CI/CD Deployment Pipeline Runner

Technical Specification

Document ID	PLAT-SPEC-RUNNER-001
Version	1.0.0
Status	Approved Baseline
Owner	Platform Engineering
Published	2 August 2026
Classification	Internal Engineering Standard

This document defines the normative architecture, interfaces, security controls, and service objectives for the organization's pipeline runner platform.

Document control

Field	Value
Purpose	Establish a buildable, testable, and auditable baseline for a multi-tenant CI/CD runner and deployment control plane.
Audience	Platform engineers, security engineers, service owners, SREs, compliance reviewers, and implementation partners.
Normative language	The terms **MUST**, **SHOULD**, and **MAY** are to be interpreted as requirement levels.
Source of truth	The version-controlled copy of this document and the associated approved architecture decision records.
Review cadence	At least annually and after a material change to trust boundaries, execution isolation, identity, or deployment strategy.

Approval record

Role	Accountability	Approval state	Evidence
Head of Platform	Product and technical ownership	Approved	Version-control review
Security Engineering	Threat model and control design	Approved	Security design review
Site Reliability Engineering	Operability and service objectives	Approved	Production readiness review
Developer Experience	Workflow usability and adoption	Approved	Pilot exit review

Change history

Version	Date	Author	Summary
1.0.0	2 Aug 2026	Platform Engineering	Approved baseline: architecture, contracts, security, SLOs, rollout, and acceptance criteria.

Contents

List of Tables

1 Executive summary

The pipeline runner is the controlled execution layer between source-control events and changes to runtime environments. It accepts validated pipeline definitions, turns them into an immutable execution plan, schedules isolated jobs onto eligible runner pools, collects logs and evidence, and mediates deployments through policy gates. The design separates the highly available control plane from short-lived, untrusted execution workers.

The baseline uses ephemeral runners, workload identity, least-privilege credentials, content-addressed artifacts, and signed provenance. Builds have no implicit access to deployment credentials. A deployment job receives scoped, short-lived authorization only after policy evaluation and, where required, human approval. All control-plane mutations and security-sensitive runner actions create append-only audit events.

Key architectural decisions. The control plane is the system of record; runners are disposable and never authoritative. Jobs are delivered using renewable leases rather than fire-and-forget messages. Every job runs in a fresh isolation boundary. Secrets are resolved just in time and are never persisted in pipeline plans, caches, artifacts, or runner images. Deployments refer to immutable artifact digests and use environment-specific policy gates.

1.1 Target outcomes

- A repository can run its first policy-compliant pipeline within 15 minutes of onboarding.
- A queued job begins within 60 seconds at the 95th percentile under planned load.
- No single runner compromise grants control-plane or cross-tenant access.
- Every production change is attributable to an actor, source revision, artifact digest, policy decision, and deployment result.
- Control-plane recovery preserves accepted jobs and prevents duplicate deployment effects.

1.2 Primary measures of success

Table 1: Program success measures

Measure	Definition	Target
Adoption	Active repositories using the managed runner platform	$\geq 80\%$ within two quarters
Change traceability	Production deployments with complete provenance	100%
Queue latency	Time from job readiness to lease acquisition	p95 < 60 s, p99 < 180 s
Platform availability	Successful control-plane API service	99.95% monthly
Isolation escapes	Confirmed cross-job or cross-tenant isolation failures	0

2 Scope and context

2.1 In scope

This specification covers pipeline ingestion and validation, directed-acyclic-graph (DAG) planning, job scheduling, runner registration and attestation, execution isolation, logs, artifacts, caches, secrets delivery, deployment environments, approvals, policy enforcement, observability, audit, API contracts, service objectives, capacity, disaster recovery, and production rollout.

2.2 Out of scope

- Source-control hosting, code review, and branch-protection implementation.
- Artifact repository internals, secret-vault internals, and cloud-provider control planes.
- Application-specific test frameworks and deployment strategies implemented by teams.
- General-purpose interactive development environments or arbitrary persistent workloads.
- End-user billing; resource usage is measured and exported for showback or chargeback.

2.3 Assumptions and constraints

- Source revisions are addressable by immutable commit identifiers.
- An enterprise identity provider supports OpenID Connect (OIDC), groups, and multi-factor authentication.
- A secret manager and artifact/object store are available through authenticated APIs.
- Production environments can validate federated workload identities and artifact signatures.
- Linux is the mandatory first execution platform; Windows and macOS are optional pool types.
- The platform is multi-tenant. Repository administrators are not infrastructure administrators.
- A regional network or cloud failure is credible and must not corrupt accepted pipeline state.

2.4 Personas and responsibilities

Table 2: Primary personas

Persona	Needs	Accountability
Contributor	Fast, reproducible feedback with understandable errors	Pipeline code and remediation of job failures
Service owner	Safe promotion, approvals, deployment visibility	Service policy, environments, and release decisions
Platform engineer	Scalable scheduling and standardized runner images	Control plane, base images, capacity, upgrades
Security engineer	Enforced isolation, identity, evidence, and response	Guardrails, detections, exceptions, investigations
SRE / operator	Clear SLOs, diagnostics, and recovery procedures	Availability, incidents, capacity, disaster recovery
Auditor	Complete, immutable evidence without platform-admin access	Independent control verification

3 Terminology and invariants

Pipeline A versioned declaration of triggers, jobs, dependencies, inputs, and policies.

Run One evaluated instance of a pipeline for a source revision and event.

Job The smallest independently scheduled node in the run DAG.

Step An ordered command or action within a job; steps share the job sandbox.

Runner An agent that claims a job lease and manages its execution sandbox.

Runner pool A policy and capacity boundary defined by platform, region, trust tier, and capabilities.

Lease Time-bounded ownership of a job attempt by one runner.

Attempt One execution of a job. Retries create new attempts with distinct identities.

Artifact Immutable output addressed by cryptographic digest and retained independently of a runner.

Cache A performance optimization whose content is untrusted and may be deleted without affecting correctness.

Environment A protected deployment target with policy, concurrency, identity, and approval rules.

Provenance Signed evidence connecting source, plan, builder, materials, commands, and outputs.

The following invariants are non-negotiable:

1. A job is never considered successful until required logs, result metadata, and output digests are durably committed.
2. At most one active lease exists for a job attempt. A stale attempt cannot finalize after its fencing token expires.
3. A deployment consumes an immutable artifact digest, never a mutable tag alone.
4. A runner has no durable control-plane credential. Its identity is short-lived and bound to an attested instance.
5. A cache hit cannot grant more trust than a cache miss; security decisions never depend on cache contents.
6. Redaction is defense in depth. A value's secrecy cannot rely solely on log filtering.
7. Authorization is evaluated by the control plane at the time of each protected action.

