Establishing Workload Identity for Zero Trust CI/CD: From Secrets to SPIFFE-Based Authentication

Surya Teja Avirneni

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

April 2025

Abstract

CI/CD systems have become privileged automation agents in modern infrastructure, but their identity is still based on secrets or temporary credentials passed between systems. In enterprise environments, these platforms are centralized and shared across teams, often with broad cloud permissions and limited isolation. These conditions introduce risk—especially in the era of supply chain attacks—where implicit trust and static credentials leave systems exposed.

This paper describes the shift from static credentials to OpenID Connect (OIDC) federation, and introduces SPIFFE (Secure Production Identity Framework for Everyone) as a runtime-issued, platform-neutral identity model for non-human actors. SPIFFE decouples identity from infrastructure, enabling strong, portable authentication across job runners and deployed workloads. We show how SPIFFE identities support policy alignment, workload attestation, and mutual authentication. The paper concludes by outlining next steps in enabling policy-based access, forming the basis of a broader Zero Trust architecture for CI/CD.

1 Introduction

Continuous Integration and Continuous Deployment (CI/CD) platforms are essential automation layers in modern software delivery. These systems orchestrate privileged workflows—building, testing, and deploying code—but are often treated as anonymous infrastructure actors. Historically, CI/CD pipelines have relied on static secrets, shared service accounts, and injected credentials to interact with downstream systems.

This approach creates significant risk. In many enterprises, CI systems are multi-tenant, centrally managed platforms with broad access to cloud resources. A compromised CI job can lead to privilege escalation, cross-tenant data access, or unauthorized production changes. The rise of software supply chain attacks—such as dependency hijacking or malicious release tampering—further exposes the weakness of identity assumptions in pipeline automation.

Zero Trust architecture requires strong identity and continuous verification for all entities, including non-human actors. However, common patterns such as OIDC federation remain tightly bound to CI platform metadata and are difficult to federate or audit across environments. SPIFFE (Secure Production Identity Framework for Everyone) addresses this by issuing cryptographically verifiable identities to workloads at runtime, based on attested selectors such as container metadata, service accounts, or host attributes.

This paper focuses on the application of SPIFFE to CI/CD workflows. We examine the evolution from secrets and OIDC federation to runtime-issued identity, describe how SPIFFE integrates with ephemeral jobs and target workloads, and demonstrate how this model enables fine-grained, policy-driven authentication across the delivery lifecycle. In later sections, we preview how these identities can be used for access control, justification-aware authorization, and intent-based governance.

As enterprises increasingly adopt Zero Trust principles, the management of Non-Human Identities (NHIs)—such as CI/CD runners, service accounts, and automation agents—has become a top priority. Gartner recently highlighted NHI management as a 2025 strategic trend, underscoring the need for policy-aware identity issuance and governance for ephemeral workloads [5].

2 Historical Approaches to CI/CD Authentication

2.1 Static Secrets

Legacy authentication in CI/CD systems was centered around long-lived credentials, often stored as encrypted pipeline variables or pulled from external secrets managers like HashiCorp Vault. These credentials included cloud IAM tokens, API keys, and service account passwords, all of which were injected into jobs at runtime. While operationally simple, this model suffers from multiple shortcomings:

- **Over-permissioning:** Credentials are frequently scoped for broad access, violating least privilege.

- **Lack of traceability:** Pipelines reuse identities across environments, branches, and repositories, making attribution difficult.

- **Inflexibility:** Secret rotation or revocation requires manual intervention across pipelines and teams.

- **No contextual awareness:** Secrets cannot express why access is needed or under what justification.

Even with dynamic secrets from tools like Vault, the authentication model remains static. Workloads and jobs are not attested at runtime, and the identity tied to a credential does not change with the pipeline's context. In multi-tenant environments, shared runners and static roles can lead to credential leakage or unintended cross-tenant access.

Example: Injected static credentials

```
export AWS_ACCESS_KEY_ID="AKIA***********"
export AWS_SECRET_ACCESS_KEY="****************"
```

These secrets often persist in logs or shell history and are difficult to audit. The limitations of this model led to the rise of identity federation approaches like OpenID Connect (OIDC).

