Workload Identity in Multi System Environments
Internet-Draft
Intended status: Informational
Expires: 8 January 2026

A. Schwenkschuster
SPIRL
Y. Rosomakho
Zscaler

7 July 2025

Workload Identity Practices
draft-ietf-wimse-workload-identity-practices-02

Abstract

This document describes industry practices for providing secure identities to workloads in container orchestration, cloud platforms, and other workload platforms. It explains how workloads obtain credentials for external authentication purposes, without managing long-lived secrets directly. It does not take into account the standards work in progress for the WIMSE architecture [I-D.ietf-wimse-arch] and other protocols, such as [I-D.ietf-wimse-s2s-protocol].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 8 January 2026.

Copyright Notice

Table of Contents

1. Introduction 3
2. Terminology 5
3. Delivery Patterns 5
3.1. Environment Variables 5
3.2. Filesystem 5
3.3. Local APIs 6
4. Practices 6
4.1. Kubernetes 6
4.2. Secure Production Identity Framework For Everyone (SPIFFE) 10
4.3. Cloud Providers 12
4.4. Continuous Integration and Deployment Systems 14
5. Security Considerations 15
5.1. Credential Delivery 15
5.1.1. Environment Variables 15
5.1.2. Filesystem 15
5.1.3. Local APIs 15
5.2. Token typing 16
5.3. Custom claims are important for context 16
5.4. Token lifetime 16
5.5. Workload lifecycle and invalidation 16
5.6. Proof of possession 17
5.7. Audience 17
6. IANA Considerations 17
7. Acknowledgements 17
8. References 17
8.1. Normative References 17
8.2. Informative References 18
Appendix A. Variations 20
A.1. Direct access to protected resources 20
A.2. Custom assertion flows 20
Appendix B. Document History 20
Contributors 22
Authors' Addresses 22

1. Introduction

Just like people, the workloads inside container orchestration systems (e.g. Kubernetes) need identities to authenticate with other systems, such as databases, web servers, or other workloads. The challenge for workloads is to obtain a credential that can be used to authenticate with these resources without managing secrets directly, for instance, an OAuth 2.0 access token.

The common use of the OAuth 2.0 framework [RFC6749] in this context poses challenges, particularly in managing credentials. To address this, the industry has shifted to a federation-based approach where credentials of the underlying workload platform are used to authenticate to other identity providers, which in turn, issue credentials that grant access to resources.

Traditionally, workloads were provisioned with client credentials and use for example the corresponding client credential flow (Section 1.3.4 [RFC6749]) to retrieve an OAuth 2.0 access token. This model presents a number of security and maintenance issues. Secret materials must be provisioned and rotated, which require either automation to be built, or periodic manual effort. Secret materials can be stolen and used by attackers to impersonate the workload. Other, non OAuth 2.0 flows, such as direct API keys or other secrets, suffer from the same issues.

Instead of provisioning secret material to the workload, one solution to this problem is to attest the workload by using its underlying platform. Many platforms provision workloads with a credential, such as a JWT token ([RFC7519]). Cryptographically signed by the platform’s issuer, this credential attests the workload and its attributes.

Figure 1 illustrates a generic pattern that is seen across many workload platforms, more concrete variations are found in Section 4.