4 System architecture

4.1 Logical architecture

Figure 1 shows the principal components and trust boundaries. The public ingress path ends at the control-plane API. Job payloads flow to runners through an authenticated broker or long-poll endpoint; runners do not accept inbound connections from repositories.

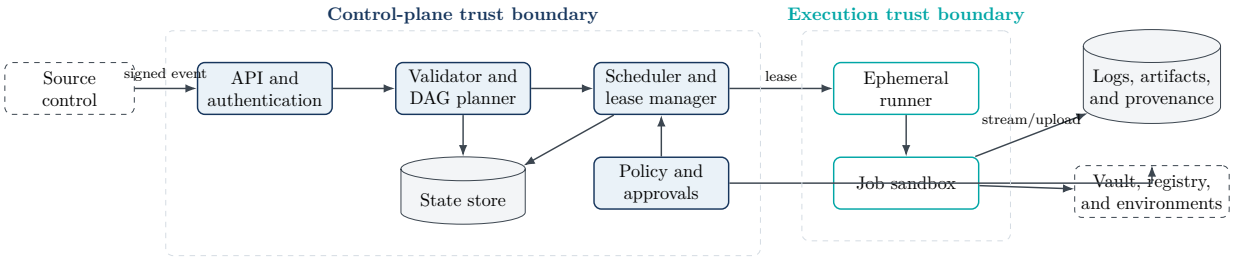

Figure 1: Logical architecture and major trust boundaries

4.2 Component responsibilities

Table 3: Component responsibility model

Component	Responsibilities	Must not do
API gateway and identity	Authenticate users and services, verify webhook signatures, authorize commands, enforce quotas, issue idempotency keys	Execute pipeline commands or expose internal service credentials
Pipeline service	Resolve revision, parse schema, evaluate expressions, expand matrices, construct immutable DAG and plan digest	Fetch or interpolate secret values during planning
Scheduler	Match ready jobs to eligible capacity, apply fairness, issue leases and fencing tokens, process heartbeats and retries	Trust runner-reported authorization or bypass policy gates
Policy and approval service	Evaluate organization, repository, environment, and risk policies; record signed decisions	Mutate source, artifacts, or deployment output
Runner manager	Register and attest workers, manage pool capacity and image rollout, quarantine unhealthy instances	Persist repository credentials or reuse dirty sandboxes
Runner agent	Claim one or more permitted leases, create sandbox, stream logs, upload outputs, report result, erase workspace	Decide protected-environment authorization
Artifact service	Issue scoped upload/download grants, verify digests, retain metadata and provenance	Execute uploaded content or accept digest mismatch
Audit service	Append, integrity-protect, export, and retain security and administrative events	Allow ordinary administrators to rewrite or delete evidence

4.3 Deployment topology

The control plane **MUST** run across at least three failure domains in the primary region. Stateless services use an active/active topology. The state store uses a quorum-backed high-availability

configuration and continuous point-in-time recovery. Object storage uses cross-zone durability. A warm standby control plane in the recovery region receives replicated configuration, audit data, and backups but does not lease jobs until promoted.

Runner pools are independent scaling units. Pools are segmented by operating system, architecture, region, network zone, workload trust tier, and specialized capability (for example GPU or hardware signing). Internet-facing untrusted builds and trusted production deployments **MUST** not share a pool, node, service account, or mutable image.

4.4 Trust boundaries and data flows

1. **Source boundary:** repository content and pull-request code are untrusted input.
2. **Control-plane boundary:** only authenticated APIs and signed service-to-service traffic enter.
3. **Execution boundary:** a sandbox receives an immutable plan fragment and scoped grants.
4. **Protected-target boundary:** environment access requires workload identity, policy approval, and artifact verification.
5. **Evidence boundary:** audit and provenance records are append-only and independently retained.

5 Pipeline contract

5.1 Definition discovery and versioning

The default definition path is `.pipeline/pipeline.yaml`. Repositories may select another path through an administrator-controlled setting. The pipeline file **MUST** declare an explicit schema version. Unknown top-level keys are errors unless the schema marks them as extension points. The control plane stores the original file digest, the fully resolved plan, the source revision, and the planner version.

Included templates are resolved by immutable revision or signed release digest. A floating template reference may be used only if the control plane resolves and records its immutable digest before planning. Recursive includes, path traversal, and templates outside the authorized organization are rejected.

5.2 Reference pipeline

Listing 1: Reference pipeline definition

```
version: "v1"
name: service-release

on:
  pull_request: {}
  push:
    branches: [main]
  manual:
    inputs:
      release_ref: {type: string, required: false}

permissions:
  repository: read
  artifacts: write
  id-token: write
```

```
16
17 concurrency:
18   group: "service-${{ ref.name }}"
19   cancel_in_progress: true
20
21 jobs:
22   test:
23     pool: linux-untrusted
24     timeout: 20m
25     matrix:
26       runtime: ["20", "22"]
27     steps:
28       - uses: org/checkout@sha256:4f57...a012
29       - run: npm ci --ignore-scripts
30       - run: npm test
31
32   package:
33     needs: [test]
34     pool: linux-build
35     steps:
36       - uses: org/checkout@sha256:4f57...a012
37       - run: ./scripts/build-image.sh
38       - upload:
39           name: service-image
40           path: out/image.oci
41
42   deploy_staging:
43     needs: [package]
44     environment: staging
45     artifact: "${{ jobs.package.artifacts.service-image.digest }}"
46     steps:
47       - uses: org/deploy@sha256:8c11...9be2
48
49   deploy_production:
50     needs: [deploy_staging]
51     if: "${{ event.type == 'manual' }}"
52     environment: production
53     artifact: "${{ jobs.package.artifacts.service-image.digest }}"
54     steps:
55       - uses: org/deploy@sha256:8c11...9be2
```

5.3 Expression safety

Expressions are evaluated by a deterministic, side-effect-free interpreter. They have no filesystem, network, clock, random-number, reflection, or dynamic-code access. Values carry a type and sensitivity label. Secret-derived values are prohibited in job names, cache keys, concurrency groups, conditions visible before authorization, and other metadata. String interpolation into a shell command triggers a warning and may be denied by policy; values **SHOULD** be passed using structured environment or input fields.

5.4 Planning rules

The planner **MUST**:

1. validate schema, event eligibility, referenced actions, pool names, permissions, and limits;
2. expand matrices to a bounded set of jobs (default maximum: 256 jobs per run);

3. reject cycles and references to nonexistent jobs or outputs;
4. resolve actions and templates to immutable digests;
5. compute a canonical plan digest over all execution-relevant fields;
6. persist the plan transactionally before any job becomes schedulable; and
7. return structured diagnostics containing code, location, message, and remediation.