2.2 OIDC Federation

OpenID Connect (OIDC) has become a widely adopted approach for CI/CD-to-cloud authentication. Instead of injecting secrets, CI platforms like GitHub Actions, GitLab CI, and Azure DevOps issue ephemeral tokens containing signed claims about the job context. Cloud providers validate these tokens and issue short-lived credentials using trust relationships like AWS STS, GCP Workload Identity Federation, or Azure federated credentials.

Example: GitHub Actions to AWS STS

```
{
  "Effect": "Allow",
  "Principal": {
    "Federated": "arn:aws:iam::ACCOUNT_ID:oidc-provider/token.actions.githubusercontent.com"
  },
  "Action": "sts:AssumeRoleWithWebIdentity",
  "Condition": {
    "StringLike": {
      "token.actions.githubusercontent.com:sub": "repo:org/repo-name:ref:refs/heads/main"
    }
  }
}
```

Each repo or workflow must be explicitly mapped to a role. As environments scale, policy drift and management overhead increase.

Example: Decoded GitHub-issued OIDC token (excerpt)

```
{
  "sub": "repo:org/my-service:ref:refs/heads/main",
  "aud": "sts.amazonaws.com",
  "job_workflow_ref": "org/my-service/.github/workflows/deploy.yml@refs/heads/main",
  "repository": "org/my-service",
  "sha": "6cffe1e4b9ea91dd0a189..."
}
```

GitHub uses `job_workflow_ref` for unique job identity. GitLab, in contrast, provides `CI_JOB_ID`, a numeric value. Azure federated credentials are scoped to GitHub environment bindings. These variations reflect the lack of standardization across platforms.

Despite their improvements, OIDC federation models present challenges:

- **Policy sprawl:** Role mappings for each job or repo become hard to scale and audit.

- **Claim fragility:** Tokens are tied to claims like `sub` and `ref`, which break with pipeline restructuring.

- **Tooling gaps:** Cloud providers lack visibility into federated identity usage or access posture.

- **Lack of abstraction:** No unified standard exists across cloud identity federation models.

OIDC federation improves CI/CD identity posture within platform-bound contexts but falls short in multi-cloud and federated environments. It does not offer a portable, runtime-verifiable identity primitive that decouples trust from the platform itself. This is the problem SPIFFE aims to solve.

3 Limitations of Current Patterns

Although OIDC federation and secret management solutions have improved CI/CD security, they still fall short of delivering verifiable, runtime-scoped identity that aligns with Zero Trust principles. The limitations of current authentication models become more visible in multi-cloud and enterprise environments.

- **Static Role Bindings:** Federated trust relationships rely on pre-defined mappings between workflows and cloud IAM roles. These bindings are brittle, hard to maintain, and difficult to scale as teams and pipelines grow.

- **Lack of Intent Awareness:** OIDC tokens and injected credentials cannot express *why* access is requested. This makes it difficult to enforce justification-based policies or perform meaningful audit correlation.

- **No Approval Integration:** Traditional pipelines lack hooks for human or automated approval gating that ties into authentication. This breaks the link between access requests and organizational policy enforcement.

- **Poor Federation Across Clouds:** Each cloud provider implements OIDC federation differently, with unique claims and trust models. This fragmentation makes it hard to establish consistent access patterns across hybrid or multi-cloud architectures.

- **Unclear Separation of Identity and Authorization:** In many systems, identity and access rights are bundled together. This tight coupling limits flexibility and prevents enforcement of fine-grained authorization policies.

These gaps create an opportunity for a stronger identity foundation—one that can issue workload identities at runtime, travel with the job or workload, and support policy-driven access control across environments. SPIFFE addresses this need directly.

These shortcomings also reflect gaps noted in NIST SP 800-204, which calls for runtime identity and authorization in microservices-based systems [6]. SPIFFE extends these recommendations to the CI/CD layer, bringing verifiable, workload-scoped identity to modern pipelines.

