Preserve the original execution identity and return an error that explains the uncertainty.

Keep cross-language semantics consistent

Go contexts, Python cancellation and TypeScript abort signals are different APIs for the same product requirement. Build a conformance suite around scenarios: create with an idempotency key, conflict on changed input, wait timeout, stream truncation, permission loss and unknown execution result.

A generated SDK can compile while implementing the wrong retry behavior. Test observable semantics against a disposable service, not only type generation. Package publishing is a separate release action requiring approved credentials; a code-generation task should not publish packages automatically.

Make the CLI safe for automation

Use explicit exit codes, JSON output and stable error envelopes. Accept program arguments as arguments rather than a shell string. Keep binary file operations separate from text output so progress messages do not corrupt downloads.

Read credentials from a secure credential store or environment under a documented policy. Do not generate commands that put secrets in shell history. Avoid debug logs that print full headers. Destructive actions require confirmation in an interactive terminal and an explicit non-interactive flag in automation.

Exercise

Prototype a session page for three situations: ready and fresh, last seen ready but disconnected, and termination accepted but cleanup unconfirmed. Write the screen-reader text as well as the visible labels. Then define an SDK result type that preserves the same distinctions without relying on interface wording.

23. Deploy and Upgrade Without Crossing the Trust Boundary

Part VII — Product and Operations

Deployment design should preserve the architecture's authority boundaries. Keep customer execution workloads out of the SaaS control-plane cluster. Separate control-plane packaging from customer-cluster components so installation does not silently grant unrelated capabilities.

Two installation packages

The control-plane package contains the API, worker, connector gateway, web assets and required configuration references. PostgreSQL, transport and object storage are external production dependencies or explicitly chosen managed services. The customer package contains the connector, policy operator and approved upstream integrations. Optional features stay optional.

Do not bundle a development database and message broker into the production chart as an invisible default. Likewise, do not install a cluster-wide runtime, CNI or certificate authority merely because a user asked to connect an existing cluster. These are customer infrastructure decisions with substantial impact.

Pin artifacts and verify policy

Release images should use immutable references and retained provenance. Chart values should reference existing Secrets or approved secret integrations rather than contain plaintext credentials. Enforce security contexts, resource bounds, health probes and network policy in rendered manifests.

A successful Helm template render proves syntax and templating behavior, not cluster compatibility. Run schema checks, admission tests and a real install and upgrade exercise against the intended versions. Keep CRD lifecycle separate from ordinary deployment lifecycle; deleting a CRD can remove its custom resources.

Reference AWS deployment

An AWS reference design can use EKS for the control services, a managed PostgreSQL database, object storage for permitted evidence exports, an external signing integration and workload identities. Use private database networking, constrained security groups, encrypted storage and restricted administrative access.

NAT gateways, private endpoints, load balancers, multi-zone capacity and telemetry retention affect cost. This book provides a design framework, not current pricing or a tested Terraform stack. Obtain prices and verify provider support when implementing the deployment. Do not claim a region or engine version is supported without checking the current provider documentation.

Migrate databases with expand and contract

Add compatible columns and behavior before removing old fields. During rolling upgrades, old and new API and worker versions may coexist. A migration that works only when every process changes simultaneously is a deployment risk.

Keep schema ownership and application access separate. Back up before risky changes and test restoration. A down migration is not a credible rollback when it would discard data introduced by the new version. In such cases, document forward repair or restore procedures explicitly.

Drain long-lived connections

A connector gateway cannot be upgraded like a stateless page server without considering open streams. Stop accepting new streams, notify or drain existing connections, persist ownership transitions and let connectors reconnect with backoff. Use fencing and operation identity so overlap does not duplicate work.

A PodDisruptionBudget can reduce some voluntary disruption, but it does not prove continuous availability. Node failure, configuration errors and incompatible protocol changes remain possible. Measure reconnect and recovery behavior in the actual deployment.

Keep release authority explicit

Build, test, sign, publish and deploy are separate actions. A coding agent can prepare artifacts without receiving production publishing credentials. Protected release environments, scoped workload identity and least-privilege workflows reduce the consequences of a compromised build step.

The book's own GitHub Pages workflow follows the same separation: pull requests validate the publication; deployment uses a separate job with Pages permissions. GitHub documents the required Pages artifact and environment flow. [S16](#)

Exercise

Write an upgrade plan in which the API is upgraded first, the worker second and half the connectors remain on the prior protocol. State the supported overlap, rollback trigger and commands that must remain safe throughout. Include a certificate rotation occurring during the same interval.