5.5 Limits

Table 4: Default pipeline limits

Limit	Default	Enforcement
Definition size	1 MiB	Reject during ingestion
Jobs after expansion	256 per run	Reject during planning
Steps per job	100	Reject during planning
Job wall-clock time	60 minutes	Kill sandbox; configurable to 6 hours by policy
Log output	100 MiB per job	Truncate with explicit terminal marker; retain result metadata
Artifacts	20 GiB per job	Deny additional uploads
Concurrent runs	20 per repository	Queue with fair scheduling
Run retention	90 days	Tier or delete according to policy and legal hold

6 Job lifecycle and scheduling

6.1 State model

The persisted state machine is authoritative. State transitions use compare-and-swap semantics and append an event to the run history. Figure 2 omits administrative states for readability.

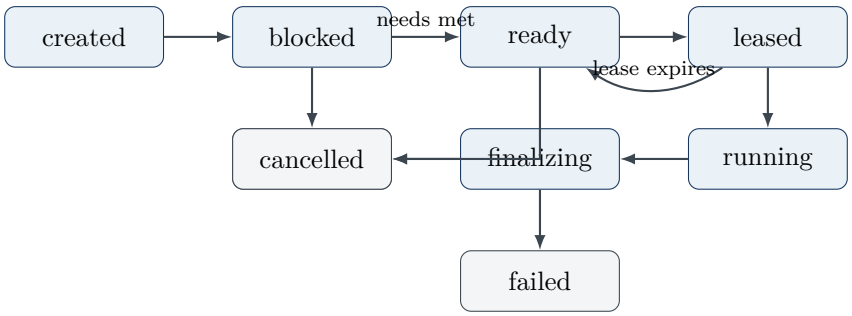

Figure 2: Simplified job state machine

A job may also end as `timed_out`, `skipped`, or `infrastructure_failed`. An infrastructure failure is distinct from a user-command failure and is eligible for automatic retry. Terminal states are immutable except through a documented administrator correction process that appends, rather than rewrites, history.

6.2 Lease protocol

1. An attested runner opens an authenticated long-poll request containing pool, capabilities, current concurrency, image digest, and health status.
2. The scheduler selects a ready job and atomically creates an attempt with a unique fencing token and a default 60-second lease.
3. The runner acknowledges within 10 seconds and renews every 20 seconds. Renewal proves possession of runner identity and the current fencing token.
4. Missing two renewal windows makes the lease suspect; expiry returns the job to ready or creates a retry according to policy.
5. Finalization is accepted only for the current attempt and fencing token. Late results are retained as diagnostic events but cannot change job state.

External deployment effects cannot generally be rolled back by fencing alone. A deployment attempt therefore carries an idempotency key derived from environment, run, job, attempt, and artifact digest. Deployment adapters **MUST** enforce that key or expose a reconciliation operation.

6.3 Scheduling policy

Eligibility is a hard filter over pool, platform, architecture, labels, trust tier, region, network zone, image policy, and resource request. Selection among eligible jobs uses weighted fair queuing across organizations and repositories, with aging to prevent starvation. Protected deployment jobs may use a reserved capacity class but cannot preempt a running job.

For tenant i, the scheduler computes normalized dominant resource share

$$S_i = \max_{r \in R} \left(\frac{u_{i,r}}{w_i C_r} \right),$$

where $u_{i,r}$ is current use of resource r, C_r is allocatable capacity, and w_i is the tenant weight. Among ready queues, the scheduler prefers the lowest S_i after applying priority and age bounds. Policy limits prevent a tenant from raising priority above its authorized class.

6.4 Concurrency and cancellation

A concurrency group serializes runs or selected jobs. Acquisition and release are transactional. If `cancel_in_progress` is true, the prior run receives a cancellation request and a bounded grace period; the successor does not enter a protected environment until the previous deployment has stopped or reconciled. Cancellation is cooperative for 10 seconds and then forceful. Cleanup steps run with separate, tightly scoped credentials and a maximum duration.

7 Runner and execution environment

7.1 Runner registration and attestation

Runner instances bootstrap with a one-time, pool-scoped registration token delivered through instance identity. Registration proves pool membership, image digest, platform, agent version, and, where available, hardware or virtual-machine attestation. The control plane issues a mutually authenticated TLS certificate valid for no more than 30 minutes. Certificates are automatically rotated and immediately revocable.

Static runner tokens are prohibited in machine images, startup scripts, CI variables, and configuration management. A runner whose image, clock, patch level, attestation, or agent version violates pool policy is quarantined and receives no leases.

7.2 Sandbox requirements

Every attempt **MUST** execute in a fresh virtual machine, micro-VM, or equivalently isolated container meeting the pool's threat model. Untrusted pull-request code uses a micro-VM or dedicated virtual machine by default. A standard container is permitted only for trusted code when the kernel-sharing risk is explicitly accepted.

The sandbox **MUST** provide:

- a read-only base image identified by digest and a disposable writable layer;
- an unprivileged user, dropped capabilities, syscall filtering, and resource limits;
- a job-specific network policy with deny-by-default egress for protected pools;
- per-job cgroups or equivalent CPU, memory, process, and I/O enforcement;
- a clean workspace, isolated temporary directories, and no host socket mounts;
- monotonic time and synchronized wall clock within 30 seconds;
- forced teardown and cryptographic erasure of ephemeral storage after completion.

Privileged execution, host networking, nested virtualization, device access, and mounting the container runtime socket are denied by default. Exceptions require a dedicated pool, documented threat model, named owner, expiry date, and compensating controls.

7.3 Workspace and checkout

The source checkout action uses a job-scoped, read-only credential unless the plan explicitly requests an authorized write permission. It checks out the exact revision, disables persistent credential helpers, validates submodule and large-file-storage destinations, and removes authentication material immediately after transfer. Pull requests from forks never receive repository write tokens or protected secrets.

7.4 Network policy

Pool policy defines approved destinations by service identity or domain plus port; raw address allowlists are used only where identity-aware controls are unavailable. DNS is logged and protected against rebinding. Requests to instance metadata, link-local networks, the control-plane administrative plane, and other runner sandboxes are blocked. Artifact, log, package-mirror, vault, and OIDC access traverse authenticated private endpoints where feasible.

7.5 Runner images and updates

Base images are generated by code, scanned, signed, and promoted through test, canary, and stable channels. Each job records the image digest and software inventory. Critical security patches enter canary within 24 hours and stable within 72 hours, subject to emergency rollback. The control plane can deny images by digest and drain affected instances. Image updates never modify an executing sandbox in place.

8 Artifacts, caches, logs, and provenance

8.1 Artifacts

Artifact upload follows a two-phase protocol: request a scoped upload grant, upload parts directly to object storage, then finalize with manifest, size, media type, and SHA-256 digest. The artifact service independently verifies digest and size before marking the artifact available. Partial uploads expire automatically.