4 SPIFFE as a Standard for Workload Identity

The Secure Production Identity Framework for Everyone (SPIFFE) defines a set of open standards for issuing, representing, and verifying identities for workloads. Unlike traditional identity systems that rely on static credentials or platform-bound tokens, SPIFFE enables the issuance of cryptographically verifiable identity documents at runtime, based on the workload's environment and attestation data.
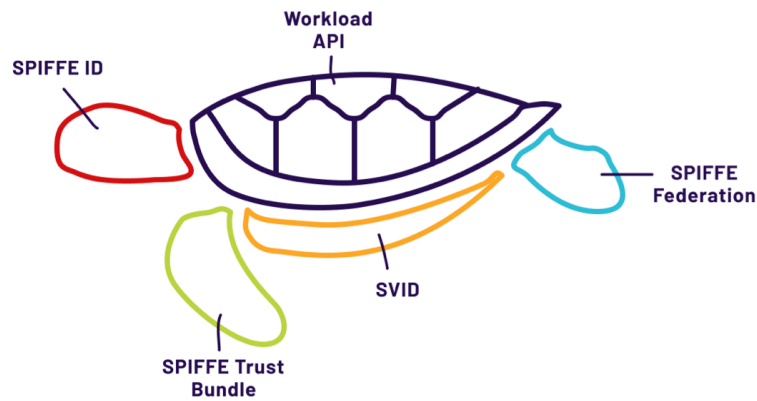
SPIFFE consists of five parts:

Figure 1: The five core components of the SPIFFE identity model. Source: *Solving the Bottom Turtle*, licensed under CC BY 4.0.

Figure 1 provides a visual overview of the SPIFFE identity model. While this paper does not cover each component in depth, the diagram serves as a useful orientation for readers interested in exploring the full specification and implementation landscape. SPIFFE defines five fundamental elements that work together to establish secure workload identity:

- **SPIFFE ID:** A URI-formatted identifier that uniquely names a workload or service within a trust domain.

- **SVID (SPIFFE Verifiable Identity Document):** A cryptographically verifiable document—X.509 or JWT—that proves possession of a SPIFFE ID.

- **Workload API:** A node-local interface through which workloads retrieve their identities securely at runtime, without needing to authenticate first.

- **Trust Bundle:** A collection of root or intermediate CA public keys used to validate SVIDs issued by a SPIFFE authority.

- **Federation:** A trust model that allows multiple SPIFFE trust domains to interoperate by exchanging trust bundles, enabling cross-domain workload authentication.

4.1   What Is a SPIFFE ID?

A SPIFFE ID is a unique, URI-based identity assigned to a workload. It follows the format:

`spiffe://<trust-domain>/<path>`

For example: `spiffe://org.example/frontend/build-runner`

The trust domain defines the administrative and cryptographic boundary for SPIFFE identity issuance. Each workload receives a SPIFFE ID that reflects its logical role, and this identity is stable across restarts, deployments, or network changes.
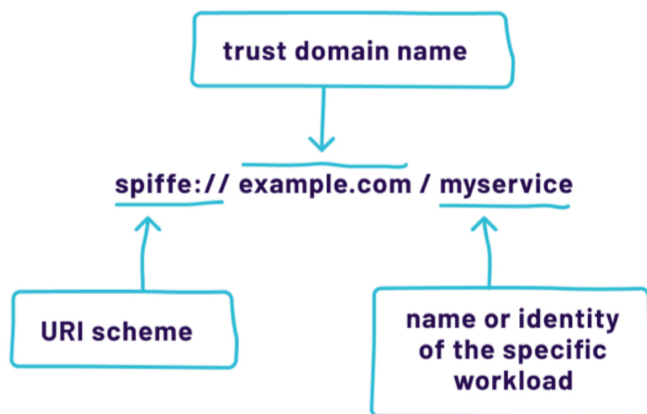

Figure 2: Structure of a SPIFFE ID and its mapping to workloads within a trust domain. Source: *Solving the Bottom Turtle*, licensed under CC BY 4.0.