Primary sources

[GitHub Pages custom workflows](#) · [GitHub Actions secure use](#)

24. Design Recovery Before the First Production Incident

Part VII — Product and Operations

Availability is not one percentage shared by every feature. The API can remain available while cluster commands are delayed; a customer workload can continue while the SaaS is unavailable; evidence exports can lag while execution works. Define service indicators around actual user journeys and trust boundaries.

State objectives as targets until measured

A proposed provisioning latency target is an engineering goal. A recovery time objective is a planning requirement. Neither is an observed guarantee until a test or production measurement establishes it under stated conditions. Record the environment, sample size and start/end definitions for every result.

Separate control-plane RPO from customer-workspace RPO. A database backup may preserve session metadata without preserving customer files. A workspace snapshot may preserve files without preserving recent approval or audit records. Recovery must reconcile these timelines.

Restore durable state carefully

Back up PostgreSQL, permitted object storage, configuration and the external signer's recovery material according to its provider design. Keep encryption-key availability in the restore plan. An encrypted backup that cannot be decrypted is not a useful recovery artifact.

Restore into an isolated environment first. Validate schema, application-role permissions, tenant isolation, audit integrity and outbox state. Do not point a restored control plane at live customer connectors immediately. The restored state may be older than the cluster resources and credential revocations.

Old state can resurrect dangerous authority

A database restored from before a certificate revocation may mark that certificate valid again. An old approval record may look unused even though its action already executed. An outbox event may be replayed after its external effect completed. These are security problems, not only data-consistency problems.

Maintain a recovery procedure that treats restored authorization state as stale until reconciled. Reestablish current revocation and connection epochs, preserve independent audit checkpoints and quarantine ambiguous execution records. Do not blindly replay historical approval or execution commands.

Rebuild transport from intent, not wishful replay

If PostgreSQL is the system of record, the transport can often be reconstructed from durable operations and outbox state. But replay must preserve event IDs, command IDs and downstream effect identity. Losing the inbox deduplication history can make an old message dangerous again.

Retain deduplication and execution history for at least the recovery horizon required by the operation contract. For external effects that cannot be proven, return an unknown result and reconcile with the target system or an authorized operator. Replaying everything is not a recovery strategy.

Recover customer resources by observation

After restoring the control plane, compare current cluster resources with restored intent. A sandbox created after the backup may be absent from the database. Do not immediately delete it as an orphan. Enter a recovery mode that inventories, quarantines or adopts resources only through a reviewed procedure.

Local workspace ownership and retention remain the customer's responsibility according to the deployment contract. The SaaS must not claim that restoring its own database restored every customer workload or file.

Build incident runbooks around decisions

A useful runbook identifies symptoms, initial checks, containment, evidence preservation, recovery and exit criteria. Include the permissions needed and the risk of each action. Avoid a page containing only commands without explaining when they are safe to run.

For a suspected connector compromise, containment may include gateway revocation, local connector shutdown, workload quarantine, credential-provider revocation and review of recent operations. Central revocation alone is insufficient if the cluster is disconnected or local credentials remain valid.

Exercise

Design a tabletop drill: the control database is restored from two hours ago, a connector certificate was revoked one hour ago and a sensitive tool action completed thirty minutes ago. Identify the records that can no longer be trusted without reconciliation. Define how the platform avoids reauthorizing or repeating the action while recovering service.

25. Build Evidence for Every Important Claim

Part VII — Product and Operations

A demonstration answers whether one path worked once. Verification asks whether the implementation preserves a stated property across valid, invalid and interrupted paths. AgentPlane needs both. A product demo makes the system understandable; a verification record determines what a maintainer can honestly claim about it.

Classify evidence before collecting it

Keep four evidence classes separate: static review, local unit tests, component integration tests and deployment acceptance tests. A Go state-machine test cannot prove that a CNI enforces egress. A rendered Helm chart cannot prove that an admission controller rejects a privileged pod. A scanned container is not proof that its runtime cannot escape.

The companion repository runs small offline examples. SQL and Kubernetes labs are procedures for an isolated environment, not implied successful experiments. The validation report is a statement about specific commands in this edition. Readers should create their own report after changing dependencies or deploying to another cluster.