Artifacts are immutable. Names are friendly references to versions; execution and deployment contracts use digests. Encryption is mandatory in transit and at rest. Authorization is checked at grant issuance and download. Retention is inherited from organization, repository, environment, and legal-hold policy, taking the longest applicable duration.

8.2 Caches

Caches are keyed by tenant, repository, trust tier, platform, declared key, and an internal schema version. Pull-request jobs from forks may read only caches explicitly published as non-sensitive and may not overwrite trusted-branch cache entries. Cache archives reject absolute paths, traversal, special devices, set-user-ID files, and unsafe links. Extraction occurs as an unprivileged user within the sandbox.

Cache entries are scanned opportunistically but always treated as untrusted input. Integrity failures cause eviction and a cache miss, not job success. Eviction follows quota and least-recently-used policy; correctness must not depend on retention.

8.3 Logs

The runner streams ordered log frames containing attempt identifier, step identifier, sequence number, monotonic timestamp, stream, and bytes. The service acknowledges durable offsets so a reconnect can resume without gaps. Duplicate sequence numbers are idempotent. The UI clearly marks missing, delayed, truncated, and redacted segments.

Secret values and registered variants are masked at the runner and again on ingestion. The platform also detects common credential formats. Binary output is encoded safely. ANSI control sequences are allowlisted to prevent terminal injection. Access to raw logs follows repository permissions; download is audited.

8.4 Provenance and software bill of materials

Successful package jobs emit signed provenance containing, at minimum:

- source repository and immutable revision;
- pipeline and canonical plan digests;
- action, template, runner-image, and dependency material digests;
- builder identity, pool, attempt, timestamps, parameters, and output digests;
- whether the build was isolated, hermetic, reproducible, and policy-compliant; and
- links to the software bill of materials (SBOM), test evidence, and scan results.

Signing uses a short-lived workload certificate rooted in the platform trust domain. Production policy **MUST** verify provenance signature, source eligibility, builder pool, required evidence, and artifact digest before deployment.

9 Secrets, identity, and authorization

9.1 Identity model

Human access uses enterprise SSO and multi-factor authentication. Service-to-service access uses mutually authenticated TLS identities. A job obtains a unique OIDC workload identity whose claims include organization, repository, source revision, run, job, attempt, environment, plan digest, runner pool, and token audience. Default token lifetime is five minutes and cannot exceed the remaining job lease.

Cloud and environment access uses federation from this identity; long-lived cloud keys are prohibited. Token exchange policy matches exact issuer, audience, repository, branch or tag rule, environment, and protected-pool claim.

9.2 Secret delivery

The pipeline plan carries secret references, never values. At step start the runner requests an access grant from the control plane. After a fresh authorization decision, the runner exchanges its workload identity directly with the secret manager. Secret values are delivered in memory or a job-local in-memory filesystem and are removed after the step. Values are not included in environment dumps, command arguments, process titles, crash reports, caches, or artifacts.

Secret access from untrusted fork code is denied. Environment secrets are available only after environment policy succeeds. Rotation does not require rebuilding runner images. Every read creates an audit event with secret identifier, not value.

9.3 Authorization model

Access control combines role-based permissions with attributes such as organization, repository, environment, source ref, event type, actor, runner pool, and risk score. Deny rules override allow rules. Policy evaluation returns a decision identifier, matched policy version, reasons, obligations, and expiry.

Table 5: Minimum authorization roles

Role	Permitted actions	Explicit exclusions
Viewer	View authorized runs, metadata, and redacted logs	Retry, cancel, approve, administer
Developer	Trigger, cancel own run, retry non-protected jobs	Change policy, approve own deployment
Maintainer	Manage repository pipelines and ordinary schedules	Administer platform or bypass environment policy
Environment approver	Approve or reject eligible deployments	Modify the pending artifact or approve own request if separation is required
Organization admin	Manage repositories, quotas, pools allowed to the organization	Access platform break-glass or rewrite audit
Platform operator	Operate capacity and services	Read secrets or approve application deployments by default
Auditor	Read retained audit, policy decisions, and provenance	Mutate any production resource

9.4 Approvals and separation of duties

Approval rules specify eligible groups, minimum count, self-approval policy, expiry, and whether a new commit, artifact, plan, policy, or vulnerability result invalidates approval. The approval record binds the exact environment, plan digest, artifact digest, actor, decision, and timestamp. Production defaults to two eligible approvers, no self-approval, and 24-hour expiry. Emergency access is time-bound, reason-coded, independently alerted, and retrospectively reviewed.

10 Deployment orchestration

10.1 Environment contract

An environment defines target identity, allowed source refs, allowed artifact classes, required evidence, runner pool, network zone, reviewers, maintenance windows, concurrency, timeout, rollback policy, and credential federation. Environment configuration is versioned and changes are audited.

Before leasing a deployment job, the control plane **MUST** confirm:

1. upstream jobs succeeded and the run has not been superseded or cancelled;
2. artifact digest exists, is immutable, and has valid provenance and required SBOM;
3. source revision and event satisfy branch, tag, and repository rules;
4. vulnerability, license, test, and change-management evidence satisfy current policy;
5. approvals are valid and bound to the current digests and policy version;
6. environment concurrency and change window permit entry; and
7. an eligible protected runner and workload identity can be issued.

10.2 Promotion

Promotion reuses the same artifact digest across environments. Rebuilding between staging and production is prohibited. Environment-specific configuration is injected at deployment or runtime and is separately versioned. A promotion record links the source deployment, target environment, evidence snapshot, approvals, and outcome.

10.3 Deployment strategies

Adapters may implement rolling, blue/green, canary, or custom strategies. Each adapter exposes `prepare`, `apply`, `observe`, `rollback`, and `reconcile` operations with idempotency keys. Health gates use bounded observation windows and explicit success criteria. A timeout is an unknown outcome until reconciliation proves whether the target changed.

10.4 Rollback

Rollback is a new, audited deployment of a previously approved artifact or an adapter-specific compensating action. It does not erase the failed deployment record. Automatic rollback requires pre-authorized policy, a known-good target digest, and a health signal whose false-positive risk is understood. Database changes and other irreversible effects require an application-owned recovery plan referenced by the job.

11 Functional requirements

Table 6 is normative. “Verification” identifies the primary evidence; it does not limit additional testing.