Figure 2 illustrates the structure of a SPIFFE ID and how it maps to workloads within a trust domain.

4.2   Workload Attestation

In SPIFFE, identity is not granted statically or manually. Instead, every workload must undergo *attestation*—a process by which the SPIRE Agent verifies environmental attributes before issuing a SPIFFE ID. This ensures that identities are tied to real, verifiable conditions at runtime.

SPIRE performs two layers of attestation:

- **Node attestation:** Verifies the identity of the host running the SPIRE Agent, using metadata from the cloud instance, TPMs, or other platform-bound mechanisms.

- **Workload attestation:** Validates attributes such as Kubernetes service accounts, container image hashes, or filesystem paths.

Attestation is handled locally by the SPIRE Agent using **selectors**, which extract metadata from the host platform. These selectors can include:

- Kubernetes service account, namespace, and pod labels

- Executable binary path

- Docker container runtime metadata

- Host-specific data such as EC2 tags or node UUIDs

The SPIRE Server uses these selectors to match the workload against registration policies. If the selectors match a valid entry, the Agent issues an SVID through the Workload API.
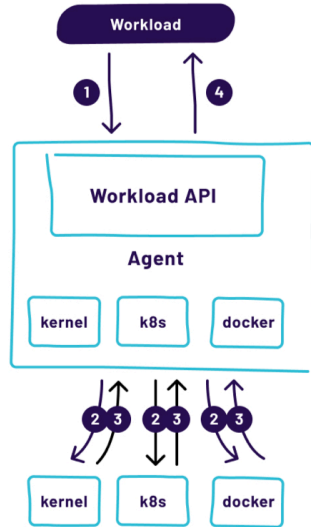

Figure 3: Workload attestation flow in SPIRE using platform-specific plugins and the Workload API. Source: *Solving the Bottom Turtle*, licensed under CC BY 4.0.

Figure 3 shows how the Agent performs attestation using different plugins—such as kernel, Kubernetes, or Docker—to derive selectors. Once verified, it returns a valid identity through the Workload API.

This dynamic, plug-in-based attestation is critical for ephemeral environments like CI/CD pipelines, where jobs must be verified and issued identity at runtime, without hardcoded trust or pre-registered secrets.

4.3   Verifiable Workload Identity: SVIDs and SPIRE

SPIFFE Verifiable Identity Documents (SVIDs) are the cryptographic credentials used to prove workload identity. These are issued dynamically at runtime and come in two main formats:

- **X.509-SVIDs:** Used for mutual TLS (mTLS) authentication in workload-to-workload communication. The SPIFFE ID is embedded as a URI in the certificate's Subject Alternative Name (SAN) field.

- **JWT-SVIDs:** Used for OIDC-compliant identity federation in cloud environments, HTTP-based APIs, or service meshes that consume bearer tokens.

For example, a JWT-SVID used in a cloud authentication flow may look like:

```
{
  "aud": "sts.amazonaws.com",
  "sub": "spiffe://ci/org/deploy-job",
  "exp": 1717200000,
  "iat": 1717196400,
  "iss": "spiffe://org.example"
}
```

This token conforms to OIDC expectations and can be passed to identity-aware services like AWS STS using the `AssumeRoleWithWebIdentity` API to obtain scoped credentials at runtime.