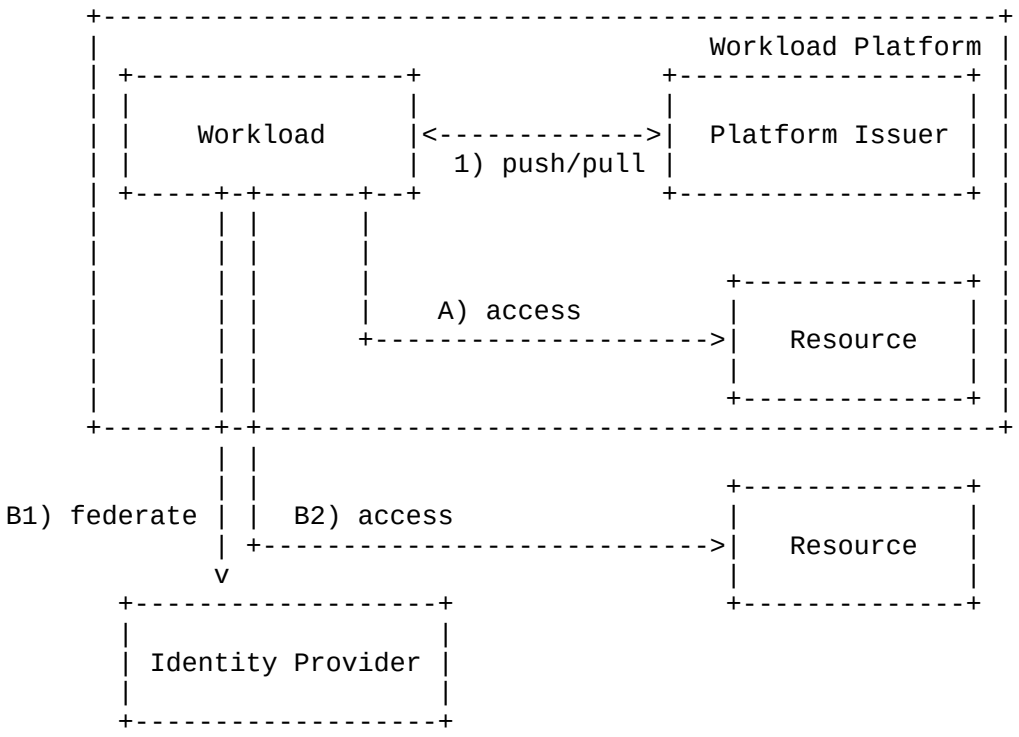

Figure 1: Generic workload identity pattern

The figure outlines the following steps which are applicable in any pattern.

* 1) Platform issues credential to workload. The way this is achieved differs from the platform, for instance, it can be pushed to the workload or pulled by the workload.

* A) The credential can give the workload direct access to resources within the platform or the platform itself (e.g. to perform infrastructure operations)

* B1) The workload uses the credential to federate to an Identity Provider. This step is optional and only needed when accessing outside resources.

* B2) The workload accesses resources outside of the platform and uses the federated identity obtained in the previous step.

Accessing different outside resources may require the workload to repeat steps B1) and B2), federating to multiple Identity Providers. It is also possible that step 1) needs to be repeated, for example in situations where the platform-issued credential is scoped to accessing a certain resource or federating to a specific Identity Provider.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Delivery Patterns

Credentials can be provisioned to the workload by different mechanisms, which each have their own advantages, challenges, and security risks. The following section highlights the pros and cons of common solutions. Security recommendations for these methods are covered in Section 5.1.

3.1. Environment Variables

Injecting the credentials into the environment variables allows for simple and fast deployments. Applications can directly access them through system-level mechanism, e.g., through the env command in linux. Note that environment variables are static in nature in that they cannot be changed after application initialization.

3.2. Filesystem

Filesystem delivery allows both container secret injection and access control. Many solutions find the main benefit in the asynchronous provisioning of the credentials to the workload. This allows the workload to run independently of the credentials update, and to access them by reading the file.

Credential rotation requires a solution to detect soon-to-expire secrets as a rotation trigger. One practice is that the new secret is renewed _before_ the old secret is invalidated. For example, the solution can choose to update the secret an hour before it is invalidated. This gives applications time to update without downtime.

3.3. Local APIs

This pattern relies on local APIs to communicate between the workload and the credential issuer. Some implementations rely on UNIX Domain Sockets (e.g. SPIFFE), loopback interfaces or link-local "magic addresses" (e.g. Instance Metadata Service of cloud providers) to provision credentials. Local APIs offer the capability to re-provision updated credentials. Communication between workload and API allows the workload to refresh a credential or request a different one. This group of solutions relies on network isolation for their security.

Local APIs allow for short-lived, narrowly-scoped credentials. Persistent connections allow the issuer to push credentials.

This pattern also requires client code, which introduces portability challenges. The request-response paradigm and additional operational overhead adds latency.

4. Practices

The following practices outline more concrete examples of platforms, including their delivery patterns.

4.1. Kubernetes

In Kubernetes, machine identity is implemented through "service accounts" [KubernetesServiceAccount]. Service accounts can be explicitly created, or a default one is automatically assigned. Service accounts use JSON Web Tokens (JWTs) [RFC7519] as their credential format, with the Kubernetes Control Plane acting as the signer.

Service accounts serve multiple authentication purposes within the Kubernetes ecosystem. They are used to authenticate to Kubernetes APIs, between different workloads and to access external resources. This latter use case is particularly relevant for the purposes of this document.

To programatically use service accounts, workloads can:

* Have the token "projected" into the file system of the workload. This is similar to volume mounting in non-Kubernetes environments, and is commonly referred to as "projected service account token".