Table 6: Functional requirements

ID	Requirement	Verification
FR-001	The system **MUST** authenticate every user, service, webhook, and runner before accepting work.	Auth integration and negative tests
FR-002	The system **MUST** verify event signatures, timestamp freshness, delivery identity, and replay status.	Webhook conformance tests
FR-003	The planner **MUST** produce and persist a deterministic DAG and canonical digest before scheduling.	Golden plan tests
FR-004	The planner **MUST** reject cycles, unresolved references, excessive expansion, and unauthorized permissions.	Schema and fuzz tests
FR-005	The scheduler **MUST** enforce pool eligibility, quotas, fairness, priority bounds, and environment gates.	Simulation and policy tests
FR-006	Job assignment **MUST** use expiring leases and fencing tokens that reject stale completion.	Fault-injection test
FR-007	Every attempt **MUST** execute in a fresh sandbox and leave no reusable writable state.	Image and teardown test
FR-008	Runner and job credentials **MUST** be short-lived, scoped, audience-bound, and revocable.	Token-claims inspection
FR-009	Secrets **MUST** be resolved just in time and excluded from plans, logs, caches, artifacts, and images.	Secret canary tests
FR-010	Logs **MUST** be ordered, resumable, durably acknowledged, redacted, and explicitly marked if incomplete.	Disconnect and redaction tests
FR-011	Artifact finalization **MUST** independently verify size and cryptographic digest.	Corruption tests
FR-012	Cache namespaces **MUST** prevent writes from a lower-trust context into a higher-trust context.	Fork-boundary tests
FR-013	The system **MUST** emit signed provenance tying outputs to source, plan, materials, builder, and attempt.	Signature verification
FR-014	A production deployment **MUST** consume an immutable digest and pass current environment policy.	End-to-end release test
FR-015	Approval **MUST** bind actor, environment, artifact, plan, policy version, decision, and expiry.	Approval invalidation tests
FR-016	Cancellation **MUST** stop future jobs, revoke grants, terminate active sandboxes, and record final disposition.	Cancellation race tests
FR-017	Retries **MUST** create a new attempt identity while preserving job and run history.	Retry audit test

ID	Requirement	Verification
FR-018	All administrative, authorization, secret, approval, and deployment actions **MUST** generate immutable audit events.	Audit completeness test
FR-019	Public mutation APIs **MUST** support idempotency and return stable structured error codes.	API conformance suite
FR-020	Operators **MUST** be able to drain, quarantine, rotate, and revoke runners and images without database edits.	Operational exercise
FR-021	The system **MUST** enforce configurable resource, concurrency, retention, and cost quotas by tenant.	Load and quota tests
FR-022	Protected deployment adapters **MUST** support status reconciliation after timeout or control-plane interruption.	Recovery test
FR-023	The system **MUST** preserve accepted run state across an availability-zone failure.	Zone-failure exercise
FR-024	The disaster-recovery process **MUST** prevent both regions from leasing jobs simultaneously.	Regional failover exercise

12 API and event contracts

12.1 General conventions

The external API is versioned under `/v1`. Requests and responses use UTF-8 JSON. Times use RFC 3339 UTC with fractional seconds; identifiers are opaque, globally unique, and never recycled. Mutating requests accept `Idempotency-Key`; the key is scoped to caller and operation and retained for at least 24 hours. List endpoints use stable cursor pagination. Unknown response fields must be ignored by clients.

Errors have HTTP status, stable machine code, human-readable message, request ID, retryability, and optional field diagnostics. Authentication failures reveal no resource existence. Rate-limit responses provide a retry hint.

12.2 Core endpoints

Table 7: Core API surface

Method	Path	Purpose
POST	`/v1/events/source`	Accept a signed source-control event and deduplicate delivery
POST	`/v1/runs`	Manually create a run for an authorized revision and inputs
GET	`/v1/runs/{run_id}`	Return run, plan, job summary, policy, and evidence links
POST	`/v1/runs/{run_id}:cancel`	Request idempotent cancellation
POST	`/v1/jobs/{job_id}:retry`	Create a policy-authorized retry attempt
GET	`/v1/jobs/{job_id}/logs`	Stream or page ordered, authorized logs

Method	Path	Purpose
POST	`/v1/approvals`	Approve or reject a digest-bound environment request
POST	`/v1/runners:register`	Register and attest an ephemeral runner
POST	`/v1/leases:claim`	Long-poll for an eligible job lease
POST	`/v1/leases/{lease_id}:renew`	Renew lease and report health/progress
POST	`/v1/attempts/{attempt_id}:finalize`	Atomically commit result and output manifest
POST	`/v1/artifacts:beginUpload`	Issue a scoped multipart upload grant
POST	`/v1/artifacts/{artifact_id}:finalize`	Verify and publish immutable artifact

12.3 Example lease response

Listing 2: Abbreviated job lease

```
{
  "lease_id": "les_01J...",
  "attempt_id": "att_01J...",
  "fencing_token": 18442,
  "expires_at": "2026-08-02T10:15:30.000Z",
  "plan_digest": "sha256:5d9f...c821",
  "runner_image": "registry/runner@sha256:c021...7aa9",
  "sandbox": {
    "cpu_millis": 2000,
    "memory_bytes": 4294967296,
    "timeout_seconds": 1200,
    "network_policy_id": "netpol_build_v7"
  },
  "steps": [
    {"kind": "action", "digest": "sha256:4f57...a012"},
    {"kind": "command", "argv": ["npm", "test"]}
  ],
  "grants": {
    "log_upload": "grant_01J...",
    "artifact_upload": "grant_01K..."
  }
}
```

12.4 Event envelope

All emitted events use an envelope containing `event_id`, `event_type`, `schema_version`, `occurred_at`, `recorded_at`, `tenant_id`, `subject`, `actor`, `correlation_id`, `causation_id`, and `data`. Delivery is at least once; consumers deduplicate by event identifier. Events are partitioned by run identifier where ordering matters. Personally identifiable data and secret material are excluded from payloads.

13 Data model and consistency

13.1 Core entities

Table 8: Core persisted entities

Entity	Key attributes	Retention / consistency
Pipeline run	Tenant, repository, event, revision, plan digest, state, actor	Strong state transitions; 90 days default
Job	Run, dependencies, pool, policy, state, outputs	Strong state transitions; follows run
Attempt / lease	Job, runner, fencing token, expiry, result, reason	Strong lease ownership; follows run
Artifact	Tenant, digest, size, media type, provenance, retention	Immutable; policy-driven retention
Cache entry	Namespace, key, version, digest, size, last access	Eventual index; evictable
Environment request	Environment, artifact, plan, evidence, policy, state	Strong; at least production retention
Approval	Request, actor, decision, bound digests, expiry	Append-only; seven years for production default
Audit event	Actor, action, resource, outcome, context, integrity link	Append-only/WORM; seven years default

13.2 Transaction boundaries

Creation of a run, its plan, and initial jobs is one logical transaction. Job state transition and outbox event creation occur in the same database transaction. Lease claim atomically checks readiness, eligibility summary, attempt count, and absence of an active lease. Artifact metadata becomes available only after object verification. Approval mutation and environment-request transition are atomic.