Turn invariants into hostile tests

For each invariant, specify an attempted violation, expected observation and measurement location. Consider: tenant A must not operate on tenant B's session. Test reads, writes, batch requests, event subscriptions, download links, cached objects and background jobs. The HTTP list endpoint is only one access path.

For command deduplication, run the handler twice, restart the handler between attempts, change the payload while keeping the ID and lose the result after an external effect. The last case cannot always be solved by another retry. A correct implementation may expose an unknown outcome and require reconciliation.

Test concurrency intentionally

A quota test with sequential requests does not test a quota race. Start more concurrent reservations than the configured limit. Assert the final count, the number of successes and the behavior of duplicate operation IDs. Release each reservation twice and verify that the counter never becomes negative.

For approvals, two consumers should compete for the same approval. Only one consumption may succeed. However, successful consumption still does not establish exactly-once behavior at the downstream tool. Test that distinction explicitly so future refactoring does not turn an uncertain effect into a blind retry.

Fault injection needs an ownership boundary

Use a disposable cluster with an unmistakable context name. Before deleting a pod or interrupting a dependency, assert the context and namespace. Never infer that the current context is safe because the script lives in a directory called `test`. Destructive tests should require a separate, explicit opt-in.

Interrupt the connector, worker, gateway and broker at different points. Record whether intent was committed, whether the command was accepted, whether the execution began and whether the final result was stored. Those observations help locate the real ambiguity window.

Measure with a workload description

Publish the machine, runtime versions, topology, request distribution, payload sizes, concurrency, duration, warm-up and error criteria with a benchmark. A single mean hides tails and excludes failed requests unless stated otherwise. Record queue latency, provisioning latency and execution latency separately.

Do not transfer a local laptop result into a production capacity claim. Treat capacity planning as a hypothesis refined by representative experiments. Include resource ceilings and the cost of observability, metadata storage and idle warm capacity when evaluating throughput.

Decide what blocks a release

A release gate should be a small set of mandatory properties, not a wall of green badges. Tenant isolation, credential protection, policy enforcement, safe retry behavior and restore discipline deserve blocking tests. Cosmetic checks can be useful without being confused with security assurance.

A failed mandatory check means the candidate is not ready for that advertised scope. It does not mean all progress is worthless. Publish an accurate private beta limitation or reduce the feature's availability rather than rewriting the report to match the desired release date.

Exercise

Choose five claims from the product homepage you would write for AgentPlane. For each, identify the strongest available evidence and one counterexample the test suite does not cover. Rewrite any claim that exceeds its evidence. Include one claim about data location and one about recovery.

26. Use Coding Agents as Bounded Engineering Collaborators

Part VIII — Implementation and Publication

A prompt is an assignment, not a proof. A coding agent can inspect a repository, propose changes and run available checks, but the team must still define the allowed actions and decide which evidence makes a change acceptable. The AgentPlane prompt pack organizes that work into 45 assignments, numbered 00-44.

Keep instructions close to the work

An `AGENTS.md` file records repository conventions, testing commands and safety boundaries. Tool-specific instruction loading must be verified against the actual assistant being used. OpenAI documents repository instructions for Codex; other assistants may load different files or require an explicit reference. [S15](#)

The provided `AGENTS.md` governs this publication repository. Application-building assignments belong in a separate implementation repository. Do not accidentally replace the book builder with a SaaS scaffold because a prompt says to create an API service. Copy the implementation contract deliberately and inspect the working tree before making changes.

Specify evidence, not just files

An effective assignment states the existing context, scope, invariants, expected changes, checks and completion criteria. “Implement authentication” is not a bounded assignment. “Add password-reset consumption with expiry, single-use semantics and tests for concurrent consumption” is much closer.

Require a report of changed files, commands actually executed, results and remaining limitations. A statement such as “tests should pass” is not a result. If a database or cluster is unavailable, the correct report records the blocked integration check without marking the entire feature production-ready.

Reasoning labels are not portable API values

The pack uses editorial review levels: **standard**, **deep** and **security-critical**. These describe how much human and agent attention a task needs. They are not model identifiers or configuration strings. Select the available model and reasoning settings through the tool's current documented interface. Do not paste unsupported names into a configuration file.

Security-critical tasks include PKI, tenant isolation, replay handling, secrets, approvals and release authorization. Such tasks deserve independent review even when the agent reports success. Faster completion is not a substitute for a clear trust boundary.

