ConductorOne

SCIM Provisioning Explained (+ Benefits and Limitations)

The System for Cross-domain Identity Management (SCIM) is an open standard that supports synchronization of user provisioning from one system to another. SCIM enables the exchange of identity data between cloud services and identity providers (IdPs), allowing companies to automate user provisioning across these systems.

For organizations with hundreds or thousands of users, it doesn't make sense to manually add, update, and remove users and security groups from all the different SaaS apps they have in use.

Using SCIM provides the same benefits that a standard for single sign-on (SSO) provides: interoperability across different IdPs, guidance on exactly what needs to be built, existing examples of how to build the implementation, existing tooling, easy integration with most IdPs, and so on.

In this article, you'll learn what SCIM is, why you should use it, common workflows, real-world practical tips, and a little bit about the future of SCIM.

What Are the Benefits of SCIM Provisioning?

Before learning the specifics of SCIM, you should know how it can help your organization and customers.

Scalable User Management

Manually provisioning user accounts across many different pieces of software quickly becomes a huge bottleneck for system administrators. SCIM saves time and effort by automating the management of users between the identity provider (IdP) and destination applications.

For SaaS or B2B organizations, using SCIM means that many different IdPs will be compatible with your system's user provisioning API. Larger enterprise customers that use various IdPs also often require this compatibility.

Enhanced Security and Compliance

Normally, IT administrators have the ability to manage user data and permissions across different systems and services that the organization uses. Some organizations may not like the idea that their IT administrators have full access to provision, modify, and delete any user whenever they want.

With automated SCIM-based user provisioning, you can reduce the number of IT personnel needed to handle user management. This enhances security by minimizing human error and enforcing tighter controls on IT administrator access.

Full User Provisioning Solution

Before SCIM gained popularity and wider use, organizations would often use SAML just-in-time (JIT) provisioning to provision users. This is a great starting point.

However, with SAML JIT provisioning, a user has to sign in to your system before changes from the IDP are synchronized. With SCIM, changes in one system will propagate to other systems *automatically*. For example, deactivating a user in your IDP will propagate immediately to your SCIM-enabled application.

SCIM is a full solution for user provisioning that gives you tighter security, a better user experience, and full provisioning functionality.

How Does SCIM Provisioning Work?

SCIM is an HTTP API standard that defines various HTTP endpoints to synchronize identities between systems. There are two resources that are available to interact with: `User` and `Group`. Each resource has a given schema.

The SCIM standard as a whole has two core parts, consisting of a schema and protocol.

SCIM Schema

The schema defines the types of attributes that are expected and required when implementing your own SCIM API. Here, you'll define email addresses, which tenant the user belongs to, the person's name, and so on. Most SCIM implementations keep the schema simple, but the standard includes many optional attributes.

For example, here are Amazon's required attributes.

For SCIM synchronization to work, every user must have a first name, last name, username, and display name value specified. If any of these values are missing from a user, that user will not be provisioned.

SCIM defines many attributes—here are some of the important attributes you should know about:

- **`id`:** A unique identifier that your system generates for every specific SCIM resource. The value must be unique across all SCIM resources in your system (*eg* across all tenants).
- **`externalId`:** An identifier that comes from the IDP or source system. Usually, this is a SAML identifier or some IDP-specific value. This is optional but can help to link your SCIM users with users who sign in to your system via SSO.

- **`userName:`** Your system's unique identifier for the given user. This is usually an email address, username, or an integer or UUID.
- **`meta:`** An object holding various metadata attributes about the resource.
- **`schemas:`** SCIM has different schemas that define various attributes. For example, there's a core user schema, an enterprise user schema extension that defines even more attributes for users, and a core group schema.

Technically, there are more schemas and ways to add your own. Check out the standard for more on these.

SCIM Protocol

The SCIM protocol defines the various functions (as HTTP endpoints) that you need to implement.

Because the SCIM protocol specification is fairly long, wordy, and covers mostly optional SCIM functionality, it's not the best place to begin learning about how the SCIM protocol works. Instead, you'll start by looking at Amazon's implementation of SCIM.