An outbox publisher provides at-least-once event delivery. Consumers are idempotent and store their last applied event identifier or version. Read models may be eventually consistent; authorization, lease ownership, approvals, and deployment gates always use the authoritative store.

13.3 Data protection

Sensitive fields are classified and encrypted using envelope encryption with managed keys. Tenant identifiers are present in every logical key and query predicate. Database roles separate application mutation, read models, audit export, and break-glass access. Production access is just in time, approved, recorded, and limited to named cases. Backups are encrypted, integrity-tested, and subject to the same retention and access policy as source data.

14 Security architecture

14.1 Threat model

The design assumes an attacker can control repository content, dependency responses, build scripts, test output, and pull-request metadata. It also considers malicious insiders, stolen user sessions, compromised runner instances, dependency confusion, cache poisoning, artifact substitution, webhook replay, log injection, resource exhaustion, and control-plane service compromise.

Table 9: Threats and primary mitigations

Threat	Security objective	Primary mitigations
Untrusted code escapes job	Prevent host and cross-job compromise	Ephemeral micro-VM, unprivileged execution, syscall and network policy, no host sockets, rapid teardown
Credential theft	Limit value and lifetime of stolen material	Just-in-time workload identity, narrow audience, short TTL, memory delivery, egress controls
Poisoned dependency/action	Preserve input integrity	Digest pinning, trusted mirrors, signature checks, allowlists, provenance and SBOM
Cache poisoning	Prevent trust-boundary escalation	Trust-tier namespaces, safe extraction, immutable manifests, cache-as-untrusted invariant
Artifact substitution	Deploy exactly reviewed output	Digest addressing, independent verification, signature and provenance checks, approval binding
Webhook replay/spoofing	Accept authentic events once	Signature, timestamp window, delivery-ID deduplication, source allowlist
Stale runner result	Preserve state correctness	Renewable lease, monotonic fencing token, compare-and-swap finalization
Log or UI injection	Protect operators and evidence	Control-sequence allowlist, output encoding, content security policy, immutable raw frames
Denial of service	Preserve fair availability	Quotas, bounded expansion, weighted fairness, autoscaling, rate limiting, circuit breakers
Privileged insider	Make misuse difficult and detectable	Separation of duties, just-in-time admin, dual approval, immutable audit, session recording
Regional split brain	Prevent double execution/deployment	Single-writer promotion lease, fenced region epoch, runner certificate revocation

14.2 Control requirements

- All traffic **MUST** use TLS 1.2 or later; TLS 1.3 is preferred. Internal service traffic uses mutual authentication.
- Cryptographic algorithms and key sizes follow the organization's approved cryptographic standard.
- Dependencies and container images are continuously scanned; critical control-plane findings block release.
- Administrative APIs are isolated from public ingress and require phishing-resistant multi-factor

authentication.

- Security policy is version controlled, peer reviewed, tested, and deployed through the same staged process as code.
- Exceptions have owner, scope, rationale, compensating controls, approval, and automatic expiry.
- An external penetration test occurs before general availability and after a material trust-boundary change.

14.3 Audit integrity

Audit events include actor, authentication context, action, resource, request ID, source address, policy decision, before/after metadata where safe, result, and timestamp. Events are chained or batch-signed and exported within five minutes to an independently administered write-once store and security analytics platform. Clock health is monitored. Missing sequence ranges or signature failures create a high-severity alert.

15 Reliability, performance, and capacity

15.1 Service objectives

Table 10: Service-level objectives

Indicator	Objective	Measurement
Control-plane API availability	99.95% monthly	Valid non-5xx responses, excluding approved maintenance
Event acceptance latency	p99 $<$ 2 s	Verified webhook receipt to durable run request
Planning latency	p95 $<$ 5 s	Accepted request to persisted plan for definitions within limits
Ready-to-lease latency	p95 $<$ 60 s; p99 $<$ 180 s	Ready timestamp to successful lease, by pool and priority
Log visibility latency	p95 $<$ 3 s	Runner frame creation to authorized UI/API visibility
Cancellation propagation	p95 $<$ 10 s	Accepted cancel to runner acknowledgement
Artifact durability	99.999999999% annual target	Backing store durability plus verification
Audit export freshness	p99 $<$ 5 min	Control-plane commit to independent store

The monthly API error budget at 99.95% is approximately 21.9 minutes in a 30.4-day month. If burn rate exceeds 14.4 times budget over one hour or 6 times budget over six hours, the on-call team is paged. Exhausting the budget freezes non-remediation changes until the service owner approves recovery.

15.2 Recovery objectives

The primary control plane has a recovery point objective (RPO) of five minutes and a recovery time objective (RTO) of 60 minutes for regional disaster. Within a region, availability-zone

failure has an RPO of zero for accepted state and an RTO of 15 minutes. Artifact durability and replication are governed by the backing store, with metadata reconciliation included in recovery exercises.

Regional promotion requires independent confirmation that the primary writer is fenced. The recovery controller increments a region epoch; schedulers and runners reject leases from older epochs. In-flight deployments enter `reconciliation_required` until their target state is observed.

15.3 Capacity model

For pool p, the minimum warm concurrency is estimated as

$$N_p = \left\lceil \frac{\lambda_p \bar{t}_p}{\rho_p} \right\rceil + B_p,$$

where λ_p is peak admitted jobs per second, $\bar{t}_p$ is mean job duration, ρ_p is target utilization (no more than 0.70 for latency-sensitive pools), and B_p is burst and failure reserve. Capacity planning uses p90 job duration and event burst profiles in addition to the mean. Protected deployment pools retain at least two healthy instances per active region and failure domain policy.

Autoscaling observes ready queue depth, oldest-ready age, lease rate, startup latency, resource fit failures, and external quota. Scale-up is predictive where scheduled workloads are known. Scale-down drains instances, issues no new leases, and waits for completion or the configured maximum drain time.

15.4 Backpressure and degradation

Ingress uses per-tenant rate limits and a durable admission queue. When downstream systems are degraded, the control plane preserves accepted state and stops leasing jobs that cannot complete safely. Noncritical read models, analytics, cache uploads, and live log fan-out may degrade before scheduling or audit. Protected deployments fail closed when policy, identity, provenance, or audit dependencies are unavailable.

16 Observability and operations

16.1 Telemetry

Every service emits structured logs, metrics, and distributed traces using shared request, run, job, attempt, and tenant correlation identifiers. Telemetry excludes secret values and raw tokens. High-cardinality identifiers remain in logs and traces, not unbounded metric labels.

Required metrics include:

- event acceptance, planning duration, planning failures by stable code;
- ready queue depth and age by pool, priority, trust tier, and tenant class;
- lease claim, renewal, expiry, stale finalization, and infrastructure retry rates;
- runner boot, registration, attestation, quarantine, utilization, and teardown time;
- log ingest lag, redaction counts, dropped frames, and artifact verification failures;
- policy decision latency and deny reason, approval age, deployment duration and reconciliation;
- database saturation, transaction conflicts, replication lag, outbox lag, and object-store errors;
- estimated CPU, memory, storage, network, and cost by tenant and repository.

