Intent-Aware Authorization for Zero Trust CI/CD

Surya Teja Avirneni
Cloud Platform and Security Architect
IEEE Member, ISC2 Certified, ACM Member
United States

April 2025

Abstract

In Zero Trust CI/CD systems, identity is only the beginning. Once a job or workload is authenticated, a decision must still be made about what actions it is allowed to perform, when, and under what conditions. This paper explores how policy engines embedded within credential brokers can support intent-aware authorization—evaluating not just identity but also justification, timing, and workload context. We describe a control loop model where SPIFFE-issued identities are passed into policy evaluators such as OPA or Cedar, and decisions are made based on runtime context and declared intent. We present architectural patterns that support fine-grained access, human approvals, and token scoping. The paper builds on our earlier work on SPIFFE-based workload identity and credential brokers, and focuses here on the policy logic that enables least-privilege, auditable access. This work is the third in a series on Zero Trust CI/CD.

1 Introduction

Most CI/CD platforms today are still driven by a narrow view of identity—either a cloud role assumed via OIDC federation or a service account mounted into a runner. But access is not just about identity. A secure deployment pipeline must also understand *intent*: what is the purpose of access? Has it been approved? Is it happening in the right context, at the right time?

In previous papers, we established SPIFFE-based identity as a secure alternative to secrets and static credentials [1], and introduced credential brokers that enforce runtime access control based on identity and policy [2]. This paper extends that foundation by focusing on how policy engines can make access decisions that are not only identity-aware, but also intent-aware.

In real-world platforms, deployments often require approvals, SLA considerations, or compliance constraints. Traditional IAM roles and ABAC policies are poorly suited to these cases. They do not evaluate justification, and they rarely operate in real time. This creates an enforcement gap—one that cannot be solved by stronger identity alone.

This paper introduces the concept of *intent-aware authorization*, where policies consume both identity and justification to decide access dynamically. We explain how policy engines like OPA and Cedar can model approvals, time-based constraints, and runtime metadata as part of a control loop. We also show how these decisions can be integrated into CI/CD credential brokers, completing the Zero Trust architecture.

2 What is Intent-Aware Authorization?

In traditional IAM systems, access control decisions are based on predefined role or group mappings. These models—RBAC (Role-Based Access Control) and ABAC (Attribute-Based Access Control)—assume that access privileges can be assigned ahead of time, regardless of when or why access is being requested. In Zero Trust CI/CD, this assumption no longer holds.

Intent-aware authorization adds a critical missing dimension: *why* access is being requested. It treats access as a contextual decision that must account for human approvals, workload metadata, SLA conditions, deployment type, and business justification.

For example, two CI jobs might share the same SPIFFE identity, but only one may have a recorded approval for production deployment. Similarly, emergency patches may require elevated access with a time-bound override token issued by an incident manager.

Why Static IAM Falls Short

Static IAM policies do not consider:

- **Justification context**: Why is access needed? Is it tied to an approved change ticket?

- **Temporal constraints**: Is this deployment occurring during an allowed window?

- **SLA-driven gating**: Does the job relate to a failed availability check?

- **Human-in-the-loop approvals**: Is there an active approval from an SRE or release manager?

Intent-aware authorization brings these dimensions into the policy decision point.

CI/CD Example: Production Deployment Approval

A real-world scenario:

- CI job: `spiffe://ci/org/release`

- Action: Push container to production

- Context:

    - Git commit approved and merged
    - PagerDuty override token attached
    - SLA breach recorded on affected service

- Policy: Allow only if all three conditions are true

This style of policy evaluation cannot be encoded in IAM roles or predefined group-based access models. It requires runtime signals and intent awareness at the point of access.

Relation to ABAC: From Attributes to Intent

ABAC systems traditionally decide access based on a set of attributes about the subject, resource, action, and environment. While this model is flexible, most real-world ABAC implementations use static tags—such as environment = "prod" or team = "billing"—to make decisions. This misses the dynamic nature of CI/CD workflows.