* Use the Token Request API [TokenRequestV1] of the control plane. This option, however, requires an initial projected service account token as a means of authentication.

Both options allow workloads to:

* Specify a custom audience. Possible audiences can be restricted based on policy.

* Specify a custom lifetime. Maximum lifetime can be restricted by policy.

* Bind the token lifetime to an object lifecycle. This allows the token to be invalidated when the object is deleted. For example, this may happen when a Kubernetes Deployment is removed from the server. Note that invalidation is only detected when the Token Review API [TokenReviewV1] of Kubernetes is used to validate the token.

To validate service account tokens, Kubernetes allows workloads to:

* Make use of the Token Review API [TokenReviewV1]. This API introspects the token, makes sure it hasn’t been invalidated and returns the claims.

* Mount the public keys used to sign the tokens into the file system of the workload. This allows workloads to validate a token’s signature without calling the Token Review API.

* Optionally, a JSON Web Key Set [RFC7517] is exposed via a webserver. This allows the Service Account Token to be validated outside of the cluster and access to the actual Kubernetes Control Plane API.

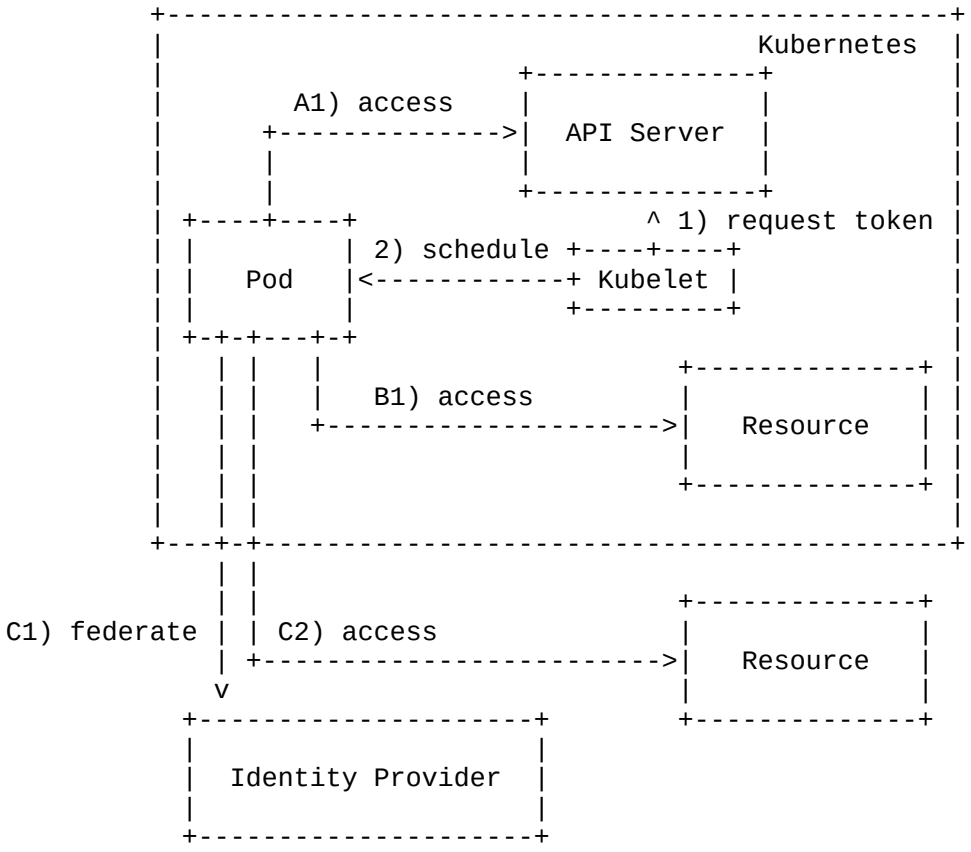

Figure 2: Kubernetes workload identity in practice

The steps shown in Figure 2 are:

* 1) The kubelet is tasked to schedule a Pod. Based on configuration it requests a Service Account Token from the Kubernetes API server.

* 2) The kubelet starts the Pod and, based on the configuration of the Pod, deliveres the token to the containers within the Pod.

Now, the Pod can use the token to

* A) Access the Kubernetes Control Plane, considering it has access to it.

* B) Access other resources within the cluster, for instance other Pods.

* C) Access resources outside of the cluster.

  - C1) The application within the pod uses the Service Account Token to federate to an Identity Provider outside of the Kubernets Cluster.