16.2 Health and readiness

Liveness indicates that a process can make local progress. Readiness verifies required dependencies and whether the instance may receive traffic. Scheduler readiness also requires a valid regional writer epoch. A runner advertises detailed health to the control plane but exposes no unauthenticated network endpoint. Synthetic probes create and execute a minimal pipeline in each critical pool at least every five minutes.

16.3 Alerts

Alerts are actionable and mapped to owned runbooks. Page-worthy conditions include SLO burn, inability to accept events, rapidly aging ready queues, widespread lease expiry, failure to export audit, identity issuance failure, deployment reconciliation backlog, regional fencing ambiguity, and suspected sandbox escape. Capacity forecasts, quota pressure, cache degradation, and noncritical read lag create tickets or warnings.

16.4 Required runbooks

Table 11: Operational runbook set

Scenario	Immediate action	Exit condition
Queue latency breach	Identify constrained pools, verify scheduler and quota, add or shift safe capacity	Oldest-ready age within objective and backlog declining
Lease expiry spike	Check network, runner health, clock, broker, and control-store latency; pause unsafe retries	Renewal success normal; duplicates reconciled
Suspected runner compromise	Quarantine pool, revoke runner identities and image digest, preserve evidence, rotate exposed grants	Incident commander and security approve rebuilt pool
Audit export failure	Preserve local outbox, restrict administrative changes, restore exporter or independent store	Backlog verified complete and freshness restored
Artifact integrity failure	Block digest, stop dependent deployments, compare storage and provenance, notify owners	Scope known, corrupted objects replaced or invalidated
Regional failover	Fence primary, verify quorum, promote epoch, rotate endpoints, reconcile in-flight work	Single writer proven; SLO probes pass
Deployment unknown	Freeze same-environment changes, query adapter and target, reconcile idempotency key	Target state and authoritative result recorded

17 Testing and verification strategy

17.1 Test layers

1. **Unit and property tests:** parser, expression evaluator, DAG, policy inputs, state machines, canonicalization, lease fencing, and safe archive extraction.
2. **Contract tests:** public API schemas, runner protocol compatibility, event evolution, artifact grants, OIDC claims, and deployment adapter behavior.
3. **Integration tests:** source event through logs, artifacts, provenance, approval, deployment, cancellation, retry, and retention.
4. **Security tests:** tenant isolation, sandbox escape attempts, token replay, SSRF, webhook replay, poisoned caches, malicious archives, log injection, and secret canaries.
5. **Resilience tests:** process crash, packet loss, duplicate delivery, stale lease, database failover, object-store errors, zone loss, and regional promotion.
6. **Performance tests:** burst admission, large DAG planning, scheduler fairness, runner churn, log fan-out, artifact throughput, and noisy-neighbor containment.

17.2 Compatibility policy

The control plane supports the current and previous two runner minor versions during a rolling upgrade. Protocol fields use additive evolution; removing or redefining a field requires a new major version. A runner advertises capabilities, and the scheduler does not lease an incompatible job. Downgrade is tested for the duration of the rollback window.

17.3 Release gates

A control-plane release requires passing automated tests, dependency and image scans, schema compatibility checks, migration rehearsal, canary health, and rollback validation. Database migrations are backward compatible with the previous application version, expand before use, and contract only after the rollback window. Runner-image releases run representative untrusted workloads and verify teardown residue before promotion.

18 Rollout and migration

18.1 Phased delivery

Table 12: Delivery phases and exit criteria

Phase	Scope	Exit criteria	Rollback
0: Foundation	Control plane, one untrusted Linux pool, logs, identity, audit	Threat model approved; end-to-end test; zone failure; no critical findings	Disable event routing
1: Pilot	5–10 low-risk repositories, build/test only	Four weeks within SLO; support playbook; user success rate $\geq 95\%$	Return repositories to legacy
2: Artifacts	Immutable artifacts, cache, provenance, SBOM	Integrity and isolation tests; artifact recovery exercise	Disable cache; retain artifacts
3: Non-production	Staging environments and adapters	Approval, reconciliation, rollback, and policy tests pass	Revoke environment federation

Phase	Scope	Exit criteria	Rollback
4: Production canary	Selected services and protected runner pool	Two successful release cycles per service; incident drill	Route deployment to legacy
5: General availability	Standard repositories and environments	Capacity reserve; SLO dashboard; DR exercise; support readiness	Per-tenant routing switch
6: Legacy retirement	Remaining migration and evidence archive	No active dependencies; signed owner acceptance; recovery export	Time-boxed compatibility lane

18.2 Repository migration

The migration tool translates supported legacy constructs, pins reusable actions, creates an explicit permission set, and produces a semantic diff. Repositories first run in shadow mode without deployment authority. Results, duration, logs, and artifacts are compared for at least five representative runs. Cutover is repository-scoped and reversible through an event-routing flag. Production credentials are enabled only after the owner signs off on evidence and rollback.

18.3 Feature flags and kill switches

Flags are typed, owned, audited, and default safe. Required kill switches can stop new run admission, planning, leasing by pool, secret issuance, artifact publication, or deployments by environment without stopping read-only diagnostics. A global deployment stop fails closed and is exercised quarterly. Flag changes affecting security or production require dual authorization.

19 Acceptance criteria and traceability

General availability is approved only when all mandatory criteria in Table 13 have objective evidence and no open critical or high-risk security finding lacks an approved exception.

Table 13: Acceptance criteria

ID	Criterion	Required evidence
AC-01	Signed source event creates one deterministic run; replay creates no duplicate.	FR-001-004; webhook and planner report
AC-02	A 256-job DAG is planned within objective and scheduled without starvation under mixed tenant load.	FR-003-005; performance report
AC-03	Killing a runner causes safe lease expiry and retry; stale completion cannot finalize.	FR-006, FR-017; chaos trace
AC-04	Forensic inspection finds no cross-job writable residue, token, or secret after teardown.	FR-007-009; isolation report
AC-05	Log disconnect and replay produce ordered output with no silent gap; secret canaries are absent.	FR-009-010; redaction report
AC-06	Corrupted artifact and malicious cache archive are rejected without changing job trust.	FR-011-012; negative tests