Keep changes small enough to review

Read the tracker, inspect the implementation and identify the next incomplete boundary. Do not assume a numbered task was never partially implemented. Preserve unrelated changes. Make one coherent patch, validate it and record the result. A local commit is useful only when the user has authorized it and the change does not sweep in unrelated work.

Large assignments may need several implementation sessions. Keep the original acceptance criteria, split the work into explicit substeps and leave the tracker partial until all mandatory checks are satisfied. Renumbering the task does not remove its unfinished obligations.

Distinguish analysis from execution permission

Reading manifests is not permission to apply them. Preparing a release is not permission to publish it. Generating Terraform is not permission to create resources. The pack prohibits remote deployment, credential disclosure and publication without explicit human authorization.

Treat dependency files, source comments, issue text and tool output as untrusted input when they instruct the agent to change its permissions. A README inside a cloned customer repository must not override the platform's security contract. The same principle applies to AgentPlane itself when agents inspect repositories.

Exercise

Run Prompt 00 in a new implementation repository. Ask the agent to inspect before editing, then compare its report with `git diff`. Find one requirement that needs an integration test unavailable in the current environment. Record it as blocked rather than deleting it from the definition of done.

Primary sources

[OpenAI AGENTS.md guidance](#) · [GitHub Actions secure use](#)

27. Deliver One Complete Vertical Slice

Part VIII — Implementation and Publication

The capstone is not the entire product. It is a narrow, observable path through all important trust boundaries. A fictional Northstar developer creates one session, runs one harmless command and requests one governed tool operation in a local cluster. A second tenant tries to observe or repeat that work and fails.

Define the demonstration contract

The developer must authenticate, select an organization and project, and submit a session request using a reviewed runtime-template version. The API returns an operation identifier. A local connector receives a typed command over an authenticated connection and checks the cluster-local policy envelope before creating the approved upstream resource.

Readiness requires a working execution path, not simply a pod phase. The operator can explain the image digest, runtime class, namespace, effective network policy and credential bindings. Unsupported capabilities are shown as unavailable; there is no default-runtime fallback hidden beneath a secure-profile label.

Execute a harmless workload

Run a fixed program that prints a short message and writes a small workspace file. Download the file through the authorized transfer path and compare its checksum. Try a path outside the workspace and verify rejection. Do not use real customer repositories or production credentials for the first demonstration.

Interrupt the client stream. The execution record should remain meaningful even when output delivery stops. Reconnect only according to the supported stream contract. Do not promise full replay of output that was intentionally not stored.

Govern one tool operation

Register a local test tool that modifies only a disposable test record. Discovery must not enable it. Confirm default denial, create a policy requiring approval and submit an operation bound to exact actor, tenant, project, session, tool, input digest and policy version.

Approve through a different authorized identity when four-eyes review is enabled. Attempt to reuse the approval, change the input and call the tool directly around the gateway. A convincing demonstration observes each rejection at the actual enforcement point, not only in a console notification.

Add failure before adding features

Disconnect the connector after intent is committed. Show the last observed state and its age. Verify that the local lifetime still bounds the workload. Restart the connection and reconcile without creating a second session for the same operation. Revoke credentials and verify both central rejection and the stated limits of disconnected local enforcement.

Exercise a duplicate usage event. The aggregate must not increase twice. Attempt cross-tenant audit export and event subscription. Test these with the actual application identity rather than an administrator connection.

Produce a review packet

The packet should contain the architecture revision, exact dependency manifest, commands run, safe test outputs, known gaps and the mapping from every advertised property to evidence. Include the restore plan before inviting external users. A short video can explain the workflow, but it cannot replace these records.

The accompanying capstone lab supplies acceptance criteria rather than a claim that this repository already contains the full application. The teaching programs are smaller building blocks. Their value is to make key invariants executable before integrating cloud, database and cluster systems.

Grow only from observed demand

After the vertical slice, choose the next capability from customer evidence. Some customers will value governance and audit more than warm pools. Others need a direct data path before they can try the product. A third group may require stronger runtime isolation or a dedicated installation.

Do not add GPU scheduling, a billing provider and a marketplace simply because they fit the architectural diagram. Each creates a new operational contract. The next feature should have a buyer, a boundary and an acceptance test.