  - C2) Using the federated identity the application within the Pod accesses resources outside of the cluster.

As an example, the following JSON illustrates the claims contained in a Kubernetes Service Account token.

```
{
  "aud": [  # matches the requested audiences, or the API server's default audiences w
hen none are explicitly requested
    "https://kubernetes.default.svc"
  ],
  "exp": 1731613413,
  "iat": 1700077413,
  "iss": "https://kubernetes.default.svc",  # matches the first value passed to the --
service-account-issuer flag
  "jti": "ea28ed49-2e11-4280-9ec5-bc3d1d84661a",  # ServiceAccountTokenJTI feature mus
t be enabled for the claim to be present
  "kubernetes.io": {
    "namespace": "my-namespace",
    "node": {  # ServiceAccountTokenPodNodeInfo feature must be enabled for the API se
rver to add this node reference claim
      "name": "127.0.0.1",
      "uid": "58456cb0-dd00-45ed-b797-5578fdceaced"
    },
    "pod": {
      "name": "my-workload-69cbfb9798-jv9gn",
      "uid": "778a530c-b3f4-47c0-9cd5-ab018fb64f33"
    },
    "serviceaccount": {
      "name": "my-workload",
      "uid": "a087d5a0-e1dd-43ec-93ac-f13d89cd13af"
    },
    "warnafter": 1700081020
  },
  "nbf": 1700077413,
  "sub": "system:serviceaccount:my-namespace:my-workload"
}
```

Figure 3: Example Kubernetes Service Account Token claims

4.2. Secure Production Identity Framework For Everyone (SPIFFE)

The Secure Production Identity Framework For Everyone, also known as SPIFFE [SPIFFE], is a Cloud Native Computing Foundation (CNCF) project that defines a "Workload API" to deliver machine identity to workloads. Workloads can retrieve either X.509 certificates or JWTs. How workloads authenticate on the API is not part of the document. It is common to use platform metadata from the operating system and the workload platform for authentication to the Workload API.

SPIFFE refers to the JWT-formatted credential as a "JWT-SVID" (JWT - SPIFFE Verifiable Identity Document).

Workloads are required to specify at least one audience when requesting a JWT-SVID from the Workload API.

For validation, SPIFFE offers:

* A set of public keys encoded in JWK format that can be used to validate JWT signatures. In SPIFFE this is referred to as the "JWT trust bundle".

* A validation method on the Workload API to validate JWT-SVIDs.

Additionally, many SPIFFE deployments choose to separately publish the signing keys as a JWK Set on a web server to allow validation where the Workload API is not available.

The following figure illustrates how a workload can use its JWT-SVID to access a protected resource outside of SPIFFE.