ID	Criterion	Required evidence
AC-07	Production policy rejects unsigned, mismatched, vulnerable, or unapproved artifacts.	FR-013-015; policy evidence
AC-08	Cancellation revokes grants and prevents a queued or running deployment from entering the environment.	FR-016; race test
AC-09	All protected actions appear in immutable audit storage with actor and correlation data within five minutes.	FR-018; audit reconciliation
AC-10	API retries with the same idempotency key create one effect and stable response.	FR-019; contract suite
AC-11	Operators quarantine a compromised image and drain its pool without direct database mutation.	FR-020; incident exercise
AC-12	A noisy tenant reaches quota while other tenants remain within queue-latency objective.	FR-005, FR-021; load report
AC-13	An interrupted deployment is reconciled to the observed target state before later promotion.	FR-022; adapter recovery test
AC-14	Zone loss preserves accepted runs; regional exercise meets RPO/RTO with one active lease epoch.	FR-023-024; DR report

20 Risks and design decisions

Table 14: Principal residual risks

Risk	Consequence	Treatment	Owner
Sandbox vulnerability	Host or tenant compromise	Micro-VM isolation, patched kernels, egress controls, dedicated trust pools, rapid image deny and rotation	Security
Supply-chain compromise	Malicious build input or deployed output	Digest pinning, mirrors, signatures, provenance, SBOM, independent policy gate	Platform
Credential exfiltration	Unauthorized environment access	Short-lived federation, narrow claims, deny-by-default egress, anomaly detection	Security
Capacity shock	Long queues and delayed releases	Admission control, fair queuing, warm reserve, quota, multi-pool spillover where safe	SRE
Complex policy	Unexpected denial or accidental privilege	Policy testing, explainable decisions, staged rollout, explicit exceptions	Security

Risk	Consequence	Treatment	Owner
Deployment ambiguity	Duplicate or unsafe target change	Idempotency keys, reconciliation, environment serialization, adapter contracts	Service owner
Regional split brain	Duplicate execution and deployments	External fencing, region epoch, certificate revocation, failover drills	SRE
Cost growth	Unsustainable platform spend	Per-tenant metering, budgets, cache and image optimization, rightsizing feedback	Platform

20.1 Resolved decisions

1. **Ephemeral versus persistent runners:** ephemeral is the default; persistent runners are allowed only for specialized hardware in a dedicated, single-tenant pool with mandatory workspace reset and periodic reimaging.
2. **Push versus pull assignment:** authenticated runner pull with renewable leases avoids inbound access and supports fencing and natural backpressure.
3. **Credential model:** workload identity federation replaces stored cloud keys.
4. **Artifact identity:** digests are authoritative; names and tags are presentation aliases.
5. **Execution semantics:** jobs are at least once at the infrastructure layer; leases, fencing, idempotent adapters, and reconciliation produce safe effects.
6. **Regional operation:** one active scheduling region at a time; recovery is warm standby to keep the split-brain model auditable.

20.2 Deferred choices

The implementation team must record architecture decision records before committing to specific database, message broker, micro-VM runtime, policy engine, artifact store, and telemetry vendor. These choices may vary without changing the contracts and invariants in this specification. Native Windows/macOS isolation, GPU pools, and confidential computing are post-baseline capabilities.

21 Implementation readiness checklist

☐ Product owner, platform owner, security owner, SRE owner, and support rotation are named.

☐ API, event, pipeline schema, runner protocol, and deployment adapter contracts are versioned.

☐ Threat model, data classification, privacy review, and penetration-test scope are approved.

☐ State-machine and lease-fencing proofs are represented in executable tests.

☐ Runner base image, attestation, sandbox, egress, teardown, and quarantine controls are verified.

☐ Workload identity trust rules and environment approval separation are tested end to end.

☐ Logs, artifacts, caches, provenance, SBOM, retention, and legal-hold workflows are operational.

☐ SLOs, dashboards, paging rules, capacity forecasts, and cost allocation are active.

☐ Backup restoration, zone failure, region fencing, and deployment reconciliation are exercised.

☐ Pilot repositories meet exit criteria and retain a tested rollback path.

☐ All acceptance criteria have linked, immutable evidence and accountable sign-off.

A Status and reason codes

Stable reason codes are uppercase namespaced tokens. Messages may evolve; automation uses the code and retryability fields.

Table 15: Representative reason codes

Code	Retryable	Meaning
PLAN_SCHEMA_INVALID	No	Definition failed schema validation
PLAN_DAG_CYCLE	No	Job dependencies contain a cycle
POLICY_DENIED	No	Current policy denied the requested action
APPROVAL_REQUIRED	After approval	Environment requires additional approval
POOL_CAPACITY_WAIT	Yes	Eligible pool currently lacks allocatable capacity
LEASE_EXPIRED	Yes	Runner did not renew before deadline
LEASE_FENCED	No	Attempt no longer owns the authoritative fencing token
RUNNER_ATTESTATION_FAILED	No	Runner identity or measured image violated pool policy
ARTIFACT_DIGEST_MISMATCH	No	Uploaded bytes do not match declared digest
DEPENDENCY_UNAVAILABLE	Yes	Required internal dependency is temporarily unavailable
DEPLOYMENT_UNKNOWN	Reconcile	Target effect is uncertain and must be observed
QUOTA_EXCEEDED	Later	Tenant exceeded an enforced resource or concurrency quota

B Audit event minimum set

The platform emits, at minimum, events for authentication success/failure; role and policy changes; runner registration, attestation, quarantine, and revocation; run creation and cancellation; plan creation; lease grant, expiry, and fencing; secret and identity grant issuance; artifact publication and denial; approval decision and invalidation; deployment start, result, rollback, and reconciliation; administrative override; retention or legal-hold change; audit export health; and regional promotion.

C Normative configuration example

Listing 3: Abbreviated protected runner-pool configuration

```
apiVersion: platform.example/v1
kind: RunnerPool
metadata:
  name: production-deploy
spec:
  platform: linux
```

```yaml
  architecture: amd64
  trustTier: protected
  regions: [primary]
  isolation: microvm
  ephemeral: true
  maxJobsPerRunner: 1
  runnerCertificateTTL: 30m
  jobIdentityTTL: 5m
  imagePolicy:
    channel: stable
    requireSignature: true
    allowedPublishers: [platform-image-builder]
  networkPolicy:
    defaultEgress: deny
    destinations:
      - service: artifact-store
        port: 443
      - service: workload-identity
        port: 443
      - service: deployment-api
        port: 443
  resources:
    minWarm: 2
    maxConcurrentJobs: 50
  exceptions:
    privileged: false
    hostNetwork: false
    hostRuntimeSocket: false
```

D Definition of done

The platform baseline is done when the deployed system conforms to all **MUST** requirements; all acceptance criteria pass in the production-like environment; SLO, security, recovery, and operational evidence is approved by accountable owners; pilot repositories complete the defined observation period; and rollback remains available. Documentation alone, a successful demonstration, or partial component completion does not satisfy this definition.

End of PLAT-SPEC-RUNNER-001 – version 1.0.0