This is an excellent starting point because Amazon's documentation is concise yet sufficient. Its implementation of SCIM covers only the necessary parts of the standard and nothing more. It's a great real-world implementation example instead of a theoretical blueprint.

Here are all the actions available via its SCIM implementation:

- `CreateUser`
- `GetUser`
- `ListUsers`
- `DeleteUser`
- `PutUser`
- `PatchUser`
- `CreateGroup`
- `GetGroup`
- `ListGroups`
- `DeleteGroup`
- `PatchGroup`
- `ServiceProviderConfig`

Note that **`PutUser`** and **`PatchUser`** can be used to update the **`active`** attribute to activate or deactivate a user. This is different from the **`DeleteUser`** endpoint. See the standard for more.

CreateUser SCIM Endpoint

Here’s what a basic request to the `CreateUser` endpoint might look like:

```
{
  “externalId”: “701984”,
  “userName”: “bjensen”,
  “name”: {
    “formatted”: “Ms. Barbara J Jensen, III”,
    “familyName”: “Jensen”,
    “givenName”: “Barbara”,
    “middleName”: “Jane”,
    “honorificPrefix”: “Ms.”,
    “honorificSuffix”: “III”
  },
  “displayName”: “Babs Jensen”,
  “emails”: [
    {
      “value”: “bjensen@example.com”,
      “type”: “work”,
      “primary”: true
    }
  ],
  “active”: true
}
```

Here’s a sample of a successful response to that request:

```
{
  “id”: “9067729b3d-94f1e0b3-c394-48d5-8ab1-2c122a167074”,
  “externalId”: “701984”,
  “meta”: {
    “resourceType”: “User”,
    “created”: “2020-03-31T02:36:15Z”,
    “lastModified”: “2020-03-31T02:36:15Z”
  },
  “schemas”: [
    “urn:ietf:params:scim:schemas:core:2.0:User”,
    “urn:ietf:params:scim:schemas:extension:enterprise:2.0:User”
  ],
  “userName”: “bjensen”,
  “name”: {
    “formatted”: “Ms. Barbara J Jensen, III”,
    “familyName”: “Jensen”,
    “givenName”: “Barbara”,
    “middleName”: “Jane”,
    “honorificPrefix”: “Ms.”,
    “honorificSuffix”: “III”
  },
  “displayName”: “Babs Jensen”,
  “active”: true,
  “emails”: [
    {
      “value”: “bjensen@example.com”,
      “type”: “work”,
      “primary”: true
    }
  ]
}
```

Notice that the response mostly echoes the attributes sent in the request. However, there are a few new attributes that the destination system generated: `id`, `meta`, and `schemas`.

ServiceProviderConfig SCIM Endpoint

The `ServiceProviderConfig` API endpoint is how an IDP discovers what SCIM functionality your API provides. As there are many optional pieces to SCIM, an IDP may change how it interacts with your API depending on what your API supports. For example, if your API supports HTTP PATCH requests, then the IDP may choose not to use HTTP PUT at all. Some IDPs may ignore this altogether.