To use a SPIFFE-issued JWT with AWS, an IAM role must be configured to trust the SPIRE trust domain as an identity provider. A sample trust policy might look like:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::ACCOUNT_ID:oidc-provider/spire.example.org"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "spire.example.org:sub": "spiffe://ci/org/deploy-job"
        }
      }
    }
  ]
}
```

This OIDC-compliant policy ensures that only jobs with the correct SPIFFE ID—issued by SPIRE at runtime—can assume the role. It enables least privilege access without relying on long-lived credentials or CI platform-specific tokens.

SPIRE (SPIFFE Runtime Environment) automates this lifecycle—handling attestation, SVID issuance, rotation, and trust bundle distribution. It consists of a central SPIRE Server and one or more node-local SPIRE Agents that interact with workloads.

4.4 Trust Domains and Federation

SPIFFE introduces the concept of trust domains to isolate administrative boundaries and cryptographic trust. Cross-domain federation is enabled through trust bundles — collections of public keys that allow workloads in one domain to authenticate workloads in another, without requiring global trust anchors.

Trust bundles define the root CA material used to validate SVIDs across SPIFFE deployments. Federation enables trust between independently managed domains—critical in CI/CD pipelines that span vendors, clouds, or business units.

This trust model makes it possible for CI/CD jobs, build runners, and deployed workloads to participate in identity federation securely — even across organizational or cloud boundaries.

SPIFFE federation follows IETF draft-ietf-spiffe-federation v07 [2], using X.509 bundle exchange over HTTPS with mTLS authentication.

4.5 Comparison with OIDC Federation

Unlike OIDC federation, SPIFFE operates entirely within the workload runtime and is not tied to a CI/CD provider or cloud platform. Identity is issued based on attested attributes like the workload's binary path,

Kubernetes service account, or node metadata. This enables strong identity guarantees, even in ephemeral environments like CI jobs or containers.

OIDC tokens are scoped to platform-defined job metadata and require pre-configured IAM roles and trust policies. SPIFFE IDs, in contrast, are portable, runtime-verifiable, and usable across heterogeneous infrastructure. They enable identity-centric policy enforcement and eliminate reliance on static or platform-bound credentials.

5 Applying SPIFFE in CI/CD Workflows

CI/CD systems often operate with broad privileges but lack strong, verifiable identity. Secrets and static roles are still common, and OIDC federation is tied to the source platform. SPIFFE offers a runtime-issued identity model that improves isolation, traceability, and policy enforcement across the pipeline.

5.1 Identity for CI Job Runners

Ephemeral CI jobs—such as GitHub runners or Kubernetes jobs—can be attested at runtime based on selectors like image hashes, service accounts, or host attributes. Once attested, the SPIRE Agent issues a SPIFFE ID that reflects the job's purpose (e.g., `spiffe://ci/org/release-job`).

In multi-tenant CI/CD environments, this pattern is even more powerful. For example, a runner in a centralized CI platform shared across teams might receive an identity such as:

`spiffe://platform.example.org/ci/team-a/release-runner`

This ID clearly reflects the platform boundary (`platform.example.org`), tenant (`team-a`), and workload role (`release-runner`). It enables scoped access control, policy enforcement, and audit tracking without relying on hardcoded roles or credentials.

This pattern works across a variety of environments. SPIRE can issue identities to Kubernetes jobs, GitHub-hosted runners using self-managed agents, or even bare-metal CI agents running on hardened Linux hosts. Identity issuance is driven by attestation plugins, not tied to platform-specific capabilities.

Each SPIRE Agent issues SVIDs based on local attestation, but does not require direct permissions to assume cloud roles. Instead, a job-specific JWT-SVID can be passed to AWS STS (or a similar cloud identity service), which validates the SPIFFE ID via federation and issues short-lived credentials. This enables per-job authorization across accounts or environments—without granting persistent privileges to shared agents or relying on static, cross-tenant IAM roles.

This pattern is especially valuable in multi-cloud or multi-account environments where CI/CD platforms serve multiple teams. Each job receives an isolated identity, mapped to least-privilege permissions via policy, with no need for runners to be bound to privileged service accounts or shared cloud roles.

5.2 Identity for Deployment Targets

Deployed workloads also receive SPIFFE IDs based on their environment. This enables mutual authentication: the CI job has its identity, and the runtime workload has its own. The two can authenticate each other using mTLS or token verification, allowing secure, traceable handoffs.

5.3 Cross-Domain Federation