Intent-aware authorization extends ABAC by introducing runtime signals and justification tokens as first-class attributes. For example:

- `input.justification.approval = "change_12345"`

- `input.deployment.window = "offpeak"`

- `input.sla_breach = true`

These attributes are ephemeral, externally validated, and tied to the specific execution context—not hardcoded into the identity or environment.
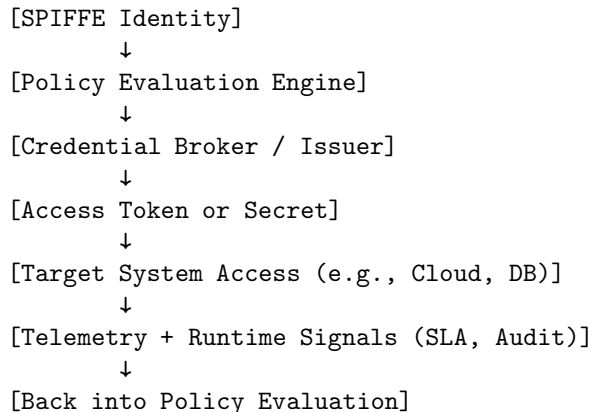
Rather than replacing ABAC, intent-aware systems build on its foundations, adding dynamic attributes, temporal logic, and external state evaluation. This evolution supports precise, policy-driven access in CI/CD pipelines where context and justification matter more than static roles or tags.

3 Control Loops in Authorization

Traditional access control models treat authorization as a one-time gate: a request arrives, a policy is evaluated, and access is either granted or denied. In Zero Trust CI/CD systems, this binary, stateless decision is insufficient.

Intent-aware authorization models access as a *continuous control loop*, where each access decision is influenced by identity, justification, runtime context, and feedback signals. This loop supports revocation, dynamic approval, and real-time policy enforcement—particularly important in CI/CD environments where jobs are ephemeral and operational conditions change rapidly.

Authorization as a Runtime Control Loop:

```
[SPIFFE Identity]
        ↓
[Policy Evaluation Engine]
        ↓
[Credential Broker / Issuer]
        ↓
[Access Token or Secret]
        ↓
[Target System Access (e.g., Cloud, DB)]
        ↓
[Telemetry + Runtime Signals (SLA, Audit)]
        ↓
[Back into Policy Evaluation]
```

This structure formalizes intent-aware enforcement as a feedback-driven system. Each access request is evaluated based on real-time conditions, and access may be revoked or denied in subsequent iterations if policy violations or environmental changes are detected.

Contrasting Static and Loop-Based Authorization

In traditional IAM models (RBAC/ABAC), access is granted through predefined mappings:

```
[Job Identity] → [IAM Role] → [Static Access Grant]
```

In intent-aware systems, access is granted only after dynamic, real-time evaluation:

```
[Job Identity + Context] → [Dynamic Policy Evaluation] → [Just-in-Time Access]
```

This distinction is crucial in Zero Trust CI/CD: jobs are short-lived, policies must reflect current state, and context must evolve across pipeline stages.

Examples of Loop Triggers

Intent-aware control loops respond to signals such as:

- Expiration of an incident override token

- Approval withdrawn on a change ticket

- SLA status changing from "critical" to "stable"

- Time-of-day restrictions or maintenance windows

- Security alert on the source environment

Each of these can trigger re-evaluation, denial of further access, or revocation of previously granted credentials.

Rego (OPA) Example

```
package authz

default allow = false

allow {
  input.spiffe_id == "spiffe://ci/org/deploy-job"
  input.action == "push"
  input.resource == "s3://prod-release-artifacts"
  input.justification.status == "approved"
  within_maintenance_window(input.time)
}

within_maintenance_window(time) {
  time >= "02:00"
  time <= "05:00"
}
```

Sample Input (runtime context):

```
{
  "spiffe_id": "spiffe://ci/org/deploy-job",
  "action": "push",
  "resource": "s3://prod-release-artifacts",
  "justification": { "status": "approved" },
  "time": "02:15"
}
```