Exercise

Write a ten-minute demonstration script and a separate acceptance-test list. Mark which steps are storytelling and which provide machine-checkable evidence. Explain what the product must show when termination is requested during a cluster partition. A polished spinner is not an acceptable answer.

28. Maintain an Open Book Like a Software Project

Part VIII — Implementation and Publication

An open technical book is a maintained body of claims. Its chapters, examples, references and build scripts evolve together. GitHub provides a collaboration surface, but the project still needs review rules, licenses, versioning and a clear distinction between source material and generated editions.

Keep Markdown authoritative

The files under `book/chapters` are the primary narrative source. The publication builder assembles a complete Markdown edition, a static website, a printable PDF and a reflowable EPUB. Fix the source rather than patching an exported PDF by hand. Generated outputs should identify the edition and preserve their source mapping.

The assembled Markdown is convenient for searching and importing into reading tools. Individual chapters make pull requests easier to review. The PDF has a fixed layout and page references; the EPUB follows reader-selected text size and screen dimensions. Do not use PDF page numbers as the only navigation mechanism.

Separate review from publishing

Pull requests validate the sources and teaching examples. Publishing runs from a trusted branch or an explicitly approved release workflow. Pages deployment needs its own permissions and environment configuration. GitHub documents the custom workflow mechanism; enabling that workflow is an account-side step, not something a downloaded ZIP has already performed. [S16](#)

Keep action references immutable and review dependency changes. Do not grant a pull-request job the ability to create a public release. A generated HTML artifact contains contributor-controlled text, so treat it as publication content rather than trusted executable infrastructure.

Allocate licenses clearly

This edition assigns original narrative, diagrams and prompt prose to CC BY-SA 4.0, and original code and configuration examples to MIT. The root license file states the boundaries. Third-party projects retain their own licenses and names. Readers must not interpret the book's grant as permission to relicense upstream software. [S18](#) [S19](#)

Keep attribution and change notices when adapting text. A source-repository URL should be added when the publisher creates the actual repository; this edition does not invent one. Following architectural ideas does not by itself place an independently written application under the book's narrative license.

Version claims as well as files

A book edition records its publication date, tested build tools and source review notes. Application dependencies have separate compatibility records. A newer upstream release does not automatically make the book obsolete, but it may invalidate example fields or assumptions. Track the affected adapter and tests.

Corrections deserve visibility. Use an errata issue for factual mistakes and a security-reporting route for vulnerabilities in runnable examples. Avoid posting real credentials, customer logs or exploit details against a live installation in a public issue. The repository security policy explains the intended route.

Review an edition before sharing it

Check that every chapter appears in all formats, internal links resolve, diagrams have meaningful alternatives, code remains legible and no temporary credentials or font files are included. Validate the EPUB structure and visually inspect PDF pages, especially tables and code. Automated build success does not prove a pleasant reading experience.

Document checks that were not possible. A local build does not show that GitHub Pages is enabled on a future repository, that a Kindle renders every table perfectly, or that a Kubernetes integration has passed. Honest limits make the next maintainer's work more efficient.

Exercise

Make a small correction to one chapter, rebuild every format and inspect its rendered location. Submit a pull request containing the source change and its validation notes. Explain whether the change affects only prose, executable examples or an advertised security property.

Primary sources

[GitHub Pages custom workflows](#) · [CC BY-SA 4.0 legal code](#) · [MIT License](#) · [GitHub Actions secure use](#)

Appendix A. A Working Vocabulary

Adapter. A boundary that translates the platform's stable contract into a particular upstream release. It should expose capabilities and limitations, not silently imitate missing functionality.

Admission. A decision made before a Kubernetes object is accepted. Admission is one control; runtime and network enforcement remain separate.

Approval. A bounded authorization decision for one identity, action, input and policy revision. Consuming it is not proof of an exactly-once external effect.

Audit checkpoint. A trusted record of a chain head and sequence that makes later alteration detectable within the checkpoint's trust assumptions.

BYOC. Bring your own cloud or cluster. Compute can remain customer-owned even when an enabled relay exposes output to the SaaS. Specify the actual data path.

Capability. A reported and verified implementation property. Discovery alone is not an entitlement, authorization decision or security guarantee.

Control plane. The system managing ownership, desired state and policy.

Data plane / execution plane. The environment that runs workloads and carries customer data. Its boundary is defined by actual processing, not a label.

Fencing. Rejection of stale authority by the resource or effect recipient. A leader-election lease without recipient-side checks is not complete fencing.