In multi-team or multi-cloud environments, SPIFFE's trust domain model supports identity federation across boundaries. Workloads and CI jobs from different domains (e.g., `team-a.example.org`, `platform.example.org`) can trust each other through exchanged trust bundles—without breaking isolation or over-permissioning.

This architecture mitigates several common CI/CD risks: compromised runners can no longer impersonate privileged identities; cross-tenant privilege escalation is blocked by scoped trust domains; and credentials are not exposed in logs or artifacts. SPIFFE-based identity reduces the impact of supply chain attacks by enforcing identity at runtime, rather than relying on static job metadata or pre-provisioned tokens.

These identities serve as the basis for just-in-time, policy-driven access decisions, which we explore in the next section.

6 Benefits of SPIFFE-Based Workload Identity in CI/CD

Replacing secrets with SPIFFE-based identity brings several tangible benefits to CI/CD systems—especially those operating in multi-team, multi-cloud, or regulated environments.

First, the system becomes **secrets-free**. There is no need to inject or manage long-lived credentials in job configurations, runner environments, or external secrets managers. This reduces the blast radius of credential leaks and removes the operational burden of secret rotation and lifecycle tracking.

Second, SPIFFE supports **runtime-issued, verifiable identity**. Workloads and jobs are identified based on who they are, not where they run or how they were triggered. This shifts trust from environment-based assumptions to runtime attestation, allowing identity to be tightly bound to selectors like service account, image hash, and host metadata.

Third, SPIFFE IDs are **platform-agnostic and policy-compatible**. They can be consumed by policy engines such as Open Policy Agent (OPA), Cedar, or custom ABAC frameworks to enforce fine-grained access controls. Identity becomes decoupled from the CI provider or infrastructure role mapping, allowing policy logic to focus on intent and function.

Fourth, SPIFFE enables organizations to **enforce least privilege** in CI/CD. Each job can be independently scoped to access a minimal set of resources—cloud APIs, registries, secrets—based on its unique SPIFFE ID. Policies can be driven by workload role rather than by shared execution context or static job labels.

Fifth, SPIFFE facilitates **auditability and access justification**. Each identity is traceable, renewable, and scoped to a runtime context. This supports compliance programs and enables integration with systems that enforce human approvals or SLA-driven access gates.

Finally, SPIFFE IDs form the **foundation for workload-based authorization**. SPIFFE provides identity but does not directly confer access—this role is delegated to a policy-based system that maps identities to short-lived credentials or capabilities. This separation of identity and access is what enables dynamic, Zero Trust enforcement, which we explore in the following section.

These properties lay the groundwork for scalable, just-in-time, and intent-aware identity enforcement in CI/CD workflows.

Table 1 summarizes how SPIFFE compares to traditional CI/CD identity models across key dimensions of trust, scope, and portability.

Table 1: Comparison of authentication models for CI/CD workload identity

Feature	Static Secrets	OIDC Federation	SPIFFE (w/ SPIRE)
Credential Injection	Yes	No	No
Runtime Issuance	No	Partial	Yes
Platform Neutral	No	No	Yes
Identity Portability	Low	Medium	High
Supports Federation	No	Limited	Yes
Tied to Job Context	No	Yes	Yes
Supports mTLS Authentication	No	No	Yes

In addition to improving portability and policy enforcement, SPIFFE also mitigates key security risks commonly encountered in CI/CD environments:

- **Credential Leaks:** Short-lived SVIDs eliminate the need for static secrets in runners or job environments.

- **Privilege Escalation:** Trust domain boundaries restrict identity issuance and prevent cross-tenant impersonation.

- **Replay Attacks:** JWT-SVIDs include expiration, issuer, and audience fields to support bounded, verifiable access.

- **Zero Trust Alignment:** SPIFFE enforces runtime authentication, continuous verification, and least-privilege access by design.

7   From Identity to Access: Enabling Policy-Based Authorization

SPIFFE provides strong identity—but identity alone does not grant access. CI/CD systems must translate that identity into scoped permissions such as access to cloud APIs, internal secrets, or deployment targets. This transition is governed by policy.

A policy-based access system serves as the decision point between a verified identity and a protected resource. When a CI job or workload presents its SPIFFE ID, a policy engine—such as Open Policy Agent (OPA) or Cedar—evaluates whether access should be granted, to which resource, under what conditions, and for how long. This replaces static role bindings with real-time, context-aware decision making.

Example: OPA Rego Policy