Cedar Policy Example

```
permit(
  principal == workload::"spiffe://ci/org/deploy-job",
  action == action::"publish",
  resource == artifact::"prod-artifact"
)
when {
  context.justification == "approved_change_ticket" &&
  context.override == false &&
  context.timestamp >= "2025-04-20T02:00:00Z" &&
  context.timestamp <= "2025-04-20T05:00:00Z"
};
```

Python Broker Simulation (Control Loop)

```
def evaluate_access(input_data):
    if input_data["spiffe_id"] != "spiffe://ci/org/deploy-job":
        return False
    if input_data["justification"]["status"] != "approved":
        return False
    return is_within_window(input_data["time"])
```

This code simulates runtime re-evaluation on every access request. If conditions change, access is denied—even for a previously valid SPIFFE ID.

Feedback-Driven Authorization

By treating policy as a loop—not a static rule set—CI/CD systems gain:

- **Fine-grained revocation** of active credentials

- **Continuous enforcement** based on live environment state

- **Observability-aware gating**, tying access to monitoring or audit conditions

These loops enforce Zero Trust principles: *never trust, always verify*—and reverify with every credential issuance.

4 Revocation and Time-Bound Access

Access in Zero Trust CI/CD must be revocable—not just based on token expiration but based on runtime policy evaluation. Unlike traditional systems where access is granted once and assumed valid for its duration, intent-aware brokers must continuously re-evaluate whether access is still justified.
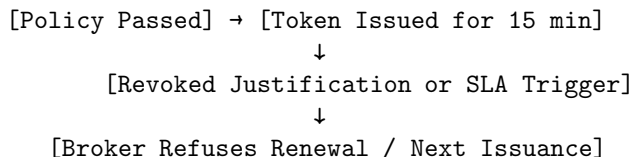
The Challenge with JWTs and Certificates

JWTs (such as SPIFFE JWT-SVIDs) are designed for self-contained identity, but do not support revocation natively. Once issued, they remain valid until expiry. To mitigate this:

- Brokers must issue JWTs with **short TTLs** (e.g., 5–15 minutes).

- Brokers must **refuse to reissue** new tokens when justification changes.

Certificates (X.509-SVIDs) can technically be revoked using CRLs or OCSP, but these mechanisms are operationally complex and not suitable for short-lived CI workloads. Instead, we favor revocation-by-denial: stop issuing new tokens if policy checks fail.

Intent-Aware Lease Pattern:

```
[Policy Passed] → [Token Issued for 15 min]
                          ↓
       [Revoked Justification or SLA Trigger]
                          ↓
   [Broker Refuses Renewal / Next Issuance]
```

Broker Simulation in Python

```python
def evaluate_access(request):
    if request["justification"] != "approved":
        return None
    if not is_within_window(request["timestamp"]):
        return None
    return issue_token(request["spiffe_id"])

def is_within_window(ts):
    return "02:00" <= ts <= "05:00"
```

Key point: revocation is enforced not by invalidating tokens mid-use, but by refusing further issuance when runtime signals change.

Credential Issuance Example: AWS STS with SPIFFE Identity

A frequent implementation is a broker issuing short-lived AWS credentials using a SPIFFE JWT-SVID and the `AssumeRoleWithWebIdentity` API. This allows SPIFFE identities to interoperate with cloud-native identity systems in a secure, revocable way.

Broker Flow:

```
[CI Job] → [SPIRE issues JWT-SVID]
           → [Broker evaluates policy: justification + time]
              → [Broker calls AWS STS AssumeRoleWithWebIdentity]
                 → [Job receives 15-min credentials]
```

Python Example:

```python
import boto3

def assume_aws_role(jwt_token):
    sts = boto3.client('sts')
    creds = sts.assume_role_with_web_identity(
        RoleArn="arn:aws:iam::123456789012:role/CIProdDeployRole",
        RoleSessionName="ci-session",
        WebIdentityToken=jwt_token,
        DurationSeconds=900
    )
    return creds['Credentials']
```

Security Characteristics:

- Token lifetime is tightly bounded (15 minutes).

- No static secrets or permanent role bindings.