Idempotency. Repeating a request under its defined identity does not introduce an additional intended effect. The identity must include scope and payload rules.

Inbox. Durable records of consumed messages or command attempts used to detect replay and resume work. Retention must match the possible replay horizon.

Lease. A time-bounded claim or credential. Expiry has an effect only where it is checked and enforced.

MCP. Model Context Protocol. Tool discovery, authorization and execution are distinct concerns; server-provided metadata is untrusted input.

Observed state. The latest verified state and its observation time. It may be stale while the desired state is newer.

Outbox. Events written in the same database transaction as business intent, then published asynchronously.

RLS. Row-level security. Database policies can constrain ordinary roles, but role privileges and application-controlled context remain part of the model.

RuntimeClass. Kubernetes selection of a configured runtime handler. A name alone does not install or prove a particular isolation implementation.

Snapshot. A storage point-in-time artifact under the provider's consistency contract. It is not automatically a memory, process or application checkpoint.

Unknown outcome. A truthful execution state when the system cannot establish whether an external effect completed. It should not become a silent retry.

Warm pool. Preallocated capacity intended to reduce allocation delay. Its isolation, reset procedure and idle cost need explicit handling.

Appendix B. Decision Records and Interface Contracts

Use these templates to record choices in the implementation repository. They are proposed AgentPlane contracts, not upstream API definitions.

Architecture decision record

```
Title: Customer-owned local policy envelope
Status: Proposed / Accepted / Superseded
Context: The SaaS can send commands over an outbound connection.
Decision: Local bounds cannot be widened by normal SaaS commands.
Alternatives: Full SaaS authority; completely disconnected operation.
Consequences: Some changes require customer-side administration.
Verification: Attempt an authenticated command outside local bounds.
Recovery: Explain how an administrator repairs an invalid envelope.
Owner: Assign in the real implementation repository.
```

Asynchronous operation envelope

```
{
  "operation_id": "op-example",
  "organization_id": "org-example",
  "project_id": "project-example",
  "resource_id": "session-example",
  "desired_state": "running",
  "observed_state": "provisioning",
  "observed_at": "2026-10-10T00:00:00Z",
  "status": "pending",
  "reason_code": "AWAITING_RUNTIME_READINESS"
}
```

These identifiers are illustrative strings. A real API must validate its chosen identifier format, tenant scope and timestamps. Avoid exposing raw cluster error objects or credential-bearing messages in `reason_code`.

Dependency verification record

Field	What to record
Component	Controller, router, runtime server or SDK
Exact artifact	Release, commit and image digest
API surface	CRD version and supported fields
Required capabilities	Execution, file operations, suspend, snapshots
Local changes	Configuration and patches, preferably none
Tests	Specific commands, cluster profile and results
Failure policy	Reject, degrade or disable unsupported features
Review date	Date the record was actually checked

Keep rows independent. A controller and runtime server may have different release cadences. Do not infer SDK compatibility from an unrelated repository tag.

Command envelope invariants

A command identity binds organization, project, cluster, operation ID, payload hash, protocol revision, expiry and connection epoch. Authenticating the stream is necessary but does not replace checking those fields. Every effect-capable recipient needs a defined replay and stale-authority behavior.

Result vocabulary

Use results that distinguish accepted, in-progress, succeeded, failed, unsupported, expired, rejected and unknown. Include a safe machine-readable reason and a correlation identifier. Do not turn unsupported into succeeded merely because a provider returned no error.

Threat-to-test record

Invariant: Tenant A cannot subscribe to tenant B's output stream.
Attempt: Use a valid A credential with B's stream ticket and session ID.
Expected: Rejection before subscribing to runtime output.
Evidence: API result, safe audit metadata and stream-access counter.
Variants: Expired ticket, changed project, revoked user, reconnect.
Limit: Does not prove runtime isolation or host security.

Appendix C. Review Answers and Failure Scenarios

These notes are discussion guides, not a claim that only one architecture is correct. A sound alternative identifies its assumptions and verifies its own invariants.

Why is an outbound connector still powerful?

Because a connection initiated inside the cluster can still carry commands from the SaaS to the cluster. Direction controls reachability, not the authority of messages after connection establishment. Constrain the command vocabulary, identity, references, local policy bounds and workload lifetime independently.

Why can RLS tests pass while isolation remains weak?