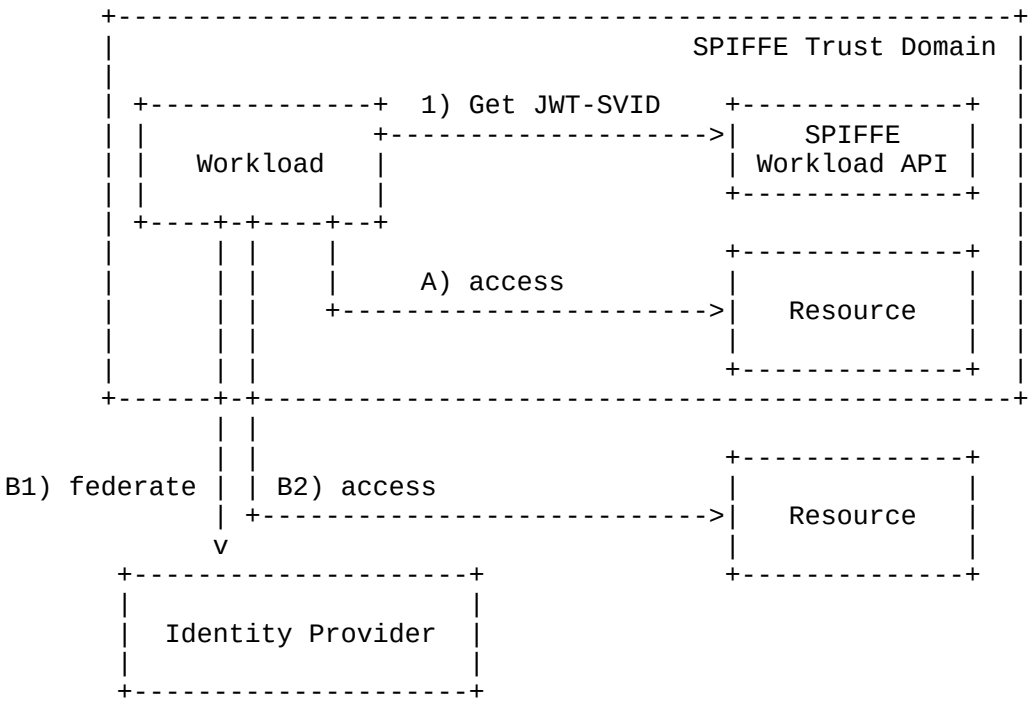

Figure 4: Workload identity in SPIFFE

The steps shown in Figure 4 are:

* 1) The workload requests a JWT-SVID from the SPIFFE Workload API.

* A) The JWT-SVID can be used to directly access resources or other workloads within the same SPIFFE Trust Domain.

* B1) To access resource proctected by other Identity Providers the workload uses the SPIFFE JWT-SVID to federate to the Identity Provider.

* B2) Once federated, the workload can access resources outside of its trust domain.

  TODO: We should talk about native SPIFFE federation. Maybe a C) flow in the diagram or at least some text.

Here are example claims for a JWT-SVID:

```
{
  "aud": [
    "external-authorization-server"
  ],
  "exp": 1729087175,
  "iat": 1729086875,
  "sub": "spiffe://example.org/myservice"
}
```

TODO: write about "iss" in JWT-SVID.

4.3. Cloud Providers

Workload in cloud platforms can have any shape or form. Historically, virtual machines were the most common. The introduction of containerization brought hosted container environment or Kubernetes clusters. Containers have evolved into serverless offerings. Regardless of the actual workload packaging, distribution, or runtime platform, all these workloads need identities.

The biggest cloud providers have established the pattern of an "Instance Metadata Endpoint". Aside from allowing workloads to retrieve metadata about themselves, it also allows them to receive identity. The credential types offered can vary. JWT, however, is the one that is common across all of them. The issued credential allows proof to anyone it is being presented to, that the workload platform has attested the workload and it can be considered authenticated.

Within a cloud provider the issued credential can often directly be used to access resources of any kind across the platform making integration between the services easy and "credentialless". While the term is technically misleading, from a user perspective, no credential needs to be issued, provisioned, rotated or revoked, as everything is handled internally by the platform.

This is not true for resources outside of the platform, such as on-premise resources, generic web servers or other cloud provider resources. Here, the workload firsts need to federate to the Secure Token Service (STS), which is effectively an Identity Provider, to receive an identity of the other cloud. Using this different identity the workoad can then access its resources.

This pattern also applies when accessing resources in the same cloud but across different security boundaries (e.g. different account or tenant). The actual flows and implementations may vary in these situations though.