```
package authz

default allow = false

allow {
  input.spiffe_id == "spiffe://ci/org/deploy"
  input.action == "write"
  input.resource == "s3://prod-release-artifacts"
}
```

This policy allows write access to a release bucket only for jobs bearing a specific SPIFFE ID. The evaluation is scoped to the identity, resource, and action.

Example: Cedar Policy

Cedar expresses access control as a set of entities, actions, and policies:

```
permit(
  principal == workload::"spiffe://ci/org/deploy",
  action == action::"publish",
  resource == artifact::"release-bucket"
);
```

This policy grants the 'publish' action to the deploy job's identity on the release artifact resource. Unlike role-based systems, both policies enforce access based on verifiable identity, not environmental assumptions.

This architecture enables just-in-time access, audit logging, and runtime enforcement. Access can be gated by human approvals, bound to job metadata, or tied to SLA-compliant runtime conditions.

Formalization

Formally, identity-based access can be represented as:

$$\text{AccessGranted} = f(\text{SPIFFE\_ID}, \text{Context}, \text{Policy})$$

Where *Context* may include workload metadata, time constraints, tags, justification tokens, or prior approvals.

This model supports dynamic access control in CI/CD pipelines where identity is issued at runtime and access decisions are driven by policy. In the next paper, we explore how this model can be extended to include credential issuance, intent validation, and continuous policy supervision.

8   Related Work and Standardization Landscape

SPIFFE and SPIRE are part of a broader effort to standardize non-human identity across distributed systems. This movement intersects with cloud-native access control, Zero Trust architectures, and software supply chain integrity initiatives.

The **SPIFFE/SPIRE project**, a graduated CNCF project, defines the identity primitives used in this paper. SPIRE supports Kubernetes-native attestation, multi-platform workload selectors, and pluggable federation with third-party PKIs. Its architecture makes it a practical implementation of runtime-issued identity.

GitHub Actions and other CI platforms support OIDC-based federation with AWS, GCP, and Azure. These mechanisms reduce reliance on static secrets, but are tightly bound to platform-specific claims and workflow metadata. They lack portability across providers and often do not support multi-domain federation.

The IETF **WIMSE (Workload Identity Management in Secure Environments)** working group is drafting specifications for SPIFFE federation and workload identity portability.[1] These drafts formalize trust bundle exchange and claims-based authorization logic between independent domains.

Other complementary standards include:

- **SCITT (Secure Component Identification and Transparency)**: An IETF initiative to ensure software component trust throughout the supply chain.[2]

- **SLSA (Supply Chain Levels for Software Artifacts)**: A framework for build and release integrity, often used alongside SPIFFE to bind provenance to identity.[3]

Legacy models—such as HashiCorp Vault, Kubernetes secrets, and static IAM roles—remain common but assume implicit trust and are ill-suited for federated, multi-tenant environments. These models lack runtime verification and do not scale to ephemeral CI/CD workloads or cross-cloud governance.

SPIFFE distinguishes itself by acting as a vendor-neutral, runtime identity layer that can integrate with OIDC, credential brokers, or SBOM frameworks. These developments point toward a converging architecture: portable, verifiable identity as the foundation for cloud-native authorization and Zero Trust delivery pipelines.

Recent industry adoption further highlights the shift toward verifiable, brokered workload access. In April 2024, Snowflake publicly described its implementation of a Workload Identity and Access Management (Workload IAM) model using Aembit.[4] Their approach mirrors the principles explored in this paper: runtime identity, dynamic credential issuance, and policy-based access across cloud and SaaS environments. SPIFFE serves as a foundational layer in such systems, decoupling identity issuance from cloud platforms and enabling secretless authentication via policy brokers.

9   Conclusion and Future Directions

In modern CI/CD workflows, identity is the foundation for access, compliance, and trust. Yet most systems still rely on secrets, environment-based assumptions, or tightly coupled OIDC tokens issued by the platform itself. These models fail to scale across cloud accounts, tenants, or ephemeral workloads.

This paper introduced SPIFFE as a portable, runtime-issued identity model that enables CI jobs and deployed workloads to authenticate securely without relying on static credentials. By decoupling identity from infrastructure, SPIFFE makes it possible to establish mutual authentication, enforce least privilege, and drive access decisions through policy.

We showed how SPIFFE can be applied to ephemeral CI runners, deployment targets, and multi-tenant environments. We explored how workload identity forms the foundation for secretless, policy-driven access via brokers or authorization engines like OPA and Cedar. These patterns not only improve security, but also reduce operational burden, enable automation, and support audit and compliance.

---

[1] https://datatracker.ietf.org/wg/wimse/
[2] https://datatracker.ietf.org/doc/draft-ietf-scitt-architecture/
[3] https://slsa.dev
[4] https://medium.com/snowflake-builders/securing-workload-access-snowflakes-journey-to-workload-iam-1c2b3b5f51f6

As CI/CD pipelines grow more dynamic and cross-organizational, SPIFFE serves as a unifying control-plane primitive. It replaces brittle secrets with cryptographic identities, and shifts access enforcement to policy systems that evaluate who, why, and under what conditions access is allowed.

In future work, we explore how these identities can be used to enforce access through broker systems, dynamic credential issuance, intent-aware governance, and justification-driven workflows. The second paper in this series focuses on credential brokers, and the third on runtime authorization and compliance control loops.

Appendix

A. Example SPIFFE IDs