Tests may use the wrong database role or omit writes and association attacks. An application role capable of changing tenant context is not a cryptographic boundary against arbitrary SQL execution. Composite references, parameterized queries, scoped caches and authorization checks address different failures.

What happens after execution but before the result is saved?

The system may not know whether an effect happened. A durable execution identity at the recipient can allow reconciliation. Without that support, a retry may repeat the effect. Expose unknown outcome and resolve it through an explicit procedure instead of claiming transport deduplication solved the problem.

Does one approval authorize any retry?

No. Approval should bind the exact operation and inputs, with an expiry and policy revision. A separate downstream idempotency contract determines whether repeating the attempted effect is safe. A tool with no deduplication support cannot inherit exactly-once semantics from the approval database.

Does deny-all override every NetworkPolicy allow rule?

Not in the ordinary additive allow model of standard NetworkPolicy. Evaluate all policies selecting the workload and prevent unreviewed broad grants. Ensure the actual network implementation enforces the rules. A manifest by itself is not proof of packet behavior. See [S04](#).

Can a hash chain prove an audit log was not rewritten?

Only within its trust model. An attacker able to rewrite every record can also recompute an unanchored chain. An independently protected checkpoint makes such rewrites detectable relative to that checkpoint. Confidentiality, durable storage and access controls remain separate properties.

Can restoring an old database restore revoked access?

Yes, if restored authorization records are treated as current. A recovery plan must reconcile revocations, active epochs, approval consumption and external effects before reconnecting live workloads. An independent revocation record or controlled re-enrollment procedure may be required.

Where can customer data still reach the SaaS?

Output relay, file transfers, diagnostic bundles, tool arguments, telemetry and exports are all possible paths. A policy of not fetching secret values from a provider does not prevent a process from printing one. State default retention, processing locations and optional direct-data-plane modes honestly.

When should a feature be disabled?

When its mandatory enforcement capability is missing or cannot be verified. Offer an explicit development-only mode only with a different assurance label. Do not quietly replace gVisor with a default runtime or FQDN enforcement with an unrestricted route while retaining the same product claim.

Appendix D. Companion Files and Build Commands

Companion files

The GitHub-ready ZIP is the complete companion repository. The PDF and EPUB are reading editions, not executable software distributions. The assembled Markdown is generated from the same chapters. No public repository URL is assumed.

Path	Purpose
book/chapters/	The 28 source chapters
book/appendices/	Glossary, records, review notes and this guide
book/assets/	Original diagrams and cover artwork
prompts/00-*.md through 44-*.md	Bounded implementation assignments
prompts/TRACKER.csv	Initial implementation tracker
examples/go-core/	State, approval and quota teaching programs
examples/audit/	Hash-chain example and tests
examples/sql/	Disposable PostgreSQL isolation lab
examples/kubernetes/	Local-only restricted manifest renderer
labs/	Guided exercises with acceptance criteria
scripts/	Publication, verification and artifact checks
reports/VALIDATION.md	Executed checks and known limitations
.github/workflows/	Validation and Pages publication workflows
dist/	PDF, EPUB and checksums for this edition
AgentPlane-Book.md	Assembled Markdown reading edition

Read without building

Open the PDF in a PDF reader, import the EPUB into an EPUB-compatible reader, or start with `book/index.md` in a Markdown viewer. The chapter files work directly on GitHub. Diagrams also have their editable source alongside the rendered images.

Rebuild locally

The publication requires Python 3.11 or later and the pinned packages in the requirements files. Full export also requires Pandoc and system libraries used by WeasyPrint. See `BUILDING.md` for the tested tool versions and platform notes. The standard build does not provision a cluster or contact a cloud account.

```
python3 -m venv .venv
. .venv/bin/activate
python -m pip install -r requirements.txt
python scripts/check.py
python scripts/build.py --format all
```

Generated files are disposable outputs. Edit chapter sources and rebuild them. The website is written to `_site`, while reading editions are written to `dist` and the root assembled Markdown file. `make verify` runs the teaching checks; `make book` produces the complete publication.

What publication does not do

The package does not create a GitHub repository, enable Pages, push a commit, publish a release, deploy AgentPlane or certify runtime isolation. Follow `PUBLISHING.md` to perform the account-side steps deliberately. The initial implementation tracker remains not started because a book is not the application.

Primary Sources and Verification Notes

Documentation reviewed for this edition on **October 10, 2026**.