- Future requests can be rejected based on live policy state.

Runtime Signals That Trigger Revocation

- SLA state changes from `normal` to `critical`

- Human override is expired or withdrawn

- Change ticket marked `rejected` or `rollback`

- Deployment window closes

By encoding these signals into broker logic or policy engines like OPA or Cedar, Zero Trust CI/CD environments maintain fine-grained control over workload access.

Summary

Revocation in this architecture is implemented as **denial of further issuance**, rather than attempting to forcibly revoke a token already issued. Combined with short TTLs and tight policy checks, this results in a safer, more controllable authorization flow suited to ephemeral CI/CD workloads.

Next, we explore how human-in-the-loop approval systems can inject intent into these policy decisions via justification tokens and policy metadata.

Sample Justification Token Payload

A justification token may be passed from an external system or stored in a credential broker cache. Here's an example of a signed approval token payload:

```
{
  "token_id": "change-req-2025-112",
  "status": "approved",
  "approver": "release-manager@example.com",
  "issued_at": "2025-04-18T21:05:00Z",
  "expires": "2025-04-19T03:00:00Z",
  "reason": "Emergency patch to fix SLA breach",
  "source": "pagerduty"
}
```

Brokers pass this token (or a reference to it) into the policy engine as part of the evaluation context. It serves as a time-bound, human-issued justification for elevated access.

5 Justification Tokens and Human-in-the-Loop Access

While SPIFFE-based identity establishes *who* is making the request, it does not capture *why*. In high-assurance environments—particularly for production deployments, incident mitigation, or sensitive data access—authorization must include a layer of human intent and approval.

What is a Justification Token?

A justification token represents external authorization metadata that a policy engine can inspect. It could be:

- A signed approval object from a change management system

- A token issued by PagerDuty for an incident response

- A temporary override key generated via Slack, ServiceNow, or internal approval tools

These tokens are passed into the broker context and validated during policy evaluation.

Authorization Flow with Justification:

```
[CI Job]
  ↓
[SPIFFE Identity] + [Justification Token]
  ↓
[Broker → Policy Engine (OPA/Cedar)]
  ↓
[Access Granted or Denied]
```

Cedar Example with Justification

```
permit(
  principal == workload::"spiffe://ci/org/critical-deploy",
  action == action::"deploy",
  resource == cluster::"prod-east"
)
when {
  context.approval == "change-req-2025-112" &&
  context.token.valid == true &&
```

```
    context.token.expires >= now()
};
```

OPA Rego Example

```
package authz

default allow = false

allow {
  input.spiffe_id == "spiffe://ci/org/deploy"
  input.approval.status == "approved"
  input.token.expiry > time.now
}
```

Broker Runtime Example (Python)

```python
def check_approval(input_data):
    token = input_data.get("token", {})
    if token.get("status") != "approved":
        return False
    if token.get("expiry") < datetime.utcnow():
        return False
    return True
```

Human-in-the-Loop Use Cases

- **Production Release Approval**: CI jobs blocked until a release manager issues a signed token.

- **Incident Patching**: PagerDuty generates temporary override tokens during an outage window.

- **Maintenance Windows**: Dev teams submit maintenance plans that generate scoped tokens valid for a time period.

Design Considerations

- Tokens should be signed or stored in an auditable approval registry.

- Brokers must be able to query or verify tokens independently.

- The expiration or status of the token must influence real-time access decisions.
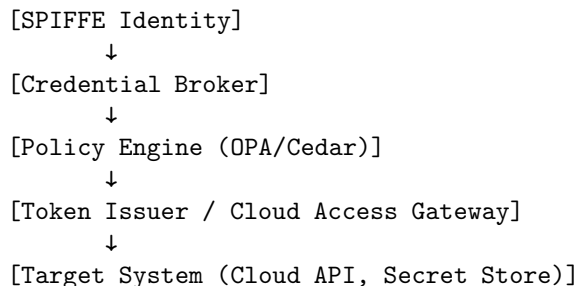
Summary

Justification tokens inject human intent into machine-time authorization. They serve as run-time-bound approval attestations, enabling safe override, emergency access, and sensitive deployment gating. In intent-aware CI/CD systems, they are essential for bridging operational workflows with policy engines.