```
spiffe://org.example/frontend/build-runner
spiffe://platform.example.org/ci/team-a/release-runner
```

These IDs reflect trust domain boundaries and job roles, enabling scoped access and federated identity.

B. Sample SPIRE Registration Entry

```
entry {
  spiffe_id = "spiffe://org.example/frontend/build-runner"
  selector = "k8s_sa:build"
  parent_id = "spiffe://org.example/spire/agent/k8s-node"
  ttl = 3600
}
```

This entry binds a Kubernetes service account to a SPIFFE ID using workload attestation.

C. Comparison Matrix

Feature	Static Secrets	OIDC Federation	SPIFFE (w/ SPIRE)
Credential Injection	Yes	No	No
Runtime Issuance	No	Partial	Yes
Platform Neutral	No	No	Yes
Identity Portability	Low	Medium	High
Supports Federation	No	Limited	Yes
Tied to Job Context	No	Yes	Yes
Supports mTLS Authentication	No	No	Yes

D. Reference Architecture Snippet

- **SPIRE Server:** High-availability deployment, backed by etcd or PostgreSQL

- **SPIRE Agents:** Deployed as Kubernetes DaemonSet or on CI runners

- **Selectors:** k8s_sa, docker_label, aws_iam_tag

- **Trust Federation:** Configured with trust bundle exchange

E. References and Tooling Links

- **SPIFFE Specification:** `https://github.com/spiffe/spiffe`

- **SPIRE Runtime:** `https://github.com/spiffe/spire`

- **SPIFFE IETF Draft:** `https://datatracker.ietf.org/doc/draft-ietf-spiffe-federation/`

- **WIMSE WG @ IETF:** `https://datatracker.ietf.org/wg/wimse/`

- **SLSA Framework:** `https://slsa.dev`

- **Aembit Workload IAM:** `https://aembit.io`

- **Snowflake Builders Blog:** `https://medium.com/snowflake-builders/securing-workload-access-snowflakes-`

References

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

[2] M. B. Jones. *SPIFFE Federation Protocol.* IETF Draft v07, 2024. `https://datatracker.ietf.org/doc/draft-ietf-spiffe-federation/`

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

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

[5] Gartner. *Top Strategic Technology Trends for 2025: Managing Nonhuman Identities.* Gartner Research Note, 2024.

[6] Anand et al. *NIST SP 800-204: Security Strategies for Microservices-based Application Systems.* National Institute of Standards and Technology, 2019. `https://doi.org/10.6028/NIST.SP.800-204`