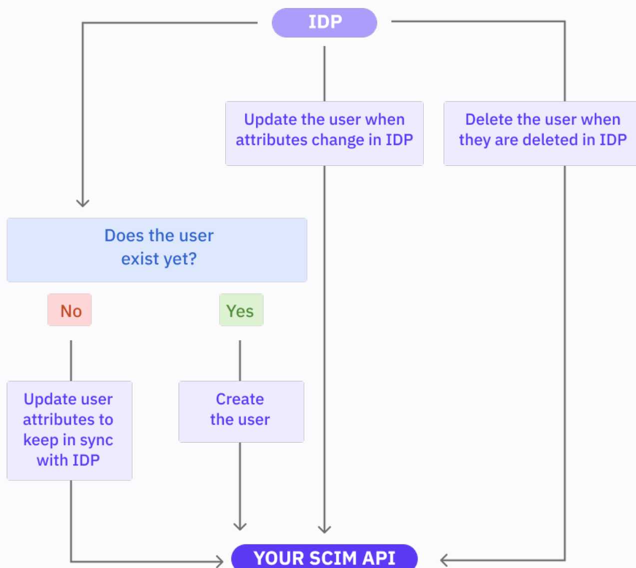
Here is a sample HTTP response from Amazon's documentation that lists other supported and unsupported optional features, including HTTP PATCH operations, bulk operations, the ability to change passwords, and filtering:


```
{
    “schemas”: [
        “urn:ietf:params:scim:schemas:core:2.0:ServiceProviderConfig”
    ],
    “documentationUri”: “https://docs.aws.amazon.com/singlesignon/ latest/
userguide/manage-your-identity-source-idp.html”,
    “authenticationSchemes”: [
        {
            “type”: “oauthbearertoken”,
            “name”: “OAuth Bearer Token”,
            “description”: “Authentication scheme using the OAuth Bearer Token
Standard”,
            “specUri”: “https://www.rfc-editor.org/info/rfc6750”,
            “documentationUri”: “https://docs.aws.amazon.com/singlesignon/
latest/userguide/provision-automatically.html”,
            “primary”: true
        }
    ],
    “patch”: {
        “supported”: true
    },
    “bulk”: {
        “supported”: false,
        “maxOperations”: 1,
        “maxPayloadSize”: 1048576
    },
    “filter”: {
        “supported”: true,
        “maxResults”: 50
    },
    “changePassword”: {
        “supported”: false
    },
    “sort”: {
        “supported”: false
    },
    “etag”: {
        “supported”: false
    }
}
```

Basic Workflow

While each IDP has its own SCIM lifecycle that depends on the specific implementation, the following is the basic lifecycle of a user that is provisioned via SCIM:

- The IDP checks if the user exists already.
  - If the user does not exist, the IDP sends an HTTP POST request to `/scim/Users`
  - If the user exists, the IDP will update your system by sending an HTTP PATCH or PUT request to `/scim/Users` .
- If the user is changed in the IDP, it will send an HTTP PATCH or PUT request to `/scim/Users` .
- If the user is deleted in the IDP, it will send an HTTP DELETE request to `/scim/Users` .



For an example of a real IDP's lifecycle, take a look at the Microsoft AD SCIM provisioning lifecycle. There are other factors involved, such as error handling, quarantining failed synchronizations, lifecycle for user creation, lifecycle for updating a user, lifecycle for deletion, and hard vs. soft delete (eg deactivation vs. full removal).

Limitations and Problems with SCIM Provisioning

Using a standard for user provisioning has many benefits, but SCIM isn't perfect, either.

A Loose Standard

SCIM offers different ways to implement certain operations, resulting in IDPs having different requirements for what your SCIM API needs to support in order to work with them. Ironically, this is what a standard is supposed to avoid!

For example, when updating a `User` and `Group` , some IDPs support HTTP PATCH, while others may only support HTTP UPDATE. Okta uses HTTP PATCH only for user activation/deactivation and password sync. According to the Okta documentation, "All other updates to `User` objects are handled through a PUT method request." Microsoft Entra ID (formerly Azure Active Directory), on the other hand, doesn't support PUT at all and only supports updates via HTTP PATCH.

Nonconformance

Sometimes, an IDP's first implementation didn't follow the SCIM standard. For example, Microsoft Entra has cases where it didn't previously conform to SCIM and has special flags that need to be included with HTTP requests. These flags can change how SCIM behaves and are extra pieces of information you must know about.

Group Members Array

The **`members`** attribute in the **`Group`** resource can cause significant problems. Your SCIM API defines the **`members`** attribute as a JSON array that lists all the users who are members of a given security group. As groups can have many members (possibly thousands) and JSON attributes cannot be paginated, the HTTP responses for large groups can become massive and slow down the API's performance.

Most SCIM APIs have handled this by setting a hard limit on how many records they return in that attribute. Either they return an empty array all of the time or don't return the attribute at all.

For example, the FastFed profile (which you'll learn about later) mandates that the IDP calling your SCIM API must explicitly request the exclusion of this attribute because it's so problematic.