These are primary-source pointers, not copies of upstream documentation. AgentPlane service boundaries, APIs, policies and exercises are proposed designs unless the validation report identifies an executed teaching example. A source URL does not establish a tested compatibility matrix. Public documentation may move after this edition.

S01

Agent Sandbox documentation

<https://agent-sandbox.sigs.k8s.io/docs/>

Resource concepts and documentation entry point. Upstream capability is not an AgentPlane implementation claim.

S02

Agent Sandbox source repository

<https://github.com/kubernetes-sigs/agent-sandbox>

Inspect release-specific APIs, runtime server, router and extension resources before implementing adapters.

S03

Kubernetes multi-tenancy

<https://kubernetes.io/docs/concepts/security/multi-tenancy/>

Namespace isolation, shared infrastructure and tenancy tradeoffs.

S04

Kubernetes Network Policies

<https://kubernetes.io/docs/concepts/services-networking/network-policies/>

Additive allow rules, implementation dependencies and network-policy limitations.

S05

Kubernetes Pod Security Standards

<https://kubernetes.io/docs/concepts/security/pod-security-standards/>

Restricted pod configuration is not a complete hostile-code isolation system.

S06

gVisor security model

https://gvisor.dev/docs/architecture_guide/security/

Isolation design, assumptions and responsibilities outside the sandbox.

S07

PostgreSQL 18 row security

<https://www.postgresql.org/docs/18/ddl-rowsecurity.html>

RLS semantics, owner bypass, BYPASSRLS and policy behavior.

S08

PostgreSQL 18 explicit locking

<https://www.postgresql.org/docs/18/explicit-locking.html>

Row locks and transaction concurrency. The SQL lab requires independent execution.

S09

NATS JetStream concepts

<https://docs.nats.io/concepts/jetstream>

Persistence and delivery semantics; transport guarantees do not make arbitrary business effects exactly once.

S10

MCP security best practices

https://modelcontextprotocol.io/docs/2026-07-28/tutorials/security/security_best_practices

Audience separation, proxy risks, discovery SSRF and untrusted tool metadata. Protocol assumptions must be versioned.

S11

agentgateway documentation

<https://agentgateway.dev/docs/>

Verify the deployed release, Kubernetes resources and external-authorization capabilities rather than assuming them.

S12

Kubernetes volume snapshots

<https://kubernetes.io/docs/concepts/storage/volume-snapshots/>

CSI snapshot resources and support requirements; not a general memory checkpoint.

S13

Kubernetes Secrets

<https://kubernetes.io/docs/concepts/configuration/secret/>

Secret objects, projections and access boundaries.

S14

OWASP session management

https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

Session lifecycle, cookies and browser attack surfaces.

S15

OpenAI AGENTS.md guidance

<https://developers.openai.com/codex/guides/agents-md/>

Repository instructions for coding agents. This book does not prescribe unverified model IDs or reasoning settings.

S16

GitHub Pages custom workflows

<https://docs.github.com/en/pages/getting-started-with-github-pages/using-custom-workflows-with-github-pages>

Pages artifact deployment, permissions and environment requirements.

S17

Mistune usage guide

<https://mistune.lepture.com/en/latest/guide.html>

The small Markdown renderer used by this book build.

S18

CC BY-SA 4.0 legal code

<https://creativecommons.org/licenses/by-sa/4.0/legalcode.en>

License for original narrative text, diagrams and prompts. License scope is defined in LICENSE.md.

S19

MIT License

<https://opensource.org/license/mit>

License for original program code and configuration examples.

S20

OpenTelemetry sensitive-data handling

<https://opentelemetry.io/docs/security/handling-sensitive-data/>

Minimize, redact and govern telemetry rather than recording everything.

S21

Kubernetes Leases

<https://kubernetes.io/docs/concepts/architecture/leases/>

Coordination mechanism; resource-side fencing remains a separate design requirement.

S22

Kubernetes operator pattern

<https://kubernetes.io/docs/concepts/extend-kubernetes/operator/>

Reconciliation-based extensions and controllers.

S23

Go traversal-resistant file APIs

<https://go.dev/blog/osroot>

Root-relative operations and filesystem race considerations; API availability depends on Go version.

S24

GitHub Actions secure use

<https://docs.github.com/en/actions/reference/security/secure-use>

Least-privilege workflows, untrusted inputs and immutable action references.