Next, we explore full-stack architectural patterns that tie together identity, broker logic, and runtime policy enforcement.

6 Architectural Patterns

Intent-aware authorization introduces a modular architecture where identity, policy, and access enforcement are fully decoupled. This separation of concerns supports composability, extensibility, and secure delegation in dynamic CI/CD environments.

Reference Flow:

```
[SPIFFE Identity]
        ↓
[Credential Broker]
        ↓
[Policy Engine (OPA/Cedar)]
        ↓
[Token Issuer / Cloud Access Gateway]
        ↓
[Target System (Cloud API, Secret Store)]
```

This layered design ensures that identity issuance (via SPIFFE), authorization logic (OPA/Cedar), and credential generation (STS, Vault, DB proxies) can evolve independently.

Push vs Pull Enforcement Models

Push Model: Policy bundles are precompiled and pushed to the broker. This offers speed but limits runtime sensitivity.

Pull Model: Brokers evaluate fresh policies with contextual input per request. This model supports dynamic signals like SLA state, justification tokens, and time windows.

We observe that:

- Push models suit high-throughput services (e.g., internal APIs)

- Pull models excel in CI/CD pipelines, where access is short-lived and context-heavy

Delegation Across Tenants and Teams

In platform engineering environments, brokers are often shared across hundreds of teams. Identity scoping and policy boundaries become critical:

```
spiffe://org/team-alpha/deploy
spiffe://org/team-beta/build
```

Per-tenant policies can be enforced by:

- Path-based scoping (prefix match)

- SPIRE selectors (e.g., region, team, SLA class)

- Metadata tags passed to OPA/Cedar

This supports secure multi-tenancy without requiring isolated broker instances per team.

Trust Domains in Multi-Cloud Deployment

Federated SPIFFE domains enable identity verification across environments:

```
[CI Job in GCP]
    ↓
[Broker in AWS]
    ↓
[Federated Trust Bundle]
    ↓
[OPA Policy + AWS STS Credential]
```

Intent-aware architectures allow brokers to issue cloud-native credentials while respecting local policy and external trust anchors.

7 Implementation Notes

OPA: Context-Rich Policy Evaluation

OPA policies are written in Rego and support JSON input. This makes it easy to pass SPIFFE IDs, deployment metadata, justification tokens, and time-based attributes.

OPA supports:

- REST API queries

- Sidecar integration (e.g., with SPIRE Agent)

- Bundle distribution with versioned policy packs

OPA is especially effective in event-driven CI/CD brokers due to its small runtime footprint and clear audit logs.

Cedar: Hierarchical and Graph-Based Policies

Cedar enables fine-grained, object-oriented access control using:

- Entity relationships (e.g., job → service → cluster)

- Conditional clauses (e.g., SLA state, token validity)

- Multi-level delegation

It supports structured ABAC and RBAC hybrid models, making it suitable for environments with shared resources and compliance constraints.

SPIRE Integration and Enrichment

SPIRE supports custom attestors and selector plugins to add:

- Region or cloud provider metadata

- SLA tiers (e.g., bronze, silver, gold)

- Tags like `pipeline_type=release`

These selectors become part of the SPIFFE ID issuance context and are passed to downstream policy systems.

8   Related Work

Standards and Frameworks

NIST SP 800-204 outlines strategies for microservices authentication and dynamic trust establishment. This paper extends those principles to CI/CD pipelines and non-human identity governance [6].

    WIMSE (IETF Working Group) focuses on workload identity interoperability across domains. Our use of SPIFFE and trust bundles builds directly on these efforts.

    SPIFFE Federation and Runtime Brokers were explored in our earlier work [1, 2]. This paper deepens the policy enforcement and intent modeling layers.

Industry Systems and Research

Aembit and other Workload IAM solutions deliver similar runtime policy evaluation, but are often closed-source or cloud-specific. Our architecture uses portable, vendor-neutral building blocks.