Likewise, Amazon's SCIM API just never returns the attribute.

User Invitation Process

The SCIM standard mandates that the **`CreateUser`** request must return the details of a successful user that was added to the destination system.

When the service provider successfully creates the new resource, an HTTP response SHALL be returned with HTTP status code 201 (Created).

However, many applications have a user provisioning process whereby a user is first invited via their email address and then has to verify that invitation before their account is created.

Do you return a fake user to the IDP on the **`CreateUser`** response? What happens when the IDP queries your SCIM API to sync all users, but some users in your system have not completed the invitation process yet?

This scenario also applies to systems where users are provisioned via asynchronous background processing in the destination system. Does the SCIM API just return a fake user since it doesn't exist yet?

There's no guidance in the SCIM standard, and this is a blind spot for such a common use case. For an example of how to think about this scenario, take a look at how GitHub handles it.

Security Best Practices for SCIM Provisioning

A SCIM API is like any other HTTP API. Bad actors can and will attempt to find vulnerabilities in your API and attempt to attack your system. Therefore, standard HTTP API security concerns also apply to SCIM APIs:

- Are you defending against denial-of-service (DoS) attacks?
- Is your authentication mechanism robust and secure?
- Are you prone to man-in-the-middle attacks?

Authentication

With SCIM, the most likely choice for authentication is a bearer token. In fact, most IDPs only give you this option.

Standard API token security best practices apply here:

- Rotate your tokens
- Make sure your tokens expire
- Ensure your SCIM token only has authorization to access SCIM functionality and no other APIs or functionality in your system (principle of least privilege)
- Limit who can access and generate SCIM tokens
- Consider using a JIT access request tool to limit access to those who can generate SCIM tokens even further

Other Security Considerations

There are still more security hardening measures to be considered. For example, OWASP has published a great guide on REST API security that you can refer to for more information. Here are a few other important security measures to consider for your SCIM APIs:

- Enforce HTTPS for your entire SCIM API
- Implement some form of rate limiting to defend against DoS attacks or aggressive IDPs
- Consider *not* synchronizing passwords via SCIM
  - Some IDPs have the option to push passwords to other systems. Even if your API is using HTTPS and has other protections, is there another way to do what you want? Chances are your customers are using SSO anyway!

The Future of SCIM

SCIM is a relatively young standard. Because of this, the standard is open to revision through real-world usage The organization that manages the SCIM specification, the Internet Engineering Task Force, has documented in RFC 7642 a first round of learnings that detail common SCIM workflows.

There is also a larger specification being worked on that includes various authentication and federation-related standards called FastFed. Part of this standard includes guidance on an opinionated way to implement SCIM.

However, a consequence of SCIM's flexibility is that two providers may find themselves incompatible despite sharing the same protocols.

To deliver the simplified experience that is the goal of FastFed, it is important that two FastFed-enabled providers have confidence that they can interoperate when sharing the same protocols.

In other words, the FastFed SCIM profile is a boiled-down, opinionated approach to implementing SCIM that makes it closer to a true standard. It also defines overarching standards so that various tools like SCIM and SSO can reuse the same authentication and access management functionalities.

Amazon’s SCIM implementation, discussed in this article, serves as an excellent foundation and is based on the FastFed profile.

Conclusion

In this guide, you learned all about SCIM and how it can help guide you in building identity synchronization. By using a common standard, your SCIM API will be compatible with many different IDPs. Although SCIM has some limitations, you can avoid most of them by following solid guidance through either Amazon's SCIM API documentation or the FastFed SCIM profile.

Solid and secure authentication is critical for all HTTP APIs. ConductorOne can help you secure and limit access to your SCIM bearer tokens by limiting who can generate tokens and who can configure your IDP's configuration. By implementing JIT access through ConductorOne, you can harden your security posture around token management and IDP configuration.

By using SCIM and robust security controls and tooling like ConductorOne, you'll be able to build secure user provisioning that delivers massive value to your own organization and your customers.

Want to learn more about our identity security platform for modern workforces?

GET A DEMO

AICPA SOC

ConductorOne team@conductorone.com