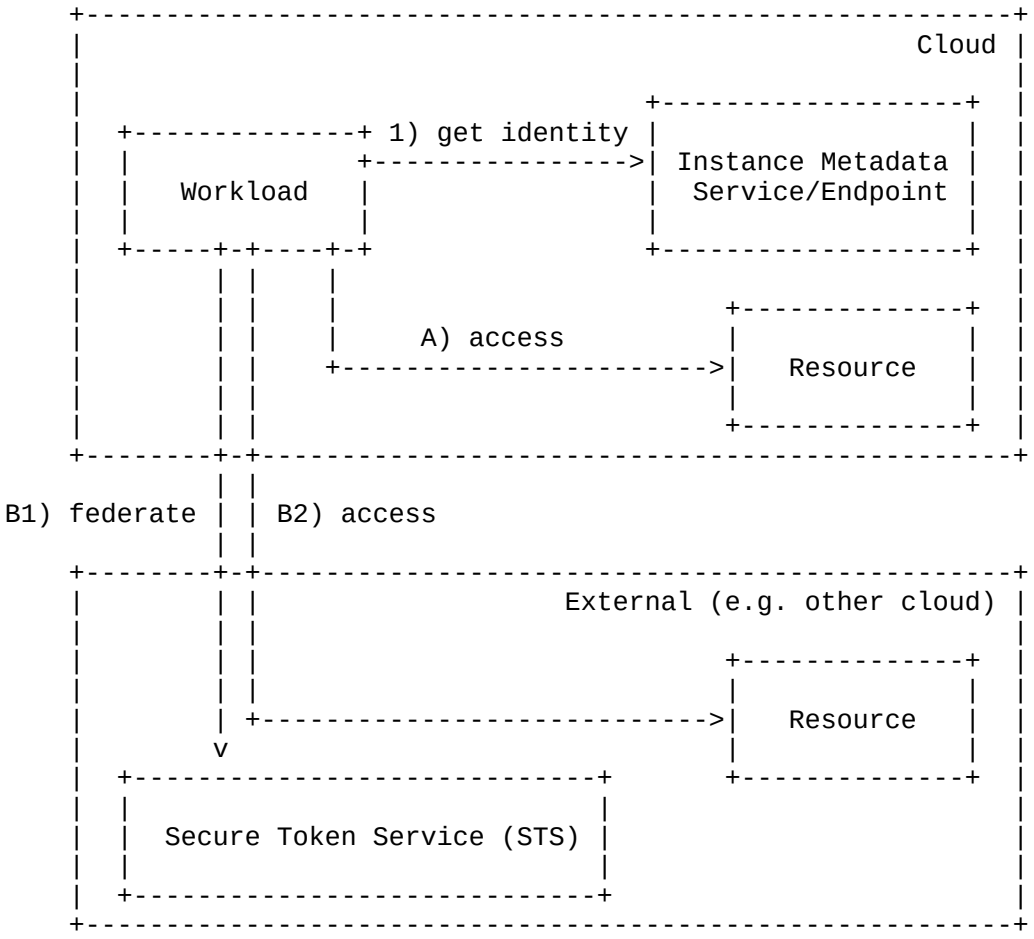

Figure 5: Workload identity in a cloud provider

The steps shown in Figure 5 are:

* 1) The workload retrieves an identity from the Instance Metadata Service or Endpoint. This endpoint exposes an well-known API and is available at well- known, but local, location.

When the workload needs to access a resource within the cloud (e.g. located in the same security boundary; protected by the same issuer as the workload identity):

* A) The workload directly access the protected resource with the credential issued in Step 1.

When the workload needs to access a resource outside of the cloud (e.g. different cloud; same cloud, but different security boundary):

* B1) The workload uses cloud-issued credential to federate to the Secure Token Service of the other cloud/account.

* B2) Using the federated identity the workload can access the resource outside, assuming the federated identity has the necessary permissions.

4.4. Continuous Integration and Deployment Systems

Continuous integration and deployment (CI-CD) systems allow their pipelines (or workflows) to receive an identity every time they run. Build outputs and other artifacts are commonly uploaded to external resources. With federation to external Identity Providers the pipelines and tasks can access these resources.

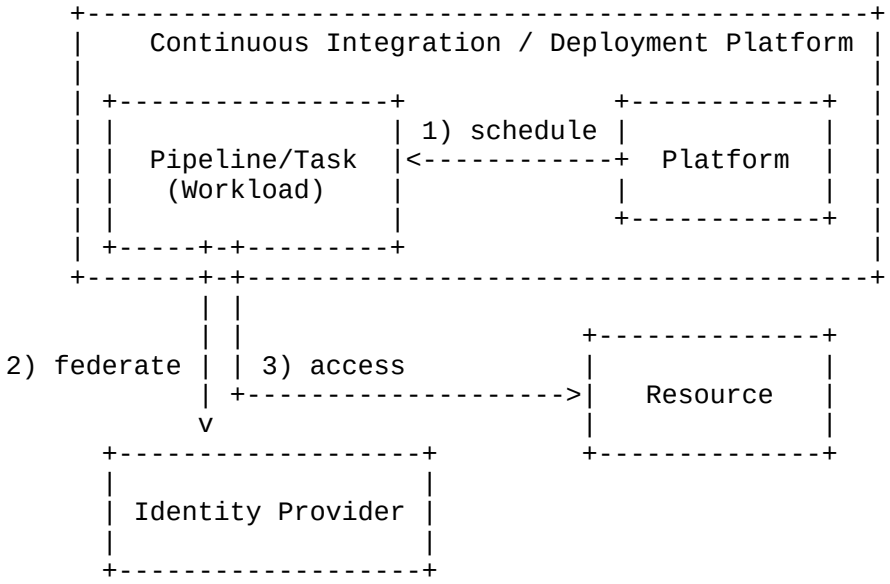

Figure 6: OAuth2 Assertion Flow in a continuous integration/ deployment environment

The steps shown in Figure 6 are:

* 1) The CI-CD platform schedules a workload (pipeline or task). Based on configuration a Workload Identity is made available by the platform.