    Google Zanzibar and **Cedar** present scalable, policy graph engines. Cedar is especially relevant as it is OSS and tailored for structured authorization across cloud-native platforms.

    Credential Revocation has long been a challenge in X.509 and JWT-based systems. Our use of control loop denial models and time-bound token leasing offers a lightweight, enforceable alternative.

9   Discussion: Threat Model, Limitations, and Evaluation

Threat Model

We assume an adversary with access to compromised CI jobs, rogue users with valid SPIFFE identities, or unauthorized reuse of expired credentials. Brokers and policy engines are trusted execution components. The system assumes availability of external approval systems (e.g., PagerDuty, ServiceNow) and audit trails.

    Potential attack vectors include:

- Forged justification tokens if signing systems are misconfigured.

- Misconfigured policy rules leading to unintended access.

- Denial-of-service or latency issues in high-throughput CI/CD environments.

- Broker compromise, which could lead to policy bypass or unauthorized credential issuance.

Limitations

- Continuous policy evaluation may introduce latency in tight CI job windows.

- External systems must be highly available for justification checks.

- Cedar and OPA policy debugging may be nontrivial for large organizations.

Operational Usability

We note that large organizations benefit from standardizing justification flows through structured tokens, while smaller orgs may prefer human approvals tied to GitHub PR metadata or Slack commands. Failure modes (e.g., approval system down) should default to deny, but configurable fallback logic may be used where appropriate.

Comparison Table: Authorization Models

Table 1: Comparison of Access Control Models

Feature	RBAC	ABAC	Intent-Aware
Role/Tag-Based	Yes	Partial	No
Runtime Context	No	Limited	Yes
Human Approval Integration	No	No	Yes
Revocation Sensitivity	No	Limited	Yes
Justification Evaluation	No	No	Yes
Suitable for CI/CD	Limited	Medium	High

10 Conclusion

Intent-aware authorization is the natural evolution of Zero Trust CI/CD. It moves beyond "who are you?" into "why do you need access, now, under what conditions?" This shift is essential in environments where pipelines are short-lived, access is sensitive, and risk is dynamic.

By combining:

- **SPIFFE-based identity** (verifiable, runtime-issued)

- **Credential brokers** (decoupling identity from access)

- **Policy engines** (OPA, Cedar) with context and justification

...we achieve secure, auditable, and revocable access at runtime.

Key Takeaways:

- Access should be granted not just by role or tag, but based on declared intent.

- Authorization must be modeled as a runtime control loop with feedback.

- Human approvals, SLA triggers, and incident context must influence policy.

- Brokers become enforcement meshes—contextual, continuous, and verifiable.

This paper concludes a three-part series on Zero Trust CI/CD. While identity and access are foundational, we argue that **intent** is the missing dimension—where compliance, governance, and security meet operational velocity.

These patterns extend beyond CI/CD pipelines. They are equally relevant to any platform engineering environment that supports multi-tenant workloads, federated systems, or infrastructure-as-a-service platforms. Embedding intent into policy decisions enables scalable enforcement without central bottlenecks—key to building reliable and secure internal platforms at scale.

References

[1] S. T. Avirneni. *Establishing Workload Identity for Zero Trust CI/CD: From Secrets to SPIFFE-Based Authentication.* arXiv preprint arXiv:2404.12345, 2025.

[2] S. T. Avirneni. *Decoupling Identity from Access: Credential Broker Patterns for Secure CI/CD.* arXiv preprint arXiv:2404.12346, 2025.

[3] P. Spiegel et al. *SPIFFE Specification.* CNCF, 2023. https://spiffe.io/docs/latest/

[4] Open Policy Agent. *Rego Language Guide.* CNCF, 2024. https://www.openpolicyagent.org/docs/latest/policy-lan

[5] Amazon. *Cedar Policy Language Overview*. GitHub, 2024. `https://github.com/cedar-policy`

[6] NIST. *SP 800-204: Security Strategies for Microservices*. 2019. `https://doi.org/10.6028/NIST.SP.800-204`