* 2) The workload uses the identity to federate to a Identity Provider.

* 3) The workload uses the federated identity to access resources. For instance, a artifact store to upload compiled binaries, or to download libraries needed to resolve dependencies. It is also common to access other infrastructure as resources to make deployments or changes.

Tokens of different providers look different, but all contain claims carrying the basic context of the executed tasks, such as source code management data (e.g. git branch), initiation and more.

5. Security Considerations

All security considerations in section 8 of [RFC7521] apply.

5.1. Credential Delivery

5.1.1. Environment Variables

Leveraging environment variables to provide credentials presents many security limitations. Environment variables have a wide set of use cases and are observed by many components. They are often captured for monitoring, observability, debugging and logging purposes and sent to components outside of the workload. Access control is not trivial and does not achieve the same security results as other methods.

This approach should be limited to non-production cases where convenience outweighs security considerations, and the provided secrets are limited in validity or utility. For example, an initial secret might be used during the setup of the application.

5.1.2. Filesystem

* 1) Access control to the mounted file should be configured to limit reads to authorized applications. Linux supports solutions such as DAC (uid and guid) or MAC (e.g. SELinux, AppArmor).

* 2) Mounted shared memory should be isolated from other host OS paths and processes. For example, on Linux this can be achieved by using namespaces.

5.1.3. Local APIs

TODO Reference to attestation might be handy here

5.2. Token typing

Issuers SHOULD strongly type the issued tokens to workload via the JOSE typ header and Identity Providers accepting these tokens SHOULD validate the value of it according to policy. See Section 3.1 of [RFC8725] for details on explicit typing.

Issuers SHOULD use authorization-grant+jwt as a typ value according to [I-D.ietf-oauth-rfc7523bis]. For broad support JWT or JOSE MAY be used by issuers and accepted by authorization servers but it is important to highlight that a wide range of tokens, meant for all sorts of purposes, use these values and would be accepted.

5.3. Custom claims are important for context

Some platform-issued credentials have custom claims that are vital for context and are required to be validated. For example, in a continuous integration and deployment platform where a workload is scheduled for a Git repository, the branch is crucial. A "main" branch may be protected and considered trusted to federate to external authorization servers. But other branches may not be allowed to access protected resources.

Authorization servers that validate assertions SHOULD make use of these claims. Platform issuers SHOULD allow differentiation based on the subject claim alone.

5.4. Token lifetime

Tokens SHOULD NOT exceed the lifetime of the workloads they represent. For example, a workload that has an expected lifetime of an hour should not receive a token valid for 2 hours or more.

For the scope of this document, where a platform-issued credential is used to authenticate to retrieve an access token for an external authorization domain, a short-lived credential is recommended.

5.5. Workload lifecycle and invalidation

Platform issuers SHOULD invalidate tokens when the workload stops, pauses or ceases to exist. How these credentials are invalidated depends on platform authentication mechanisms and is not in scope of this document.

5.6. Proof of possession

Credentials SHOULD be bound to workloads and proof of possession SHOULD be performed when these credentials are used. This mitigates token theft. This proof of possession applies to the platform credential and the access token of the external authorization domains.

5.7. Audience

For issued credentials in the form of JWTs, they MUST be audienced using the aud claim. Each JWT SHOULD only carry a single audience. We RECOMMEND using URIs to specify audiences. See section 3 of [RFC8707] for more details and security implications.

Some workload platforms provide credentials for interacting with their own APIs (e.g., Kubernetes). These credentials MUST NOT be used beyond the platform API. In the example of Kubernetes: A token used for anything else than the Kubernetes API itself MUST NOT carry the Kubernetes server in the aud claim.

6. IANA Considerations

This document does not require actions by IANA.

7. Acknowledgements

The authors and contributors would like to thank the following people for their feedback and contributions to this document (in no particular order): Dag Sneeggen, Ned Smith, Dean H. Saxe, Yaron Sheffer, Andrii Deinega, Marcel Levy, Justin Richer, Pieter Kasselmann, Simon Canning, Evan Gilman and Joseph Salowey.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/rfc/rfc6749>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/rfc/rfc7519>.

[RFC7521] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7521, DOI 10.17487/RFC7521, May 2015, <https://www.rfc-editor.org/rfc/rfc7521>.

[RFC7523] Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, DOI 10.17487/RFC7523, May 2015, <https://www.rfc-editor.org/rfc/rfc7523>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/rfc/rfc8414>.

[RFC8693] Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <https://www.rfc-editor.org/rfc/rfc8693>.

[RFC8707] Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", RFC 8707, DOI 10.17487/RFC8707, February 2020, <https://www.rfc-editor.org/rfc/rfc8707>.

[RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <https://www.rfc-editor.org/rfc/rfc8725>.

8.2. Informative References

[I-D.ietf-oauth-rfc7523bis]
Jones, M. B., Campbell, B., and C. Mortimore, "Updates to Audience Values for OAuth 2.0 Authorization Servers", Work in Progress, Internet-Draft, draft-ietf-oauth-rfc7523bis-01, 23 April 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-rfc7523bis-01>.

[I-D.ietf-wimse-arch]
Salowey, J. A., Rosomakho, Y., and H. Tschofenig, "Workload Identity in a Multi System Environment (WIMSE) Architecture", Work in Progress, Internet-Draft, draft-ietf-wimse-arch-04, 2 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-wimse-arch-04>.

[I-D.ietf-wimse-s2s-protocol]
Campbell, B., Salowey, J. A., Schwenkschuster, A., and Y. Sheffer, "WIMSE Workload to Workload Authentication", Work in Progress, Internet-Draft, draft-ietf-wimse-s2s-protocol-06, 4 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-wimse-s2s-protocol-06>.

[KubernetesServiceAccount]
"Kubernetes Service Account", May 2024, <https://kubernetes.io/docs/concepts/security/service-accounts/>.

[OIDC] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", November 2014, <https://openid.net/specs/openid-connect-core-1_0.html>.

[OIDCDiscovery]
Sakimura, N., Bradley, J., Jones, M. and Jay, E., "OpenID Connect Discovery 1.0 incorporating errata set 2", December 2023, <https://openid.net/specs/openid-connect-discovery-1_0.html>.

[SPIFFE] "Secure Production Identity Framework for Everyone (SPIFFE)", May 2023, <https://github.com/spiffe/spiffe/blob/main/standards/SPIFFE.md>.

[TokenRequestV1]
"Kubernetes Token Request API V1", August 2024, <https://kubernetes.io/docs/reference/kubernetes-api/authentication-resources/token-request-v1/>.

[TokenReviewV1]
"Kubernetes Token Review API V1", August 2024,
<https://kubernetes.io/docs/reference/kubernetes-api/authentication-resources/token-review-v1/>.

Appendix A. Variations

A.1. Direct access to protected resources

Resource servers that protect resources may choose to trust multiple authorization servers, including the one that issues the platform identities. Instead of using the platform issued identity to receive an access token of a different authorization domain, workloads can directly use the platform issued identity to access a protected resource.

In this case, technically, the protected resource and workload are part of the same authorization domain.

A.2. Custom assertion flows

While [RFC7521] and [RFC7523] are the proposed standards for this pattern, some authorization servers use [RFC8693] or a custom API for the issuance of an access token based on an existing platform identity credentials. These pattern are not recommended and prevent interoperability.

Appendix B. Document History

[[To be removed from the final specification]]

-02

* Updated structure, bringing concrete examples back into the main text.

* Use more generic "federation" term instead of RFC 7523 specifics.

* Overall editorial improvements.

* Fix reference of Kubernetes Token Request API

* Prefer the term "document" over "specification".

* Update contributor and acknowledgements sections.

* Remove section about OIDC as it is too specific to a certain implementation.

* Rewrite abstract to better reflect the current content of the document.

-01

* Add credential delivery mechanisms

* Highlight relationship to other WIMSE work

* Add details about token typing and relation to OpenID Connect

* Add security considerations for audience

-00

* Rename draft with no content changes.

* Set Arndt to Editor role.

[as draft-wimse-workload-identity-bcp]

-02

* Move scope from Kubernetes to generic workload identity platform

* Add various patterns to appendix

  - Kubernetes

  - Cloud providers

  - SPIFFE

  - CI/CD

* Add some security considerations

* Update title

-01

* Editorial updates

-00

* Adopted by the WIMSE WG

Contributors

Benedikt Hofmann
Siemens
Email: hofmann.benedikt@siemens.com

Hannes Tschofenig
Siemens
Email: hannes.tschofenig@gmx.net

Edoardo Giordano
Nokia
Email: edoardo.giordano@nokia.com

Authors’ Addresses

Arndt Schwenkschuster
SPIRL
Email: arndts.ietf@gmail.com

Yaroslav Rosomakho
Zscaler
Email: yrosomakho@zscaler.com